D3.2 | End-to-End Security Framework

WP3 – MATRYCS Data Services and Semantic Enrichment Layer

November 2021

www.matrycs.eu

Modular Big Data Applications for Holistic Energy Services in Buildings

Disclaimer

Copyright Message

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Grant Agreement Number	101000158	Acronym	MATRYCS
Full Title	Modular Big Data Applications for Holistic Energy Services in Buildings		
Topic	LC-SC3-B4E-6-2020 | Big data for buildings		
Funding scheme	H2020- IA: Innovation Action		
Start Date	October 2020	Duration	36
Project URL	www.matrycs.eu		
Project Coordinator	ENG		
Deliverable	D3.2: End-to-End Security Framework		
Work Package	WP3 – MATRYCS Data Services and Semantic Enrichment Layer		
Delivery Month (DoA)	M11	Version	1.0
Actual Delivery Date	18/11/2021		
Nature	Report	Dissemination Level	Public
Lead Beneficiary	COMSENSUS		
Authors	Timotej Gale [COMSENSUS], Carolina Fortuna [COMSENSUS], Andrej Čampa [COMSENSUS], Panagiotis Kapsalis [NTUA], Konstantinos Alexakis [NTUA], George Korbakis [NTUA]		
Quality Reviewer(s):	Zhiyu Pan [RWTH], Michal Staša [SEVEN], Pasquale Andriani [ENG]		
Keywords	End-to-End Security, Framework, Service Security, Identity and Access Management, Cloud, Edge		

Preface

MATRYCS focuses on addressing emerging challenges in big data management for buildings with an **open holistic solution** for Business to Business platforms, able to give a competitive solution to stakeholders operating in building sector and to open new market opportunities. **MATRYCS Modular Toolbox,** will realise a holistic, state-of-the-art AI-empowered framework for decision-support models, data analytics and visualisations for Digital Building Twins and real-life applications aiming to have significant impact on the building sector and its lifecycle, as it will have the ability to be utilised in a wide range of use cases under different perspectives:

- Monitoring and improvement of the energy performance of buildings - **MATRYCS-PERFORMANCE**
- Design facilitation and development of building infrastructure - **MATRYCS-DESIGN**
- Policy making support and policy impact assessment - **MATRYCS-POLICY**
- De-risking of investments in energy efficiency - **MATRYCS-FUND**

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Who We Are

	Participant Name	Short Name	Country Code	Logo
1	ENGINEERING – INGEGNERIA INFORMATICA SPA	ENG	IT	
2	NATIONAL TECHNICAL UNIVERSITY OF ATHENS	NTUA	GR	
3	FUNDACION CARTIF	CARTIF	ES	
4	RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN	RWTH	DE	
5	ACCADEMIA EUROPEA DI BOLZANO	EURAC	IT	
6	HOLISTIC IKE	HOLISTIC	GR	
7	COMSENSUS, KOMUNIKACIJE IN SENZORIKA, DOO	COMSENSUS	SL	
8	BLAGOVNO TRGOVINSKI CENTER DD	BTC	SL	
9	PRZEDSIEBIORSTWO ROBOT ELEWACYJNYCHFASADA SP ZOO	FASADA	PL	
10	MIASTO GDYNIA	GDYNIA	PL	
11	COOPERNICO - COOPERATIVA DE DESENVOLVIMENTO SUSTENTAVEL CRL	COOPERNICO	PT	
12	ASM TERNI SPA	ASM	IT	
13	VEOLIA SERVICIOS LECAM SOCIEDAD ANONIMA UNIPERSONAL	VEOLIA	ES	
14	ICLEI EUROPEAN SECRETARIAT GMBH (ICLEI EUROPASEKRETARIAT GMBH)	ICLEI	DE	
15	ENTE PUBLICO REGIONAL DE LA ENERGIA DE CASTILLA Y LEON	EREN	ES	
16	VIDES INVESTICIJU FONDS SIA	LEIF	LV	
17	COMITE EUROPEEN DE COORDINATION DE L'HABITAT SOCIAL AISBL	HOUSING EUROPE	BE	
18	SEVEN, THE ENERGY EFFICIENCY CENTER Z.U.	SEVEN	CZ	

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Contents

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Figures

Tables

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Abbreviation and Acronyms

Acronym	Description
ABAC	Attribute-Based Access Control
ACL	Access Control List
ACM	Access Control Mechanism
AI	Artificial Intelligence
API	Application Programming Interface
AS	Authorization Server
BMC	Baseboard Management Controller
CA	Certificate Authority
CBAC	Context-Based Access Control
CIA	Confidentiality, Integrity and Availability
CLI	Command Line Interface
CSR	Certificate Signing Request
CVE	Common Vulnerabilities and Exposures
D	Deliverable
DR	Disaster Recovery
GUI	Graphical User Interface
HA	High Availability
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IAM	Identity And Access Management
ID	Identifier
IEC	International Electrotechnical Commission
IoT	Internet of Things
IP	Internet Protocol
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JWT	JSON Web Token
MQTT	Message Queue Telemetry Transport
ML	Machine Learning
NFC	Near Field Communication
OS	Operating System
PaaS	Platform as a Service
PCT	Persisted Claims Token
PEP	Policy Enforcement Point
PKI	Public Key Infrastructure
RBAC	Role-Based Access Control
RDBMS	Relational DataBase Management System
REST	Representational State Transfer
RO	Resource owner
RPT	Relying Party Token
SaaS	Software as a Service
SE	Secure Element
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
UBAC	User-Based Access Control
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
WiFi	Wireless Fidelity
WP	Work Package
XSS	Cross-Site Scripting
YAML	YAML Ain't Markup Language

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Executive Summary

The D3.2 – *End-to-End Security Framework* provides a description of the initial version of the MATRYCS End-to-End Security Framework, spanning the complete MATRYCS architecture (*MATRYCS-GOVERNANCE*, *MATRYCS-PROCESSING* and *MATRYCS-ANALYTICS*), according with the MATRYCS high-level reference architecture defined in the deliverable D2.3 – *MATRYCS Reference Architecture for Buildings Data v1.0*.

This document, accompanying the 1st technology releases of other MAYTRYCS layers and modules reported in D3.1 – *MATRYCS-GOVERNANCE (1st technology release)*, D4.1 - *MATRYCS-PROCESSING (1st technology release)*, and D4.4 - *MATRYCS-ANALYTICS and Open Cloud-based Data Analytics Toolbox (1st technology release)*, is mainly focused on background overview and design in addition to preliminary evaluation of the envisaged technological solutions for the End-to-End Security Framework. It reports on the activities up to M12 in WP3 – *Data services & Semantic Enrichment Layer (MATRYCS-GOVERNANCE)*, specifically task 3.6 – *End-to-End Security Framework*.

This deliverable surveys and provides sufficient background of the relevant privacy and security aspects in information systems for application in the MATRYCS project. Moreover, it reports on the design and initial implementation of the MATRYCS End-to-End Security Framework while also presenting and valuating the related composition of envisaged technology solutions to be adopted. Finally, the deliverable outlines the integration aspects of the End-to-End Security Framework with MATRYCS modules and assets, also considering a set of security and privacy guidelines to be adopted throughout the MATRYCS DevSecOps process.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

1 Introduction

The MATRYCS project aims to provide a comprehensive, state-of-the-art AI-powered framework for real-life applications, focusing on building energy services with respect to data processing, analytics, aggregation, and visualization. With big data and information/communication technology at the heart of the project implementation and being a key enabler, sufficient measures must be applied to ensure apt security standards throughout the complete development and operational phases. To this end, the MATRYCS project proposes the End-to-End Security Framework, a holistic vertical security layer encompassing the MATRYCS architecture.

1.1 Purpose of the Document

The purpose of *D3.2: End-to-End Security Framework* is to provide sufficient background as well as guidelines to the MATRYCS consortium – especially to MATRYCS modules and services developers – on security aspects in information systems to be applied in the MATRYCS project and to report on the current implementation of the related End-to-End Security Framework module, developed as a part of the *Data services & Semantic Enrichment Layer (MATRYCS-GOVERNANCE)*. To this end, the deliverable reports on the outcomes and activities of T3.6 *End-to-end security framework* as a part of WP3 *Data services & Semantic Enrichment Layer (MATRYCS-GOVERNANCE)*.

The obtained outcomes and carried out activities heretofore aimed to provide a comprehensive study of the state-of-the-art concepts and approaches in modern information systems, focusing on both cloud as well as the so-called edge environments. In addition, the work has concentrated on designing and valuating a composition of envisaged technology solutions to be adopted as a part of the MATRYCS End-to-End Security Framework, also highlighting the interactions with various MATRYCS modules through a set of security services and guidelines.

1.2 Structure of the Document

The *D3.2: End-to-End Security Framework* is organized as follows:

- In *chapter 1*, the introduction, purpose of the document, and related structure is presented.
- In *chapter 2*, an overview of the security in information systems is given, focusing on the root of trust, end-to-end security, identity and access management, and service-level security concepts to be utilized in the MATYCS project as a part of the project's security vertical.
- In *chapter 3*, the design, technologies, and the initial implementation of the End-to-End Security Framework developed as a part of the MATRYCS-GOVERNANCE layer is presented.
- In *chapter 4*, the envisioned End-to-End Security Framework solution integration in the MATRYCS ecosystem is outlined.
- In *chapter 5*, the security recommendations and guidelines for ensuring compliance with the security standards are provided.
- Finally, *chapter 6* concludes the deliverable and outlines the future steps.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

2 Security in Information Systems

The ISO/IEC 27000:2018 standard [1] defines information security as "preservation of confidentiality, integrity and availability of information," where, additionally, "other properties, such as authenticity, accountability, non-repudiation, and reliability can also be involved." A somewhat related concept to security is privacy; whereas privacy is more concerned with rights, control and usage of information, security on the other hand imposes the safeguarding of information. Information security generally considers a balanced protection on principles also known as a CIA triad [2]:

- **Confidentiality:** Information must be protected and not disclosed to or accessible by any unauthorized entities.
- **Integrity:** Information must maintain its accuracy, consistency, and completeness throughout its whole lifecycle.
- **Availability:** Information must be accessible when required, possible availability disruptions must be prevented.

Ensuring security is typically an iterative process dealing with the identification and mitigation of risks. To prevent confidentiality, integrity, or availability compromise, various security controls on administrative, logical, and physical bases must be in place. The information should be secured both at rest and in transit where, as information normally moves through numerous information systems, security is required at each step of the processing pipeline, thus motivating the use of end-to-end security approaches. The last should hold even more so in modern complex distributed system deployment scenarios, shifting towards a mixture of cloud, fog, and edge paradigms with a diverse set of ecosystems.

Information systems must implement apt security measures and mechanisms adhering to the established cybersecurity standards. The security generally follows a multi-layered approach based on two main principles:

- **Access control:** Access to data or other secured resources is regulated using access control policies with specified roles and responsibilities supplementary to appropriate identification, authentication, and authorization mechanisms. Logging systems in relation to access control are normally employed for security auditing.
- **Cryptography:** All data handling poses a certain risk of data leakage, whether moving data from one location to another (data in transit) or while data is stored (data at rest). Encryption is a process of transforming plaintext data into a ciphertext that is intelligible to an unauthorized third party.

Establishing a secure access-controlled environment fundamentally depends on the capacity to reliably identify and prove the identity of participating entities [3]. In modern systems, entities normally rely on the public key infrastructure (PKI) [4] and identification/authentication using complementary certificates and cryptographic keys. For this, secure cryptographic operations in addition to secure key storage must be implemented also depending on the so-called root of trust, enabling a chain of implicitly trusted functions from hardware as a basis through all the layers up to any applications in the execution environment.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

2.1 Root of Trust

Information systems, inclusive of the related hardware and software, are prone to remote as well as physical attacks. This is especially true with the growing number of devices, also considering the internet of things (IoT), and rising interconnectivity of assets. With some devices being constrained and consequently lacking robust security frameworks, security in IoT has been a surging issue [5] with various efforts being made towards making IoT secure by design. The ubiquity of devices also presents a complex problem not only because of the sheer magnitude of compromise points, but also due to physical accessibility of hardware – particularly on unsecured or public premises. A compromised asset may be exploited for data leakage/alteration, coordinated botnet attacks or as a gateway enabling other attack vectors. Such compromises are largely mitigated by establishing trust; for this, any exchange should exhibit legitimacy and confidentiality, thus creating a chain of trust not only between assets, but also within a single asset – on software and hardware level.

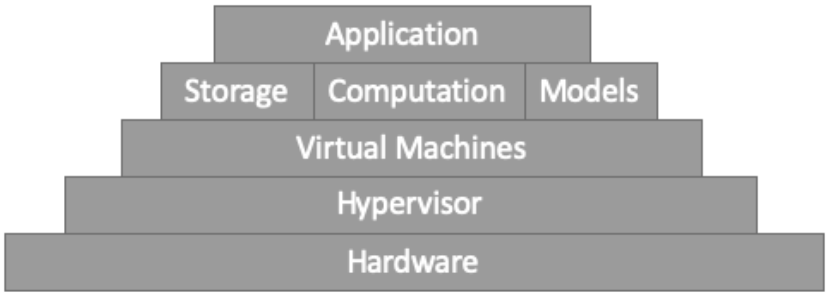

Figure 1: Multi-layered structure of trust.

Cryptography in computer science provides various approaches for identification, authentication, and encryption, enabling secured authenticated communication and trust establishment in addition to providing methods for tampering/repudiation attack prevention. However, modern cryptographic techniques heavily rely on the safety of cryptographic operations execution and the use of secrets, which must be securely stored and protected. This is solved by instituting the root of trust, a foundation for supporting system, software and data verification, integrity, and confidentiality through enabling secure cryptographic storage and operations. Trust exhibits a multi-layered structure, where the groundwork and bottom-most layer (i.e., the root of trust) are normally represented by the hardware and each consecutive layer relies on the previous one with respect to security and trust, as illustrated in *Figure 1* for a generic MATRYCS application execution environment. Hardware trust is generally a prerequisite for enabling a secure tamper-resistant asset's booting procedure using authentic firmware. On top of that, the operating system, hypervisor and associated virtual machines are executed, building upon and extending the established hardware trust. At the very top, applications depending on storage, computation, and ML models are ran, utilizing the trusted secure environment ascertained by lower levels. A compromise at any level of trust vertical implies that all higher levels are or may also be compromised.

In principle, the root of trust may be implemented on the software or hardware level. Whereas the hardware approach is preferred due to being the entry point of execution and its resilience to malware, the software approach does provide additional flexibility. The root of trust is normally implemented and

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

based on:

- **Trusted Platform Module (TPM):** An international standard specifying discrete, integrated, firmware and software variants of a secure cryptographic module.
- **Secure Element (SE):** A tamper-resistant hardware platform enabling secure data storage and limited secure application execution.
- **Trusted Execution Environment (TEE):** A secure isolated environment for executing applications with data and application integrity/confidentiality guarantee.

2.2 End-to-End Security

Due to the increased use of sensors and actuators embedded in everyday objects and increased privacy and security awareness on how and where the data is generated and consumed, security frameworks are inherently evolving. Modern end-to-end security for distributed computing, edge nodes, and edge systems have a physical scope and a digital one as depicted in *Figure 2*. Focusing on physical security, we identify node level and site level security. We refer to a node as an entity that performs computation from constrained devices embedded in objects to mainframe computers in private datacentres while we refer to a site as a more complex environment consisting of one or more interconnected nodes on a well delimited premise such as a factory floor. For the case of cloud computing systems, the taxonomy of physical security mechanisms provided in ref. [6], identifies retina fingerprint, palmprint and face recognition as authentication mechanisms.

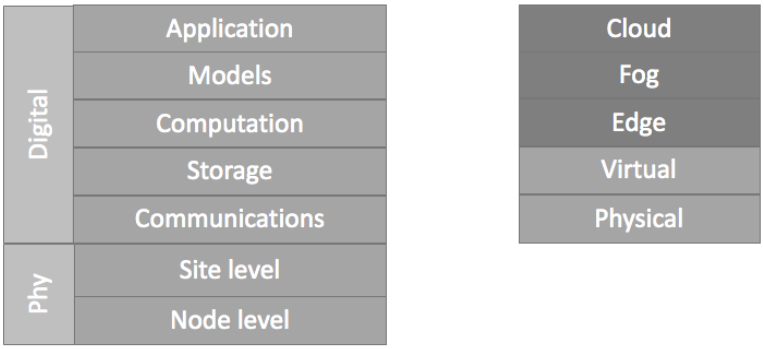

Figure 2: Scope of modern end-to-end security for distributed computing systems.

With respect to digital security, we identify communications, storage, computation, models, and application-level security that is also dependent on the used technology stacks. While in legacy brownfield computing systems, the physical interconnections reflect the logical ones, in modern systems, due to virtualization technologies this may not be the case. Several logical systems may reside on a physical one. Furthermore, the technology stack used to realize distributes computing is different between cloud and edge as also represented on the right-hand side of *Figure 2*. The technology, policies and practices in place must cover the entire scope of the end-to-end security aspects of modern distributed computing systems. For the case of cloud services, the taxonomy in [6] identifies three classes of digital security mechanisms suitable for authentication: credentials (i.e., passwords and SSH keys), multifactor (chip and pin, one time password, captchas, and patterns) and SSO and federation (enterprise

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

SSO, Open ID, SAML and OAuth).

Edge computing is a paradigm in which computation and data are placed closer to the stakeholder. Assume a digitized farm that monitors its livestock and uses this information to care for the animals. The measured data needs to be sent and possibly stored on a computing infrastructure and presented to the decision maker in a useful form. The computing infrastructure can be placed on the premises of the farm where the data from the sensors is transmitted to a gateway that pushes it to a database available on the computing infrastructure set-up on premises. The data processing and visualization services also run on the same infrastructure and the stakeholder (i.e., farm manager) can access information through fixed or portable connected devices such as phone or tabled using the local networks such as WiFi. In this example we are talking about an on-premises edge computing infrastructure. An alternative case would be when the computing infrastructure is placed outside the premises of the stakeholder at the edge of the connectivity service provider (i.e., telco operator or cable operator).

Unlike edge computing, in the case of the cloud, the computing and data storage are taking place in the few locations where the cloud provider deployed their infrastructure. As large cloud providers tend to have server farms in 2-3 locations per continent, the data and services are located far from the stakeholder. While cloud computing services are easy to use, optimized for a wide range of applications, have strict security and performance requirements and are ready to use, they physically and virtually displace the data from the stakeholder and raise risks in case of major events or attacks while also being subject to changing pricing plans.

In its attempt to expose the data at the edge producer level, MATRYCS will rely on an architectural layer referred to as MATRYCS-GOVERNANCE, where data at the building-edge layer will be made available in a privacy-aware manner – considering, among others, data anonymization and cleaning – for the storage and services run in a centralised cloud. This represents a hybrid approach where only well controlled and curated part of the data is stored in the cloud while the rest remains only accessible at the edge. This approach is expected to increase trust in data sharing among stakeholders and subsequently increase models' accuracy by raising the amounts of data available for AI-based learning.

Figure 3: Conventional data governance chain.

Figure 3 depicts the conventional data governance chain in Europe where data generated at the building level can be shared in a raw or aggregated form up the chain to various decision-making stakeholders at district, city, region, nation, or European level for informing behaviour and policies and enabling increased efficiency. The sharing is compliant with data governance rules and laws at the respective

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

levels. A more in-depth overview of the data flow at the different data governance levels is provided in *Figure 4*. The orange circles in the figure represent the flow of data under various levels of governance and the flow of shared raw or aggregated data across the levels. At each of the data governance levels described in *Figure 4*, the data is passed through a complex information and technology stack formed mainly of hardware, virtualization, computation, communication technologies and application/service technologies. A more fine-grained layering is provided by the Cloud Native Computing Foundation in the Cloud Native Landscape Map[1].

Towards the building level, the technology stack may take more the flavour of edge computing while towards the European level it may take a cloud computing flavour. At each of the levels of the technology stack there are possible security threats. To name only a few, at the hardware level there are digital key theft and hardware trojan threats, at the virtualization level there are data breach and resource hijacking threats, at the computation level there are data breach or poisoning as well as model extraction threats, at the communication level there are man-in-the-middle and eavesdropping threats while at the service and application levels there are intrusion and cross-site scripting threats as very briefly also summarized in *Figure 5*. Extensive and in-depth studies of threats and mitigation solutions in cloud stacks can be found in ref. [6]–[8] while for data processing and machine learning pipelines in ref. [9], [10].

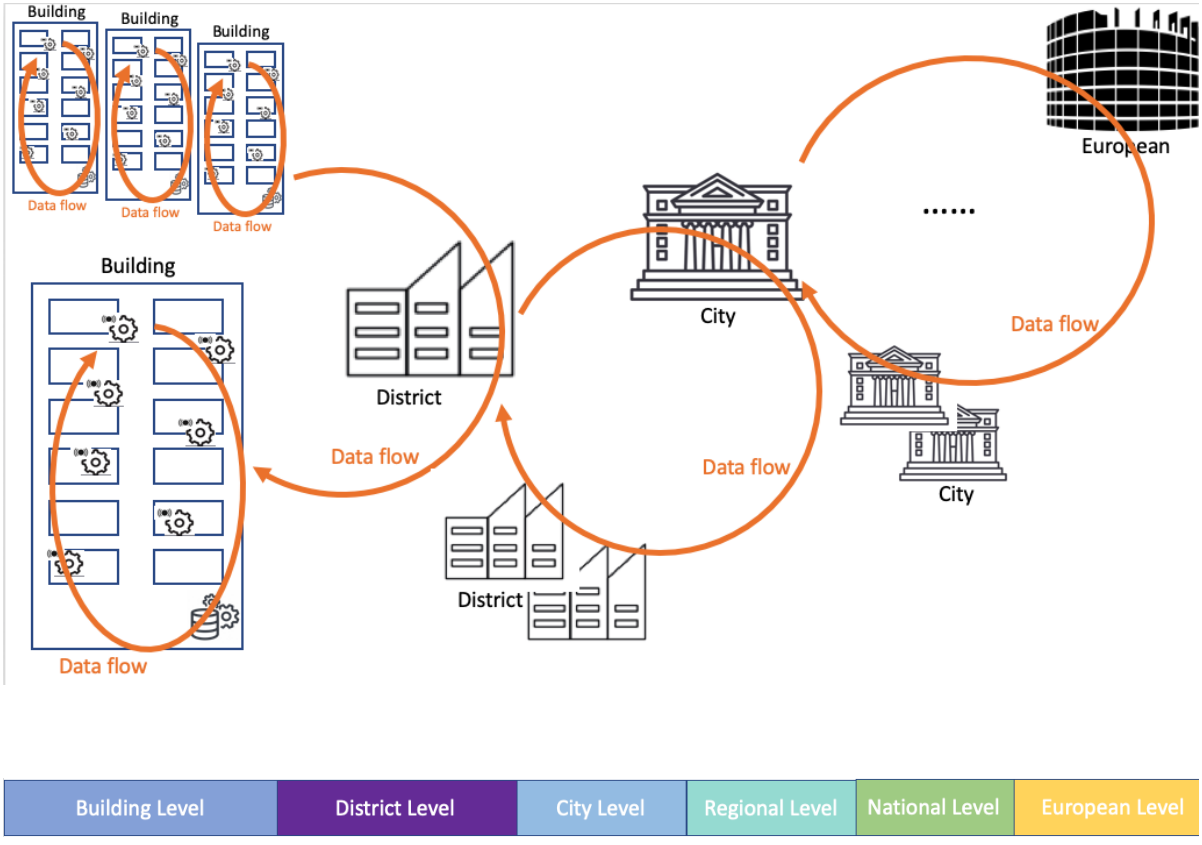

Figure 4: Data flow at different data governance levels.

[1] https://landscape.cncf.io

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

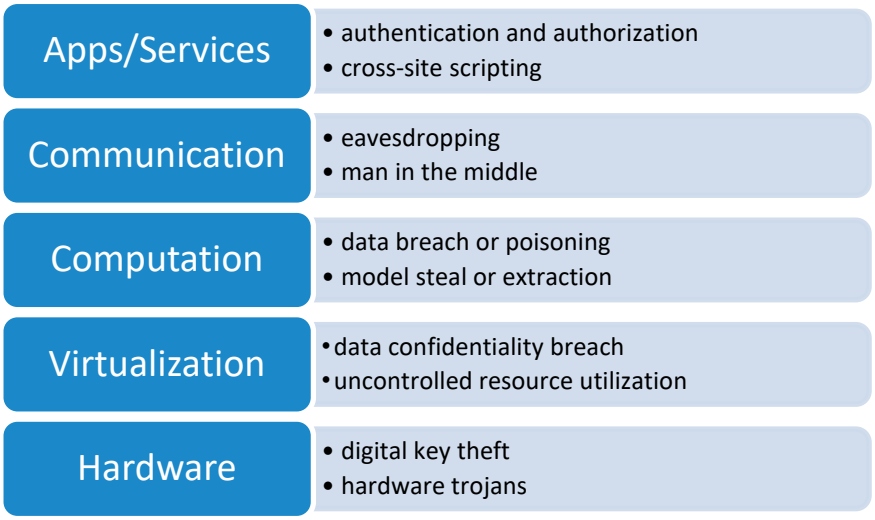

Figure 5: Security threats at different layers of the edge/cloud computing stack.

There is a wide range of tools and components available for realizing edge and cloud stacks for data processing and management at various levels of data governance. In *Table 1* we summarize a few at each level noticing that the ones dedicated for edge are lighter weight and more resource aware.

Table 1: Example technologies for realizing edge and cloud IT stacks.

	Edge	Cloud
App/Service	Flutter	React, Angular, Grafana, NodeJS, Django
Communication	LoraWAN, WiFi, BT, SigFox, NB-IoT	ETH, HTTP, MQTT, Kafka
Computation	OpenEMS, TinyML, Tensorflow Lite	Kafka, Elasticsearch, Hadoop, Spark, Hive, Tensorflow, Keras
Virtualization	KVM, Trango, Xen, K3S, CodeZero, Docker	VMware, Kubernetes, OpenStack, Ceph, libvirt, Eucalyptus
Hardware	RPi, BeagleBone, Intel Nuc, NVIDIA Jetson	Intel Xeon, Kunpeng, Ascend, NVIDIA

Focusing on the software frameworks of edge and cloud computing systems, we further discuss security aspects identified by the community and review the security features of existing open-source systems.

2.2.1 Edge Environments

In ref. [11], the authors identify four deployment models of an edge computing environments: cloudlet,

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

fog, multi-access edge and IoT. The Industrial Internet of Things Consortium[2] introduced a framework that covers the following deployment models:

- **Simple edge deployment:** "An edge node is an assembly of hardware and software components that implement edge functions. Such a standalone edge computing node can be installed anywhere in the edge system to provide computing, networking, and storage services close to data producers or consumers"
- **More complex deployment:** "In a slightly more complex model of edge deployment, multiple logical edge nodes may be instantiated in a single physical edge node. They share the same hardware platform but are fully isolated from each other. This deployment model is modular, scalable, and efficient. It is the primary support mechanism for multi-tenancy."
- **Extremely complex model:** "A logical edge computing node is assembled from the one or more physical or logical edge nodes. One version of this merges the capabilities of multiple physical edge computing nodes, which may be peers on the same or adjacent layer(s), to handle a computation, networking, or storage load heavier than a single physical edge node can manage. In a variation, multiple physical edge nodes are grouped into a fault-tolerant cluster, so that a failure in one of the edge nodes will be mitigated by its peers."

The same consortium also introduced a security framework[3] focusing on assurance, security, safety, reliability, resilience, privacy, and trust. The corresponding definitions of the key characteristics of the framework as presented in the document are:

- "**Assurance** requires the collection and analysis of evidence that supports the design, construction, deployment and test of the system, and its activities in operation."
- "**Security** is the condition of the system being protected from unintended or unauthorized access, change or destruction."
- "**Safety** is the condition of the system operating without causing unacceptable risk of physical injury or damage to the health of people, either directly or indirectly, as a result of damage to property or to the environment."
- "**Reliability** is the ability of a system or component to perform its required functions under stated conditions for a specified period of time."
- "**Resilience** is the property of a system that behaves in a manner to avoid, absorb and manage dynamic adversarial conditions while completing the assigned missions, and reconstitute the operational capabilities after causalities."
- "**Privacy** is the right of an individual or group to control or influence what information related to them may be collected, processed, and stored and by whom, and to whom that information may be disclosed."

The document provides further guidelines on how to develop such a security framework for all the layers of the edge stack represented in *Figure 2* and *Figure 5*. In *Table 2*, we overview selected existing open-

[2] https://www.iiconsortium.org

[3] https://www.iiconsortium.org/pdf/IIC_PUB_G4_V1.00_PB-3.pdf

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

source edge frameworks with respect to their current security capabilities. As with the case of cloud, also the numbers of available frameworks[4] and tools is relatively large, therefore our selection is based on their maturity and completeness.

Table 2: Security aspects analysis of existing edge frameworks.

Framework	Description	Supported security mechanisms
LF EDGE[5]	Unifies open-source edge frameworks across IoT, telco, cloud, and enterprise edge markers at the infrastructure and at the application level.	❍ HTTPS Encryption ❍ Authentication ❍ Password rotation ❍ Authorization ❍ Certificates
EdgeX[6]	Flexible and scalable open-source framework under the LF Edge umbrella that sends and receives data from enterprise, cloud and on-premises applications and enables AI at the edge.	❍ Secret creation, store, and retrieve (password, cert, access key etc.) ❍ User account creation with optional either OAuth2 or JWT authentication ❍ User account with arbitrary Access Control List groups (ACL) ❍ Data encryption ❍ Secure Secret Storage, Dynamic Secrets
StarlingX[7]	StarlingX provides an OpenStack base layer with compute, storage, and networking capabilities, along with configuration and other management functions.	❍ TLS support on all external interfaces ❍ Kubernetes service accounts and RBAC policies for authentication and authorization of Kubernetes API / CLI / GUI ❍ Encryption of Kubernetes Secret Data at Rest ❍ Keystone authentication and authorization of StarlingX API / CLI / GUI

[4] https://awesomeopensource.com/projects/edge-computing

[5] https://www.lfedge.org/projects/fledge/

[6] https://www.edgexfoundry.org

[7] https://www.starlingx.io

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

		❍ Barbican is used to securely store secrets such as BMC user passwords ❍ Networking policies / Firewalls on external APIs ❍ UEFI secureboot ❍ Signed software updates
yocto[8]	The Yocto project can be used to create tailored Linux images for embedded and IoT devices, or anywhere a customized Linux OS is needed.	❍ Configurable ❍ Linux based

2.2.2 Cloud Environments

Cloud environments are formed by a complex amalgamation of technologies from the available ones presented in the Cloud Native Landscape Map. There are three common services models for cloud environments as also illustrated in *Figure 6* and discussed in [12]:

- **Software as a Service (SaaS)** that provide applications (usually a Web application) to users. The implementation details of the application or hosting infrastructure is transparent to the users. Modern SaaS environment are developed so-called service meshes. A service mesh can be described as an infrastructure layer that is responsible for the communication between services [13]. The major providers of service mesh for SaaS are Istio, Linkerd and Consul as identified in ref. [14]. All three frameworks can be seen as security enhancements to Kubernetes that is otherwise a workload orchestration tool and does not support most of the security measures for communication between microservices that are working inside of Kubernetes environment. However, they support also other orchestrators beyond Kubernetes.
- **Platform as a Service (PaaS)** provides development and deployment platforms, a set of APIs, libraries, programming languages and associated tools used for application creation. The users in this case tend to be application developers to whom details about the hosting infrastructure are abstracted.

[8] https://www.yoctoproject.org

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

〉 **Infrastructure as a Service (IaaS)** provides the low-level infrastructure to create customized application environments or even higher-level products (which might be PaaS or SaaS). Here the users have control over the infrastructure and are able to develop their own platform and applications on top of it. The most prominent open-source enablers of this model are Nimbus, Eucalyptus, OpenStack, CloudStack with OpenStack being probably the most popular choice. Detailed feature based comparisons of these frameworks are available in ref. [15]–[17]. According to ref. [17], at the time of writing, Eucalyptus and OpenStack offered virtual machine isolation, user security features and system security.

With respect to the deployment models for these environments, there are three main options, i.e., private, public and community while also combinations of these are possible, also referred to as hybrid. The Cloud Security Alliance[9] is an organization that provides periodic security guidelines related to securing various aspects of the cloud, i.e., microservices [18], and implementing security policies such as Zero-trust, a security model that treats all network traffic as hostile, even if it is inside the perimeter. The recommendations of the Cloud Security Alliance are generic and technology/provider agnostic, therefore there are no specific technology recommendations.

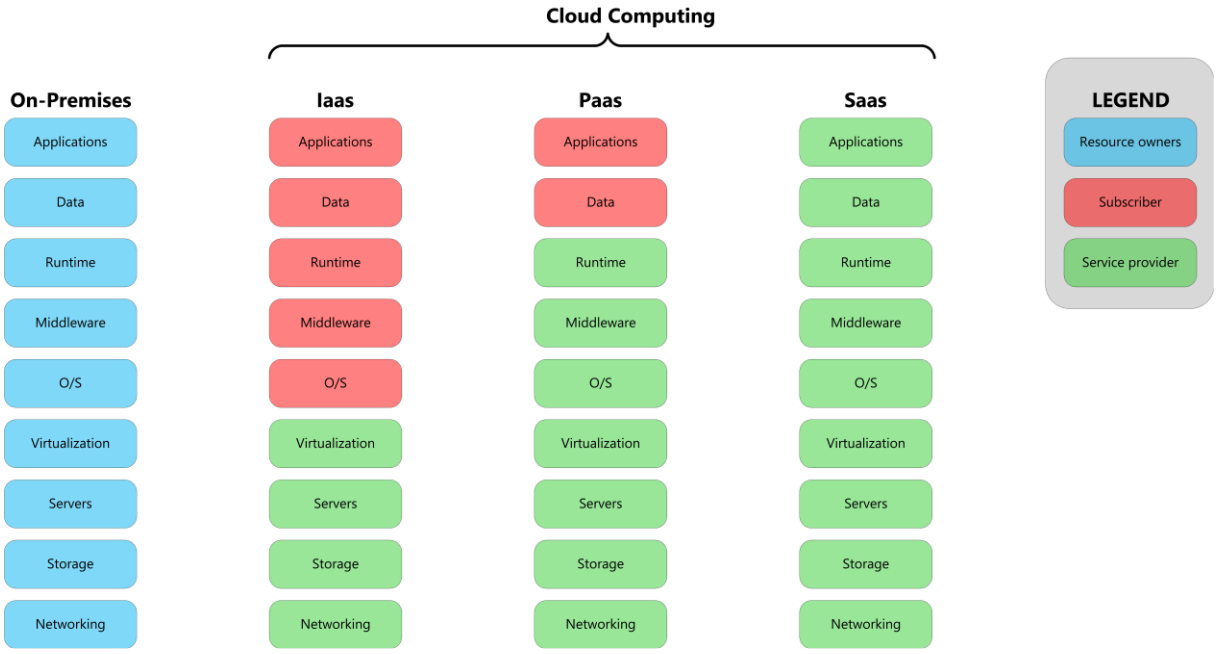

Figure 6: Service delivery models as shown in [19].

The MATRYCS project aims to deliver a SaaS cloud therefore for which the authors of [20] identified the following challenges:

i. **Data security** that they further break down to data storage security, data access control, data backup and recovery, data integrity and data transfer security.

[9] https://cloudsecurityalliance.org/research/working-groups/containerization/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

ii. **Application security** that is further being broken down to software design flaws, user interface and web technologies, web services and malware.

iii. **Software-as-a-Service Security** that is broken down in vulnerabilities of virtual machine, vulnerabilities in virtual networks, service communication vulnerabilities.

At the time of writing of [20], artificial intelligence and machine learning services were not yet such an integral part of SaaS platforms as they are now. Training and delivering AI models extends the Data security challenge identified by [20] with corrupting model training or operational data, the Application Security challenge with stealing the outputs of the models and the Software-as-a-Service Deployment Security challenge with model theft and swapping as discussed in more detail in [9].

2.3 Identity and Access Management

Identity and access management (IAM) in cloud environment is a crucial concern for the acceptance of cloud-based services. IAM systems perform different operations for providing security in the cloud environment that include authentication, authorization, and provisioning of storage and verification. IAM system guarantees security of identities and attributes of cloud users by ensuring that the right persons are allowed in the cloud systems [6]. They are capable of performing functions like, administration, discovery, maintenance, policy enforcement, management, information exchange and authentication and they are used to authenticate users, devices or services and to grant or deny rights to access data and other system resources.

As the WWW triggered a revolution in software development where, and as noticed in ref. [21] "The emergence of the software-as-a-service model, Internet-based developer forums (e.g., Stack Overflow, https://stackoverflow.com), and open source software repositories (e.g., GitHub, https://github.com) have enabled an approach in which people routinely trawl online for ready-made solutions for all kinds of problems; the discovered libraries and code snippets are included in applications with little consideration or knowledge about their technical quality or details.". According to the authors, such development is referred to as opportunistic design, opportunistic reuse, ad hoc reuse, scavenging, software mashups, mashware, or sometimes even frankensteining. The overwhelming choice of tools and libraries also affect the security of the systems they are integrated in. The authors of [22] noticed that "the variety of approaches to solve IAM makes it hard to compare or even combine them" thus "it is increasingly difficult to provide secure implementations and configurations".

Some of the available tools for realizing IAM for SaaS are Keycloak, Auth0, Okta, FreeIPA, Dex, and Vault.

2.3.1 OAuth2 Protocol

OAuth2[10] is an authorization framework that enables applications – such as Facebook, GitHub and DigitalOcean – to obtain limited access to user accounts on an HTTP service. It works by delegating user authentication to the service that hosts and authorizing third party applications to access that user account. OAuth2 provides authorization flows for web and desktop applications as well as mobile

[10] https://oauth.net/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

devices. OAuth2 protocol defines 4 roles:

- **Resource Owner:** The Resource owner is the user who authorizes an application to access their account. The application's access to the user's account is limited to the scope of the authorization granted (e.g., read or write access)
- **Client:** The client is the application that wants to access the user's account. Before it may do so, it must be authorized by the user, and the authorization must be validated by the API.
- **Resource Server:** The resource server hosts the protected user accounts.
- **Authorization Server:** The Authorization Server verifies the identity of the user then issues access tokens to the application.

The *Figure 7* depicts what OAuth2 roles are and how they interact with each other, as described in the following:

1. The application requests authorization to access service resources from the user.
2. If the user authorized the request, the application receives an authorization grant.
3. The application requests an access token from the authorization server (API) by presenting authentication of its own identity, and the authorization grant
4. If the application identity is authenticated and the authorization grant is valid, the authorization server (API) issues an access token to the application. Authorization is complete.
5. The application requests the resource from the resource server and present the access token for the authentication.
6. If the access token is valid, the resource server (API) serves the resource to the application.

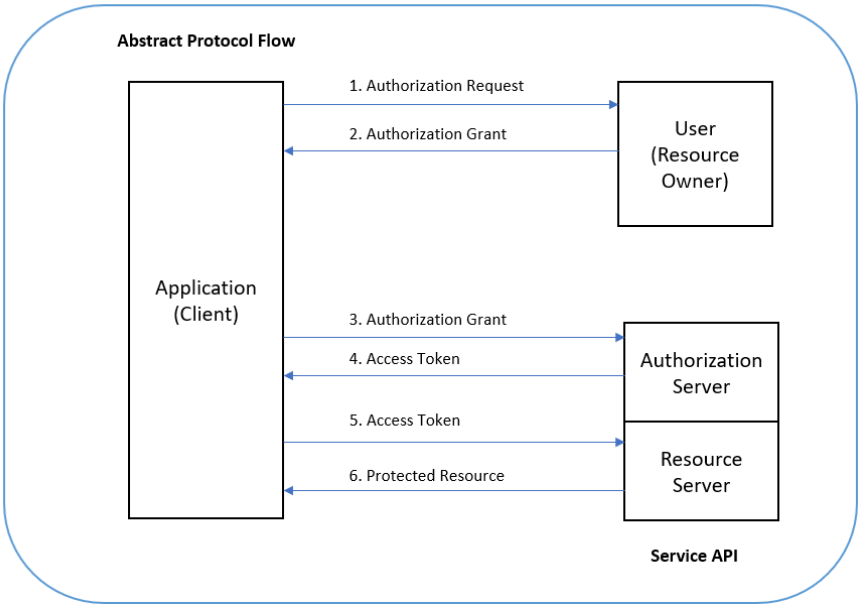

Figure 7: OAuth2 protocol flow.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Before using OAuth2 with an application, the application should be registered with the service. This is done through a registration form in the developer of API portion of the service's website, where Application Name, Application Website and the Redirect UI should be provided. The Redirect URI is where the service will redirect the user after authorization of the application and therefore the part of the application will handle authorization codes or access tokens.

Once the application is registered, the service will issue client credentials in the form of a client identifier and a client secret. The client ID is a publicly exposed string that is used by the service API to identify the application and is also used to build authorization URLs that are presented to users. The Client Secret is used to authenticate the identity of the application to the service API when the application requests to access a user's account and must be kept private between the application and the API.

In the OAuth2 Protocol Flow outlined previously, the first four steps cover obtaining an authorization grant and access token. The authorization grant type depends on the method used by the application to request authorization, and the grant types supported by the API. OAuth2 defines three primary types, each of which is useful in different cases:

- **Authorization Code:** It is the most commonly used grant type, and it is optimized for server-side Applications, where source code is not publicly exposed, and Client Secret confidentiality can be maintained.
- **Client Credentials:** Used with Applications that have API access. This Grant type provides an application a way to access its own service account. Examples of when this might be useful include if an application wants to update its registered description or redirect URI, or access other data stored in its service account via the API.
- **Device Code:** Used for devices that lack browsers or have input limitations. The purpose of this grant type is to make easier for users to authorize applications more easily on such devices to access their accounts. Examples of when this might be useful include if a user want to sign in into a video streaming application on a device that doesn't have a typical keyboard input.

2.3.2 UMA 2 Protocol

In the context of software systems, User-Managed Access (UMA 2 OR UMA 2.0) is a standard [23] that aims to strengthen data privacy based on the well-known privacy by design principles. In technical terms, UMA 2 is a party-to-party authorization protocol based on the OAuth2 authorization framework. In order to understand better how UMA 2 works, a common privacy dilemma will be used to explore this topic in detail (see *Figure 8*). Alice maintains an online bank account at Capital Bank, and she has granted the following parties to access her online bank account:

- Bob – Alice's spouse,
- Carol – Alice's account,
- NFC based mobile payment application.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Each party above accesses Alice's online account for different purposes. For example, Carol only requires read-only access to account data while the mobile application should have the privilege to perform payment transactions. Capital Bank has facilitated Alice to grant the above privileges to each party via an online banking application. This granting process is technically known as party-to-party authorization because Alice grants certain access to another specific party to access her bank account.

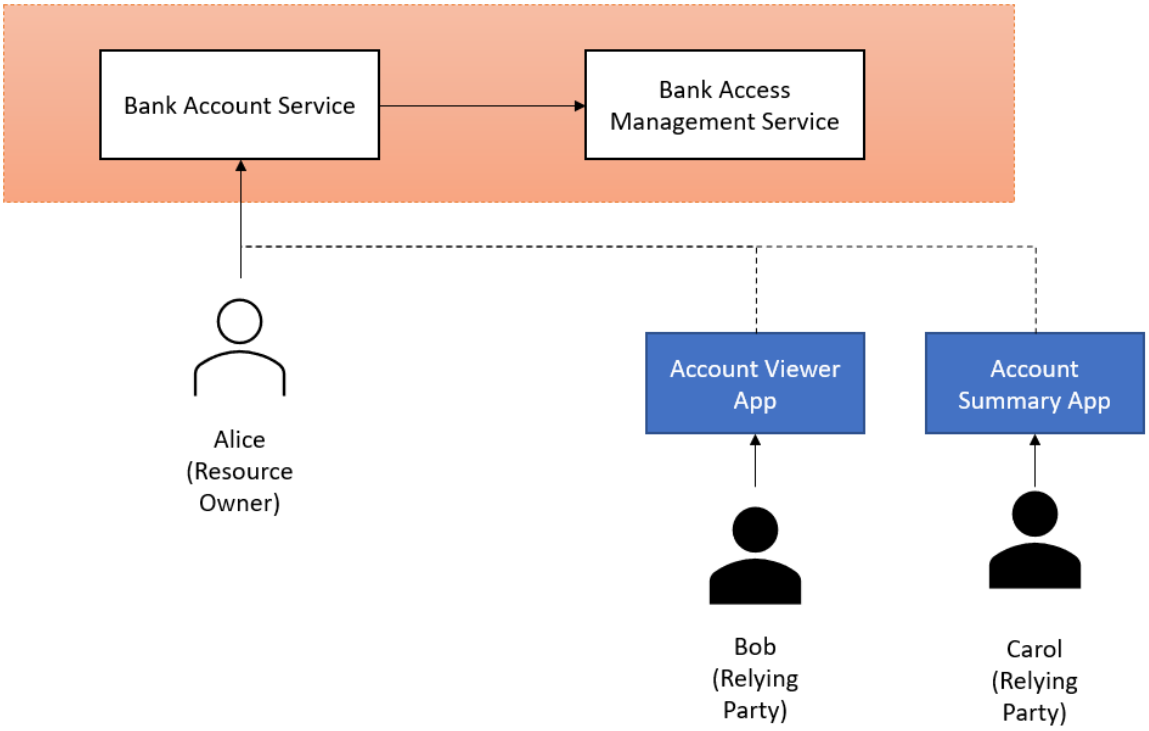

Figure 8: UMA 2 example.

In the context of UMA 2, Alice (who is the account owner) is known as Resource Owner (RO). The third parties that have access to Alice's bank account are known as Relying Parties (RP). Applications used from Relying Parties to access Alice's account are known as Clients. The entity responsible for the generation of various tokens and tickets, the RO and third-party authentication and access policy evaluation is known as the Authorization Server (AS). Generally, both the AS and RS belong to the same organization or there should be a strong trust relationship between the organization that own the AS and RS functions. Furthermore, Alice can define authorization policies. For example, Carol (the accountant) can only view account data but cannot perform any transactions, while Bib can perform transactions up to 1000 Euros. UMA 2 does not define the right approach for authorization policies.

2.3.2.1 UMA 2 Flow

Step 1 – Client Attempts to Access a Protected Resource

Let's assume that Bob is trying to check the balance of Alice's account on Capital Bank using an

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

application installed on his mobile phone. As depicted in *Figure 9*, the application sends an HTTP request to the Resource Server during this initial step, requesting the account balance without any security tokens.

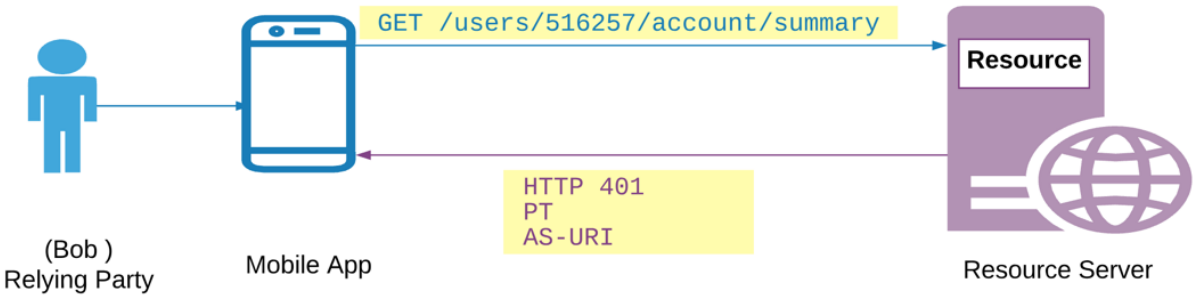

Figure 9: Client attempts to access a protected resource.

In response the Resource server sends the client HTTP 401 (Unauthorized) status code along with the address URL of the Authorization Server and a special token that is known as the permission ticket (PT) – *Table 3*.

Table 3: Response received from Resource Server – 401.

HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="example",
as_uri="https"//as.example.com"
ticket="016f84e8-f9b9-11e0-bd6f-0021cc6004de"

The value WWW-Authenticate header is set to value "UMA", which indicates that the particular resource is protected using UMA 2. The value of attribute *as_uri* indicates the URL of the AS where the client should reach for further interactions to get an access token. The value of "ticket" attribute contains the permission ticket for this particular resource access interaction by this particular client on behalf of a specific relying party.

Step 2 – Client Interacts with Authorization Server (AS) to get an Access Token

After processing the response message, the client could realize that it has to possess an access token in order to access the above resource and that it has to use UMA 2 protocol to interact with the AS.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

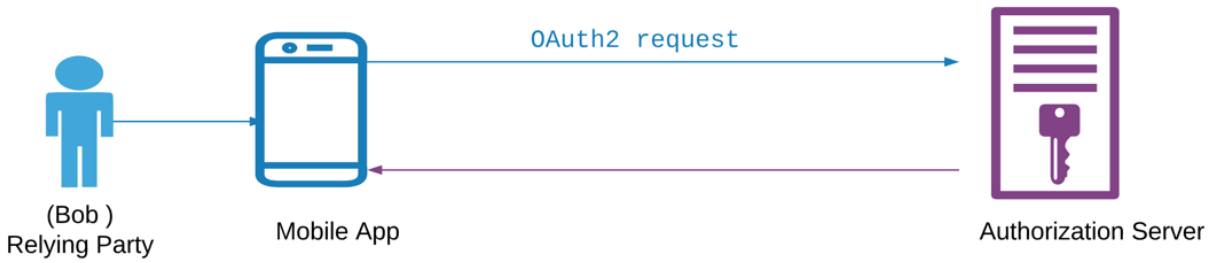

Figure 10: Client Interacts with AS to get an Access Token.

In this step, *Figure 10*, the Client makes an OAuth2 token to request to the token endpoint of the AS along with the PT – *Table 4*. In this OAuth2 request the client should use the value "urn:ietf:params:oauth:grant-type:uma-icket" as the value of grant type parameter. Other than the UMA 2 grant type value and UMA 2 defined ticket parameter, all other parameters are identical to the standard OAuth2 parameters.

Table 4: Request for Protection Token.

POST /token HTTP/1.1
Host: as.example.com
Authorizatiokn: Basic jwfLG53^sad$#f
....
grant_type= urn:ietf:params:oauth:grant-type:uma-icket&ticket=
016f84e8-f9b9-11e0-bd6f-0021cc6004de"

Additionally, we have to make the following assumptions about the client:

- The client has already registered with the AS and obtained the OAuth2 client credentials (client_id and client_secret) from the AS. To achieve this task, it is possible to use another OAuth2-related specification called OAuth2 Dynamic Client Registration Protocol, which defines a RESTfull API on the AS to facilitate client registration or any other approach supported by the particular authorization server.
- The client processes knowledge about the configuration of AS such as supported authentication mechanisms, supported grant types and endpoint URLs.

Upon receiving the OAuth2 request the AS carries out the following procedure before relying to the client:

- Authenticate the client and validate the request message according to both OAuth2 and UMA 2 specifications.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- Authenticate the Relying Party by redirecting the RP to the authentication endpoint of AS. However, the underlying authentication mechanism is out of the scope of the UMA 2 standard.
- Upon successful authentication, the AS may try to gather claims from the RP interactively.
- Evaluate authorization policies in the context of authenticated RP, requested resources, scopes and some other criteria. Once all of the above checks are completed AS returns an access token (*Table 5*). According to UMA 2 protocol, this access token is known as the Relying Party Token (RPT).

Table 5: Response from Authorization Server – Received PRT.

```
HTTP/1.1 200 OK
Content-Type: application/json
....
{
   "access_token": "ZWRjb25fhfc2F2ala5Zzwt56fd"
   "token_type": "Bearer",
   "pct": "c2F2ZWRjb25Zzw50"
}
```

In addition to the RPT, the Authorization Server may return the following tokens as well:

- **Refresh token:** Same as the refresh token used in OAuth2, the purpose is to generate an active RPT by only providing this refresh token without repeating the whole token generation process.
- **Persisted Claims Token (PCT):** An optional reference handle that represents the claims gathered during the above-mentioned claim gathering process. The client can send the PCT token when it sends the UMA 2 token request next time to skip claim gathering steps.

Step 3 – Client Attempts to Access a Protected Resource by Sending RPT to Resource Server

This step is similar to step 1. The only addition here is that the client sends the RPT value along with the resource access request through the Authorization HTTP header, *Figure 11*.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

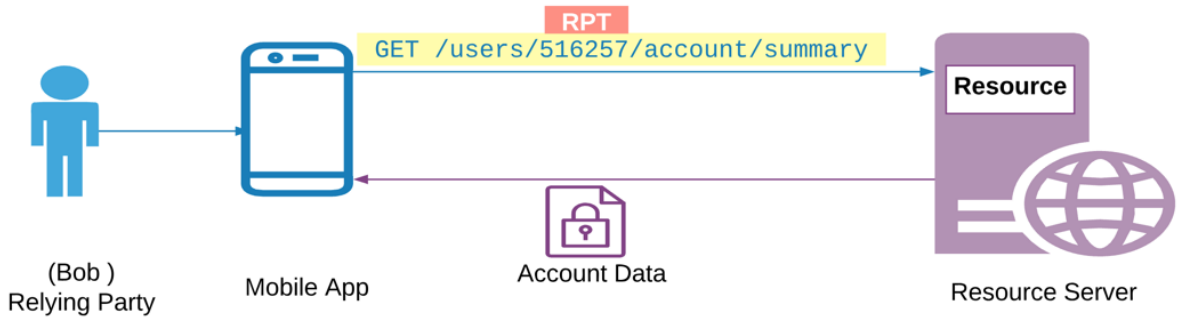

Figure 11: Client attempts to access a Protected Resource by sending the RPT to AS.

Once the Resource Server receives a request similar to the above one, it extracts the access token (RPT) and validates it. In the case of token validation, the RS can interact with the Authorization Server via the OAuth2 introspection endpoint. After this token gets validated, the AS returns the requested resource to the client. In most practical cases, it's recommended to cache the result of the token Introspection call to improve the performance of the business activity.

2.3.3 UMA 2 and OAuth2: The Difference

OAuth2 is an access delegation protocol that facilitates third-party applications to access protected resources on behalf of the resource owner (RO) using a temporary access token issued by an AS with the resource owner's approval. Generally, OAuth2 clients access protected resources on behalf of the resource owner (RO), not on their own or representing another party, as shown in *Figure 12*. This is why OAuth2 is usually known as an access delegation protocol. In contrast to OAuth2, UMA 2 allows resource owners to delegate access to third parties based on well-defined authorization policies maintained in the authorization server (AS). This is the fundamental difference between OAuth2 and UMA 2. However, in both of the cases, the client receives an access token to be used when accessing a protected resource.

OAuth2 has a concept called Scope, which can be used to denote various permissions associated with a resource. Yet, there is no defined semantic to represent the resource in OAuth2.

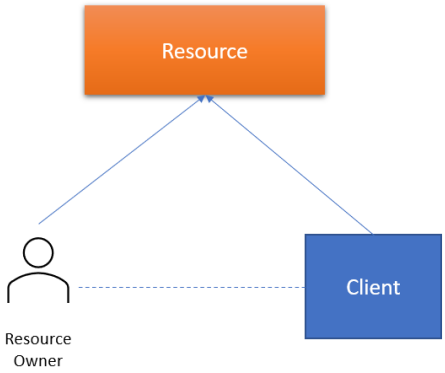

Figure 12: OAuth2 authorization flow.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

In contrast to OAuth2, UMA 2 has introduced a notation to represent resources and an API to manage them. UMA 2 is extended from OAuth2 by introducing a new grant type called "UMA 2.0 Grant for OAuth 2.0". This is similar to the way that OpenId Connect protocol crafted an authentication protocol by extending OAuth2. UMA 2 authorization flow is shown in *Figure 13*.

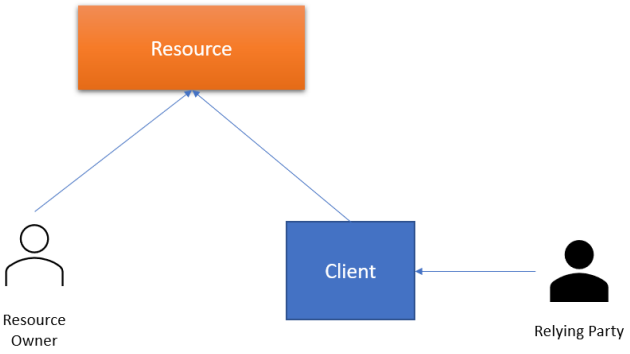

Figure 13: UMA 2 authorization flow.

2.4 Service-Level Security

Services created by different stakeholders usually need to implement more advanced security measures by themselves. This could mean the developers should configure the whole system and sometimes even modify the services they have already created or have to create clear guidelines for other stakeholders to follow to reach a sufficient level of security. In this case, scalability of such a solution is under question.

To address the security of service, communication mesh can be used [24]. The mesh optimizes the routing from one service to another, optimizing all moving parts and taking care the overloading does not happen. When a new instance of service is introduced, the communication complexity increases and with it also the possible new point of failure is introduced. Meshing monitors performance metrics between service-to-service communication, enabling creating independent rules for interservice communication to create a more efficient and reliable system. In principle, a service mesh is an infrastructure layer that:

- Manages communication between services.
- Performs load management and monitoring.
- Applies complex routing for higher scalability.
- Provides encryption and authentication.

Developer of the services can take advantage of mesh tools to:

- configure network behaviour,
- manage traffic flow,
- configure identities through policy enforcement.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

A mesh is logically split into two modules: data plane and control plane [25] as shown in *Figure 14*. Sidecar proxies handle traffic and apply actions on individual services. Service mesh data plane is responsible for:

- **Discovery** and **health monitoring** of the services.
- **Routing** and **load balancing** by setting timeouts, circuit breaking settings, fails decisions, request location configuration.
- **Authentication** and **authorization** of incoming requests, cryptographic proof of the peers, access control policies, certified refresh intervals, white and blacklists for approving connection and creating granular control factors like time of day.
- **Encryption** of requests and responses from each service.
- **Circuit breaker pattern** in which the service mesh can isolate unhealthy instances and bring them back once it is warranted.

The control plane configures the data plane, and it turns all the data planes into a distributed system. This provides flexibility to change the policies without modifying microservice code [26]. Some examples of service mesh landscape, with main complete service mesh solutions compared in *Table 6*, are:

- **Data planes:** Linkerd, NGINX, HAProxy, Envoy, Traefik
- **Control planes:** Istio, Nelson, SmartStack

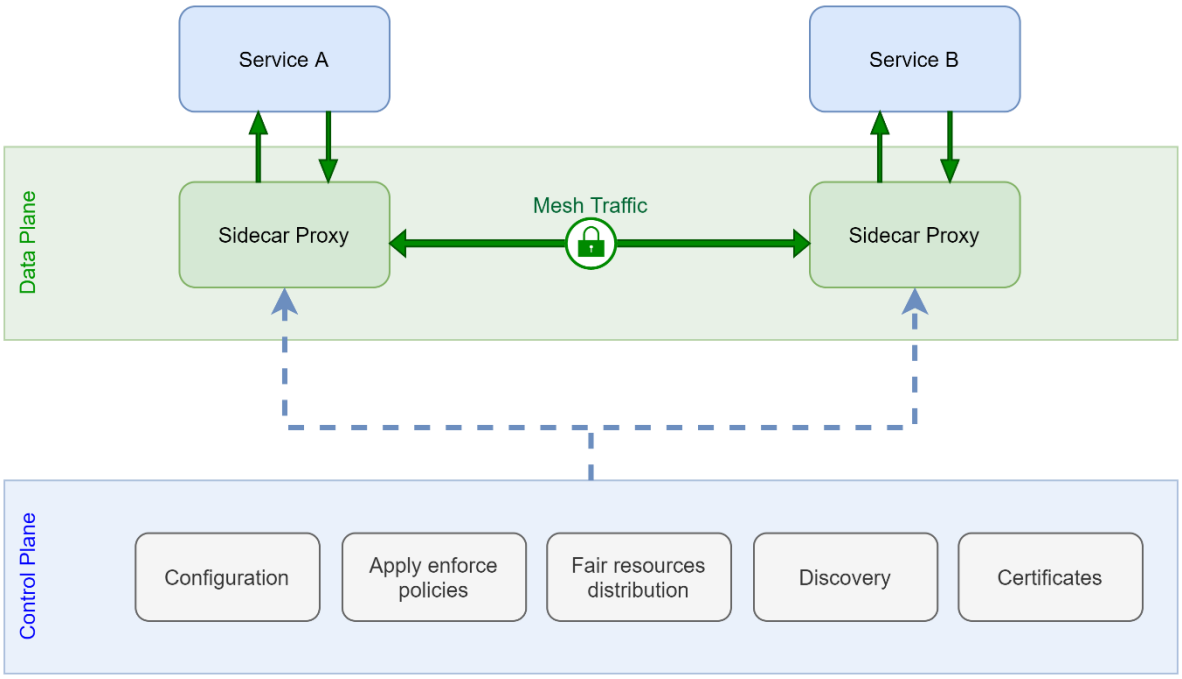

Figure 14: Service mesh composition.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Table 6: Comparison of service mesh solutions as shown in [13].

Properties	Istio	Linkerd	Consul
mTLS	Yes	Yes	Yes
Certificate Management	Yes	Yes	Yes
Authentication and Authorization	Yes	Yes	Yes
TCP	Yes	Yes	Yes
Traffic Rate Limiting	Yes	No	Yes
Traffic testing	Yes	Limited	No
Monitoring	Yes, with Prometheus	Yes, with Prometheus	Yes, with Prometheus
Distributed Tracing	Yes	Some	Yes
Multicluster support	Yes	No	Yes
Installation	Helm and Operator	Helm	Helm

2.4.1 Software Vulnerabilities and Threats

Software vulnerabilities and threats play one of the most important security issues and are mostly related to the software developer and not that much to system administrator configuration issues [27]. They usually are introduced unintentionally even though the developer of software used all precautions during design of the service such as:

- Buffer overflow risks (especially important for low-level languages such as C).
- Format string problems (e.g., when the input string is evaluated as a command by application, creating intentional misbehaviour of the system).
- Race conditions on shared data.
- Shell metacharacter.
- Poor random number generation.
- Numeric errors.
- Out of bound read.
- Use after free.
- Insufficient input validation.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- Code quality.

Other common weaknesses can be found in the Common Vulnerabilities and Exposures (CVE) database [28] maintained by non-profit organization MITRE[11]. Therefore, each MATRYCS module or service developer should be aware of them when developing the solutions. Going from the creation of the service to the deployment of the service to the specific system, another set of software vulnerabilities needs to be addressed and precautions must be taken to eliminate them or at least minimize them as much as possible. The most common system-related software security issues are:

- Unauthorized access to the service code and code injections, especially in the case of script programming languages (e.g., need to compile code, code obfuscation).
- User permission, privileges, and access control (e.g., configuration oversight in the system granting elevated access to resources that are usually protected from an application).
- Using outdated libraries and frameworks with known vulnerabilities.
- Improperly configured and badly coded APIs could lead to data leaks and exposure.
- Cross-site scripting (XSS) (e.g., to execute scripts and hijacks user session).
- Broken authentication and authorization (e.g., authentication and session management are implemented incorrectly).
- Insecure deserialization.
- Insufficient logging and monitoring.

In information systems, an important set of issues arise from the so-called continuous obsolescence phenomenon, which forces the developers and system administrators to constantly update or even reinstall system parts, software, or software libraries. This happens due to the rapidly changing technology which renders the swift updates and inter-software integration difficult. In general, such updates are necessary for the software to continue its function, but normally do not increase the utility of the software. To mitigate the related security problems, this must be address throughout the complete lifecycle of software. Consequently, in practice, it is important to adhere to open/joint standards and achieve appropriate balance between stable and new technologies, also with respect to interface control.

Many tools exist that try to identify security issues and discover possible flaws before they might be exploited. Such tools are generally necessary for employing end-to-end secure systems/software and are thus proposed for adoption in the MATRYCS project as a part of its security layer. The main idea of the tools is to help developers with limited domain knowledge to perform comprehensive tests of the system ranging from potentially dangerous files and programs, identifying version specific problems, the configuration of the system and creating reports to the user so they can start resolving the issues. The services with their associated attacks can be grouped into four major groups [6]:

i. **Protocols and standards:** The main issues are hijacking of the session, network-based attacks, cookies issues, and transport layer security attacks (TLS). Mitigation:
    a. *Using secure session:* with SSL authentication when performing the sensitive operation, time out, …
    b. *Anti-virus:* use and update regularly anti-virus software.

[11] https://www.mitre.org/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

c. *Anti-malware:* use and update regularly anti-malware software.

ii. **Web-Services:** The main issues are spoofing and wrapping attacks. Mitigation:

a. *Implementation of strict security policies* at both sides, including web service access control mechanisms.

iii. **Web-technologies:** The main issues are web sites growth infection, session attacks, download of malware and browser vulnerabilities. Mitigation:

a. *Vulnerability analysis* by a skilled and well-trained person to track and resolve network problems.

b. *Preventive actions*, e.g., use of the software tools that automatically analyse the service and system before it is exposed in the real environment.

iv. **Availability of Service:** main issues are flooding, DoS and DDoS attacks, DNS reflection and amplification attacks. Mitigation:

a. *Regulating:* regulation of request.

b. *The fleet of servers:* setting of high availability environment (HA) that spreads across multiple data centres in implements disaster recovery (DR) plan.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

3 End-to-End Security Framework

The MATRYCS project identified a need for a vertical security layer spanning and interacting with several building blocks of the MATRYCS architecture for enabling authentication, authorization, logging of various events in the system and enforcement of security as well as privacy aspects. The MATRYCS End-to-End Security Framework is thus considered a generic MATRYCS security layer, covering *MATRYCS-GOVERNANCE*, *MATRYCS-PROCESSING* and *MATRYCS-ANALYTICS* (see *Figure 15*). Specifically, the framework encompasses and relates to several entities in the MATRYCS architecture and deployment scenarios: infrastructure/assets, AI/ML services with a focus on big data, MATRYCS end-users, and data.

The End-to-End Security Framework aims to secure the MATRYCS platform and its constituent information – thus enhancing the trustfulness of the system – by applying high-level security and fine-grained access control as well as appropriate mechanisms for maintaining and reinforcing legal and security policies over MATRYCS resources. In addition to providing the means of encrypted inter-service communication, the framework considers data processing/security constraints in relation to data encryption standards and anonymization, especially in conjunction with Data Storage, Distributed Query Engine and Data Semantic Enrichment components.

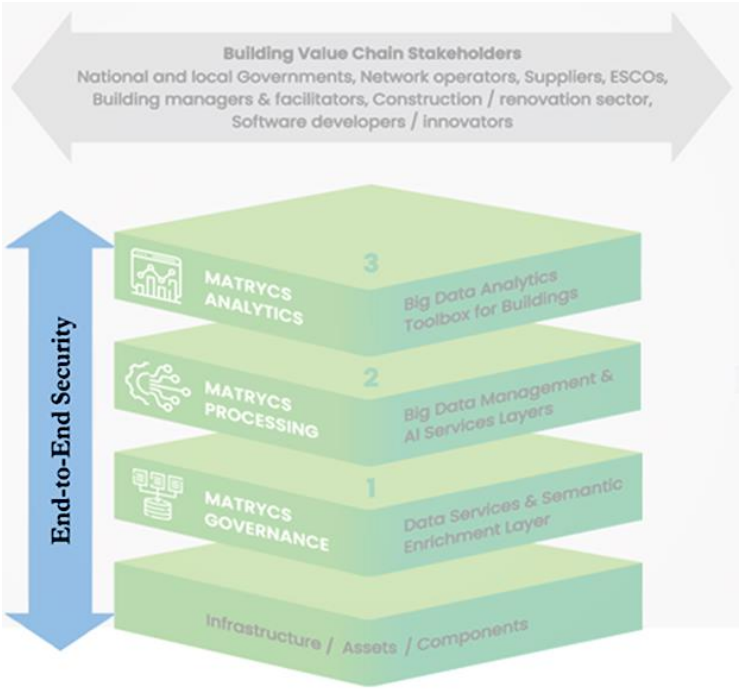

Figure 15: End-to-End Security Framework in high-level MATRYCS architecture, as shown in *D2.3*.

3.1 Design

The End-to-End Security Framework is designed as a composition of a security toolbox (interchangeably referred to as the End-to-End Security Framework) and a practical set of security guidelines and

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

recommendations to be applied during the DevSecOps process (see *Figure 16*), thus considering the relevant software tools while also recognizing applicable security standards, practices, and processes. The framework is placed into an iterative context due to the inherent characteristics of the security process; throughout the duration of the MATRYCS project, the potential security concerns will be continuously evaluated with risk identification triggering the feedback loop and potential recalibration of the guidelines and the related toolbox adaptations. Whereas this chapter mainly considers the security toolbox, chapter 5 examines the framework's proposed security guidelines and recommendations to be adopted by the MATRYCS consortium.

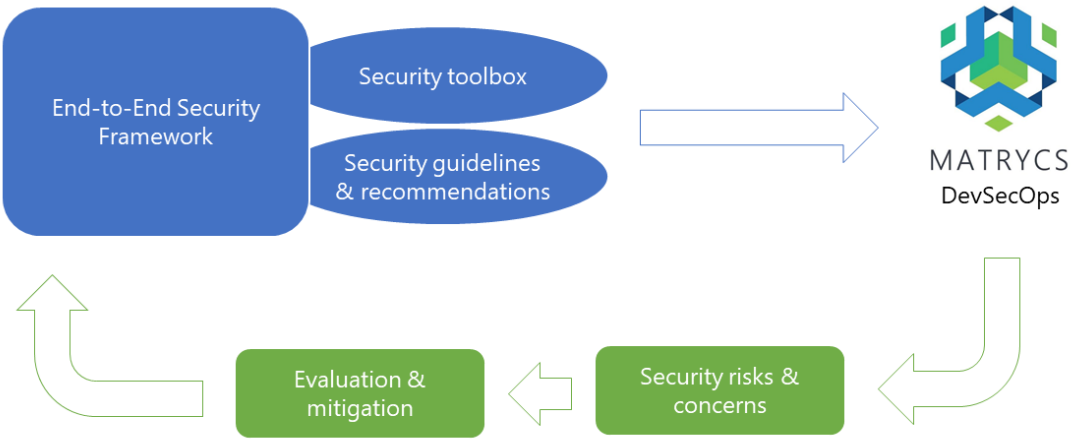

Figure 16: End-to-End Security Framework composition.

The End-to-End Security Framework toolbox, a generic, extensible and pluggable multi-layered security software toolbox, provides relevant privacy/security mechanisms with respect to anonymization, authentication, authorization, auditing, encryption, and software vulnerabilities/flaws detection and mitigation. It builds upon access control mechanisms with auditing capabilities as a foundation corresponding to identification, authentication and fine-grained authorization policies. On application level, service mesh composition for secure service provisioning and integration is employed. The architecture of the framework toolbox is shown in *Figure 17*. The solution is based on four pillars, which are described in the following: Persistent data layer, Privacy and security, Identity and access management, and API/GUI access layer.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

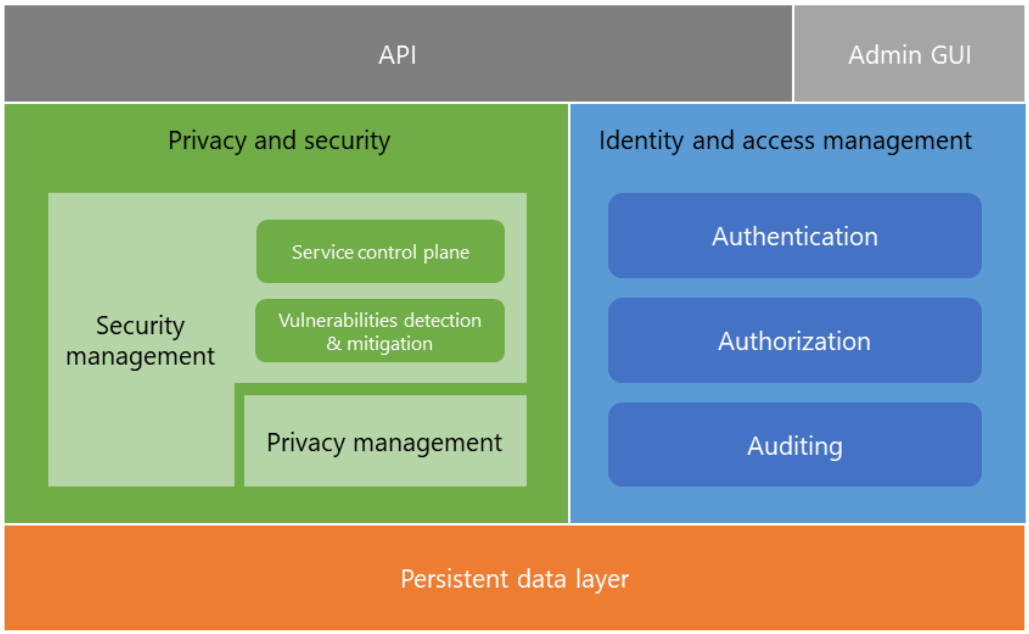

Figure 17: MATRYCS End-to-End Security Framework architecture.

Persistent data layer

The Persistent data layer implements an isolated secure data storage for confidential and other critical information. It should support granular specification of access and control policies in addition to encrypted storage with data integrity protection. Maintenance of high reliability and recovery in case of catastrophic events is required. The Persistent data layer mainly serves as a generic storage solution for the upper layers of the End-to-End Security Framework (e.g., Identity and access management), but could – if required – be extended to grant security storage capacities to other components. In general, this layer should be technology-agnostic with the extensibility enabled by swapping technological solutions.

Privacy and security

The Privacy and security module is comprised of two parts with somewhat related functionalities; whereas the Privacy management submodule is concerned with various aspects of anonymization, (regulatory) compliance and enforcement of privacy policies, the Security management submodule considers the establishment and enforcement of security policies. As MATRYCS opts for a microservice architecture, these policies mostly relate to application-level security provisioning and secure service integration. For this, a service control plane for controlling and monitoring services and their data flows is proposed. The plane should additionally prevent data compromise via ensuring encrypted service-to-service communication and data encryption at rest. Moreover, this submodule should detect and mitigate security risks in the form of vulnerabilities and flaws on an application, database, network and system level. The Privacy and security module should therefore keep and up-to-date database of potential software security risks. Furthermore, security and privacy standards must be considered while also paying special attention to local, national and international legislation.

Identity and access management

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

The Identity and access management module is responsible for authentication, authorization, and auditing aspects. Authentication delivers means for identification of entities and establishment of trust between them in the system, which serves as groundwork upon which fine-grained data/service/asset authorization policies and usage control mechanisms may be defined, applied, and enforced. The related auditing functionality should provide a comprehensive system of event logging enabling compliance assessment and traceability. This module should be extensible, provide sufficient programming interfaces and comply with established authentication/authorization standards (e.g., OAuth).

API/GUI access layer

The API/GUI access layer exposes the End-to-End Security Framework functionalities to other components and/or users. It is composed of an admin GUI, a user interface aimed at the MATRYCS operators for enabling a more straightforward graphical overview and configuration of the security layer, and an API component, which exposes all underlaying layers of the framework to other technological MATRYCS components and infrastructure. The access layer should implement secure authentication, possibly multi-factor, and authorization mechanisms being the main entry point to protected framework areas. The API should conform to established standards and architectural styles, such as REST.

3.2 Technologies

The End-to-End Security Framework for specific parts of its architecture currently adopts several technological components. Persistent data layer for providing internal data storage employs PostgreSQL open-source relational database management system. Keycloak is adopted for authentication, authorization and auditing as a part of the Identity and access management as well as Privacy management submodules whereas Istio implements service control and monitoring plane in Security management submodule. Security management additionally provides Vulnerabilities detection & mitigation component. For this, several technologies are under consideration: Nikto2, W3AF, OpenVAS, Nmap, OpenSCAP, Aircrack, and GoLismero. Finally, API/GUI access layer is implemented by each component separately. The considered technological components are described in the rest of this section.

3.2.1 PostgreSQL

PostgreSQL[12] is an ACID-compliant open-source relational database management system (RDBMS) employing and extending the structured query language (SQL) standard. It can be ran on all major platforms and is extensible via add-ons as well as through feature-specific defined APIs. PostgreSQL is highly scalable with concurrency/clustering options and includes support for internationalisation and text search. Its features – among others – include:

- Support for various data types (primitives, structures, document-based, geometric, custom).
- Data integrity protection through keys, constraints and locks definition.

[12] https://www.postgresql.org

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- High performance and concurrency using advanced indexing, query planner, transactions, parallelization, table partitioning, transaction isolation and just-in-time (JIT) compilation.
- High reliability and disaster recovery using write-ahead logging (WAL), replication, point-in-time-recovery (PITR), active standbys and tablespaces.
- Multi-factor authentication, access-control system, column and row-level security.
- Extensibility through stored functions/procedures, support for procedural languages, customizable table interfaces and add-ons.

3.2.2 Keycloak

Keycloak[13] is an open-source identity and access management platform. Keycloak implements several access control mechanisms: attribute-based access control (ABAC), role-based access control (RBAC), user-based access control (UBAC), context-based access control (CBAC), rule-based access control, time-based access control, and custom access control mechanisms (ACMs) though a Policy Provider Service Provider Interface. It provides support for user federation on available standard authentication protocols, social and single sign on as well as identity brokering. In addition to an event logging system for auditing needs, Keycloak supplies admin management consoles. Integration is enabled through client adapter libraries.

Keycloak is based on a set of administrative UIs and a RESTful APIs and provides the necessary means to create permissions for resources and scopes. Associate those permissions with authorization policies and enforce authorization decisions in applications and services. Resource servers (applications or services serving protected resources) usually rely on some kind of information to decide if access should be granted on a protected resource. For RESTful-based resource servers, that information is usually obtained from a security token, usually sent as a bearer token on every request to the server. For web applications that rely on a session to authenticate users, that information is usually stored in a user's session and retrieved from there for each request.

Keycloak has the capability to fuse heterogeneous environments where users are distributed across different regions, with different local policies, using different devices and high demand for information sharing, Keycloak Authorization Services improve the authorization capabilities by providing:

- Resource protection using fine-grained authorization policies and different access control mechanisms.
- Centralized Resource, Permission and Policy Management.
- Centralized Policy decision point.
- REST Security based on a set of REST-based authorization services.
- Authorization workflows and User-Managed Access.

[13] https://www.keycloak.org

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

3.2.2.1 Keycloak Authorization Process

In order to use Keycloak and enable a fine-grained authorization on applications, three main processes are defined:

- **Resource Management:** A resource can be a web page, a RESTful resource, a file in a file system and so on. Using Keycloak Administration REST APIs it is possible to secure the resources and grant access to specific users over them. For instance, users having a specific role will have access over a specific resource. Resources are managed through Resource server, which are Keycloak clients for handling resources attributes and functionalities.
- **Permission and Policy Management:** Once the Resource server and resources have been defined permissions and policies are needed to define the security and access requirements that govern the resources. Policies define the conditions that must be satisfied to access or perform operations on resources, but they are not tied to what they are not protecting. They are generic and can be reused to build permissions or even more complex policies. After policies definition, the next step is to define permissions. Permissions are coupled with the resource they are protected.
- **Policy Enforcement:** Involves the necessary steps to actually enforce authorization decisions to a resource server. This is achieved by enabling a Policy Enforcement Point that is capable of communicating with the authorization server, ask for authorization data and control access to protected resources based on the decisions and permissions returned by the server.

Keycloak Authorization Services consist of Token, Resource Management and Permission Management Endpoints. The Token Endpoint is used to obtain access tokens from Keycloak and use them to access resources protected from Resource Servers. The Resource Management Endpoint is used to create, delete and query resources. Finally, the Permission Management REST APIs are used to issue permission tickets that represent the permissions being requested by the clients. The Keycloak authorization process is outlined in *Figure 18*.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

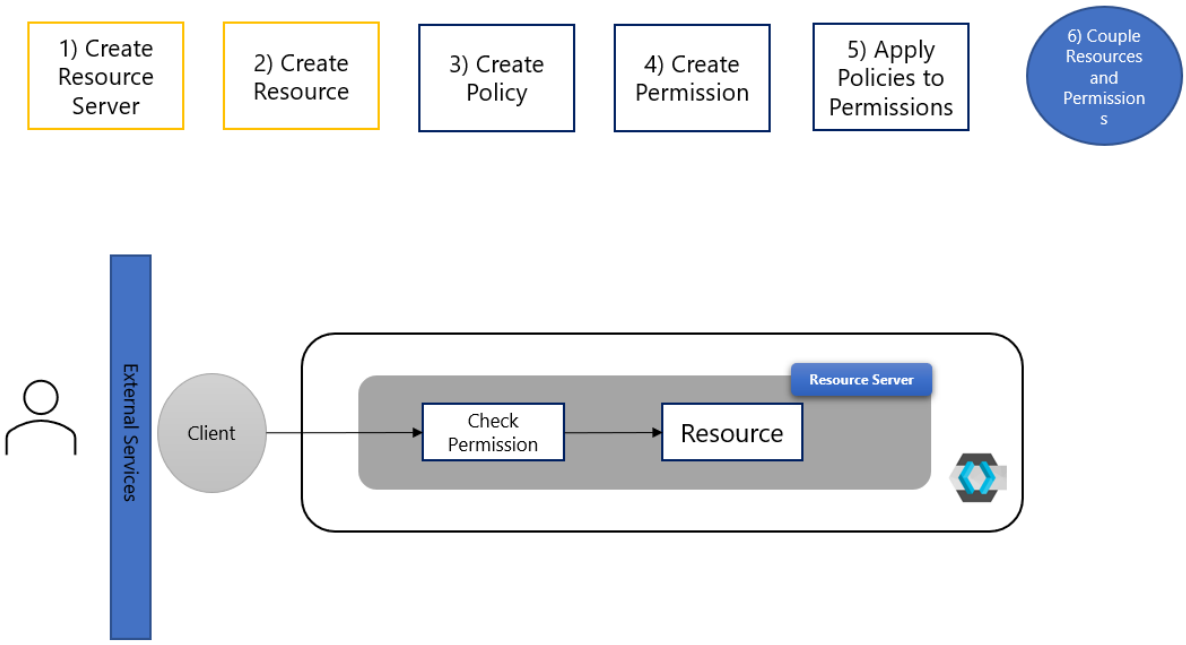

Figure 18: Keycloak authorization process.

3.2.3 Istio

Istio[14] is an open-source service mesh implementation with support for multiple platforms (Azure, VMs, …) and orchestration systems (Kubernetes, Minikube, …). It provides the following functionalities:

- automatic service discovery,
- pluggable policy layer and configuration API,
- automated metrics and logs collection,
- granular traffic control,
- load balancing,
- secure service-to-service communication.

Istio implements a data plane, a layer for controlling and monitoring inter-service communication through a set of sidecar proxies, and a control plane for managing and monitoring the sidecar proxies. The Istio traffic management, based on Envoy proxy model, enables control of traffic and API calls using routing rules. Additionally, the robustness of services and network is provided through failure recovery features and transparent service-level properties configuration, such as load balancing, circuit breakers and failure timeouts/retries. The integral building block of traffic management is represented by the so-called virtual services. Coupled with the destination routing rules, the virtual services enable flexible and comprehensive configuration of service mesh request routing. The virtual services are composed of a

[14] https://istio.io

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

set of sequential routing rules, which delineate the addressed application services.

As the service mesh operates, a multitude of events are present in the system. For this, Istio employs a comprehensive telemetry functionality, thus providing a higher level of system observability. In principle, Istio produces three distinct types of telemetry:

- **Metrics:** Aggregate monitoring of the service mesh with respect to proxy-level, service-level, and control plane metrics. Includes information on latency, traffic, errors, and saturation.
- **Distributed traces:** Monitoring and tracing of requests as they pass through the service mesh. Istio supports several tracing backends (e.g., Datadog, Zipkin, …).
- **Access logs:** Service monitoring on the level of an individual workload instance, including full metadata assigned to each request.

Finally, Istio may be extended through extensions based on WebAssembly sandboxing technology and the related Proxy-Wasm API, which are aimed at the Istio proxy – Envoy.

3.2.3.1 Istio Security Management

Istio covers various security aspects with respect to internal and external attacks on services and data, encompassing the complete platform, endpoints, and communication of the service mesh. It implements the in-depth defence, zero-trust, and security-by-default concepts to enable multi-layered defence integration on distrusted networks with no software/infrastructure modifications. Istio employs identity management considering X.509 certificates in connection with the first-class *service identity* model, which enables flexible and granular identity assignment. The periodic process applied for certificates/keys rotation and refresh as well as identity provisioning is as follows:

i. A gRPC service is sent a certificate signing request (CSR) by Istio daemon.
ii. Istio agents generate a key pair and send the constructed CSR to Istio daemon.
iii. Istio daemon validates the CSR and signs the CSR for certificate generation.
iv. Sidecar proxy requests the certificate and related key from the Istio agent when the service is started.
v. The Istio agent retrieves the certificate and the related key from Istio daemon and forwards it to the sidecar proxy.

The certificates are managed by an internal Istio certificate authority (CA). Each sidecar proxy, attached to a distinct service, implements a Policy Enforcement Point (PEP), which receives authentication and authorization policies from the Istio configuration server. Based on that, the PEP manages inter-service encryption and flexible access control based on access policies. The authentication in Istio is implemented either as peer authentication, which performs service-to-service authentication based on mutual TLS, and request authentication, which performs user request authentication based on JSON Web Token (JWT) validation or by utilizing custom authentication providers. The authentication policies, stored and circulated by the configuration server, are applied to each request received by a service. An example of an authentication policy is available in *Table 7*. Similarly, the configuration server stores and circulates the authentication policies among sidecar proxies. The authorization policies administer access

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

control to inbound sidecar proxy traffic, enabling service-to-service and user-to-service authorization on mesh, namespace, or service level. By default, i.e., if no authorization policy is in place, all traffic is permitted. If multiple policies are specified, they are evaluated in order with respect to the precedence. The policy is composed of a selector specifying the target of the policy, the action which specifies the outcome of policy evaluation, and the rules defining the action triggers. An example of an authorization policy is available in *Table 8*. Finally, Istio implements auditing functionalities by exploiting sidecar proxy telemetry extensions.

Table 7: Istio authentication policy example.

```
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: MATRYCS-auth-policy1
  namespace: ns1
spec:
  selector:
    matchLabels:
      app: service1
  mtls:
    mode: STRICT
```

Table 8: Istio authorization policy example.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: MATRYCS-autz-policy1
 namespace: ns1
spec:
 selector:
   matchLabels:
     app: service1
action: ALLOW
 rules:
 - from:
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
- source:
    notNamespaces: ["ns2"]
to:
- operation:
    methods: ["POST"]
```

3.2.4 Vulnerability Detection and Mitigation

Security management submodule in the End-to-End Security Framework aims to also provide vulnerabilities detection and mitigation functionalities. Due to the specifics of such functionalities and their coupling with specific software and system deployments, the appropriate tools are – at the time of writing this deliverable – still under consideration and will be reported in the relevant upcoming deliverable, i.e., D3.3 – *MATRYCS-GOVERNANCE (2nd technology release)*. Whereas most tools may be executed on any platform, different tools target various system or software components and provide somewhat differing functionalities. Most studied vulnerabilities detection and mitigation tools, in general, are aimed at web applications and servers with some also targeting networks, databases and operating systems. The tools mainly provide vulnerabilities scanning, risk assessment and countermeasures suggestion functionalities. A preliminary analysis and overview of examined open-source technological components is presented in *Table 9*.

Table 9: Overview of vulnerability detection/mitigation tools.

Tool	Platform	Target	Main functionalities
Nikto2	Any	Web applications and servers	Web server and configuration issues scans, outdated versions and dangerous files detection
W3AF	Any	Web applications	Vulnerability and exploitation scanning, penetration testing
OpenVAS	Any	Web applications and servers, databases, operating systems, virtual machines, networks	Vulnerability scanning, risk assessment, countermeasures recommendations
Nmap	Any	Network	Vulnerability scanning, network probing
OpenSCAP	Linux	Web applications and servers, databases, operating systems, virtual machines, networks	Vulnerability scanning and measurement, risk assessment, security measures, treat countermeasures
Aircrack	Any	Network (WiFi)	Security assessment, network

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

			monitoring, testing and auditing
GoLismero	Any	Web applications, databases, networks	Vulnerabilities scanning, treat countermeasures

3.3 Initial Implementation

The End-to-End Security Framework aims to adhere to the MATRYCS project's common deployment approach, which will enable easier integration and later deployment in target cloud-based environments following experimentation, composition and testing on local development machines. The common approach is based on Docker[15] virtualization technology, a comprehensive set of tools employing OS-level virtualization for software containerization. The initial framework implementation contains the following technological components, which – based on the feedback evaluation procedure during the duration of the project outlined in *section 3.1* – might be extended or adapted: PostgreSQL, Keycloak and Istio. Moreover, the initial implementation of the framework will be enhanced with the applicable vulnerability detection/mitigation toolset selected. The initial docker-compose deployment configuration, which orchestrates the End-to-End Security Framework tools instantiation and exposes the security services on designated ports, is available in *Table 10* whereas *Table 11* provides the related Istio policy configuration.

Table 10: End-to-End Security Framework docker-compose configuration.

```
version: "3.8"
services:
  keycloak:
    image: quay.io/keycloak/keycloak:12.0.4
    ports:
      - 8080:8080
    environment:
      - KEYCLOAK_USER=admin
      - KEYCLOAK_PASSWORD=admin
      - KEYCLOAK_IMPORT=/tmp/realm-export.json
    volumes:
      - ./docker/keycloak/realm-export.json:/tmp/realm-export.json
  postgresql:
    image: postgres:13-alpine
    ports:
```

[15] https://www.docker.com/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
    - 5432:5432
  environment:
    POSTGRES_USER: "postgres"
    POSTGRES_PASSWORD: "postgres"
    POSTGRES_DB: "db"
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d ${POSTGRES_DB}"]
    interval: 10s
    timeout: 5s
    retries: 20
```

Table 11: Istio policy configuration.

```
- apiVersion: authentication.istio.io/v1alpha1
  kind: Policy
  metadata:
    name: customerjwt
    namespace: abc-customer
  spec:
    targets:
      - name: customer
      - name: preference
      - name: recommendation
    peers:
      - mtls: {}
    peerIsOptional: ~
    origins:
      - jwt:
          audiences:
            - customer
          issuer: 'https://sso.abc.com/auth/realms/customer'
          jwksUri: 'https://sso.abc.com/auth/realms/customer/protocol/openid-connect/certs'
    principalBinding: USE_ORIGIN
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

4 Security Integration in MATRYCS Ecosystem

The MATRYCS End-to-End Security Framework interacts directly or indirectly with the complete MATRYCS ecosystem and is thus considered a generic MATRYCS security layer, covering *MATRYCS-GOVERNANCE*, *MATRYCS-PROCESSING*, *MATRYCS-ANALYTICS* and other infrastructure, assets, or users. The framework implements two different access pathways, as depicted in *Figure 19*:

- **GUI access:** The framework provides a graphical user interface for relevant functionalities aimed at MATRYCS system administrators. Admin GUI is accessible via a web browser to authorized users and enables monitoring/configuration of the Identity and access management module based on Keycloak. Among the supported functionalities are creation of user accounts and groups, definition of authentication/authorization policies, configuration of authentication/authorization aspects, and auditing of events in the system.
- **API call:** The framework exposes an interface for privacy and security services/tools via application programming interfaces (APIs) to various modules and services in the MATRYCS architecture. The interface is module agnostic, signifying any component, service or even infrastructural asset may consume the security services in a unified manner. The APIs enable programmatical access to the Identity and access management module based on Keycloak, supporting an extended set of functionalities provided by the GUI access with respect to authentication, authorization and auditing. Additionally, the APIs facilitate and coordinate security/privacy management aspects by exposing service control (Istio) and vulnerability detection/mitigation tools.

The MATRYCS modules will integrate with the End-to-End Security Framework via the framework's API layer. Although most privacy and security features will be provided through the designated interfaces, some aspects, such as vulnerability detection/mitigation, additionally require direct module integration. This will be realized through the use of plugins; the framework will implement integrable software extensions to be optionally adopted by MATRYCS modules for providing extended security/privacy features alongside consolidated framework API interactions. The definitive interactions, integration guidelines and usage examples will be provided in conjunction with the plugin implementation whereas the interactions and APIs implemented by the adopted tools are described in each respective tool's documentation – pointers given in *section 3.2*.

Whereas the initial implementation of the End-to-End Security Framework has been deployed on-premises, the following deployments are planned for public cloud-based environments, thus streamlining the integration and deployment phases. A transparent transition towards cloud environments is enabled by the application of virtualization technologies in the development process, also considering common deployment approaches. As the MATRYCS project opts for the EGI-ACE infrastructure[16], the framework is envisioned to be deployed in general purpose cloud compute instance including Docker, exposing the framework's functionalities over its API layer on a public IP for utilization by the integrated MATRYCS modules, services, and assets. A preliminary specification of required cloud resources by the framework installation, obtained using an empirical analysis and considering

[16] https://www.egi.eu/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

integration assumptions/requirements of other MATRYCS components, is presented in *Table 12*.

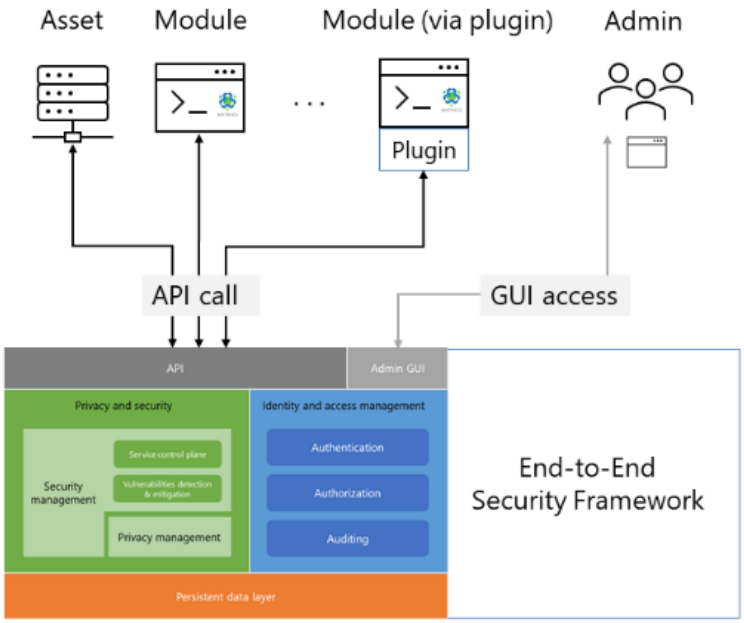

Figure 19: End-to-End Security Framework interactions.

Table 12: End-to-End Security Framework cloud resource requirements.

Number of CPU cores	Amount of RAM per core (GB)	Local disk (GB)	Number of VM instances	Number of public IPs	Volume (GB)
4	2	20	1	1	50

4.1 Keycloak and MATRYCS Toolbox

Keycloak Authorization Process and Keycloak Authorization Services, as described in *section 3.2.2.1*, will be used to secure MATRYCS Toolbox and its components. More specifically the Frontend Components offered in SaaS layer (Advanced Visualizations & Reports Engine, MATRYCS Analytics Services, Digital Building Twin) and the Virtual Workbench will be integrated with Keycloak in order to add the Authorization layer on the top of them (see *Figure 20*). More specifically Role-based policies will be created in order to grant permissions over resources, in MATRYCS case the resources will be the analytical services accessed from users. Keycloak Resource servers will check user permissions and the coupled role-based policies and according to their roles and the information that is obtained and introspected from user's Oauth2 Access Token will control the resource access. The following picture demonstrates the connection of MATRYCS toolbox services through clients to secure their interfaces.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

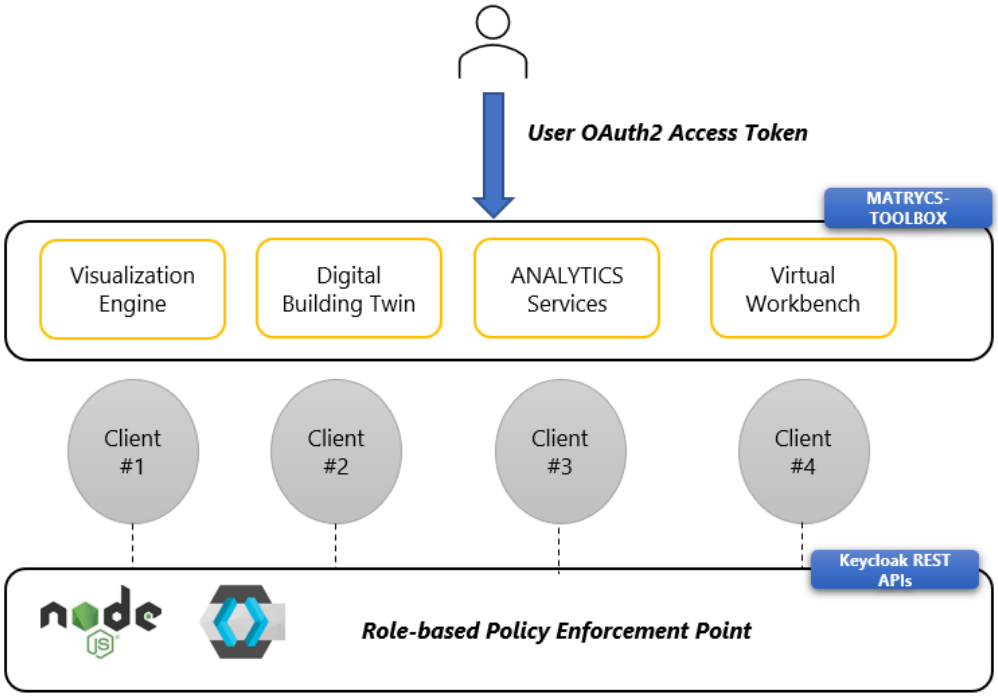

Figure 20: MATRYCS Toolbox integration with Keycloak.

KEYCLOAK REST APIs

The Keycloak REST APIs that will be used are demonstrated in *Table 13*. The provided collection is used to get user's OAuth2 access token, to Introspect and resolve user's roles and attributes, to retrieve the list of resources that exist on a Keycloak Resource Server, to create a role-based permission over a resource, to get the list of permissions and get details for a provided resource.

Table 13: Keycloak REST APIs.

Get User Token
POST /auth/realms/${REALM} /protocol/openid-connect/token HTTP/1.1
Host: ${HOST}:8080
Content-Type: application/x-www-form-urlencoded
grant_type=password&client_id=${CLIENT_ID}&client_secret=${CLIENT_SECRET}&scope=profile&username=${USERNAME}&password=${PASS}
Introspect User Token
POST /auth/realms/${REALM}/protocol/openid-connect/token/introspect/ HTTP/1.1
Host: ${HOST}:8080
Content-Type: application/x-www-form-urlencoded

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

client_id=${CLIENT_ID}&client_secret=${CLIENT_SECRET}&token=${USER_ACCESS_TOKEN}

Get List of Resources

GET /auth/realms/${REALM}/authz/protection/resource_set/ HTTP/1.1

Host: ${HOST}:8080

Authorization: ${RESOURCE_SERVER_CREDENTIALS}

Create Permission over a Resource

POST /auth/realms/${REALM}/authz/protection/uma-policy/${RESOURCE_ID}

Host: ${HOST}:8080

Authorization: $

Content-Type: application/json

{

"name":"${NAME}",

"description":"${DESCRIPTION}",

"roles":["${ROLE} "]

}

Get Resource details

GET /auth/realms/${REALM}/authz/protection/resource_set/${RESOURCE_ID}

Host: ${HOST}:8080

Authorization: ${RESOURCE_SERVER_CREDENTIALS}

4.2 Istio and Generic MATRYCS Services

Istio, adopted as a part of the Security management submodule of the End-to-End Security Framework, provides service control and monitoring capacities, enabling encrypted service-to-service communication, definition of authentication and authorization policies, and automated metrics/logs collection. Istio tool and the related security management aspects, as described in *section 3.2.3.1*, will be employed to provide secure service mesh functionalities to implemented MATRYCS modules, software solutions and services. Specifically, the service mesh shall be made available to the modules of *MATRYCS-GOVERNANCE*, *MATRYCS-PROCESSING* and *MATRYCS-ANALYTICS*, especially focusing on AI/ML services.

The End-to-End Security Framework deployment will include a common managed Istio daemon control plane. In principle, the MATRYCS module/service owners may decide to adopt Istio on their own, apply other similar security solution or mechanism, or utilize the common Istio deployment of the End-to-End Security Framework. For the latter, we assume a Kubernetes deployment scenario with a remote cluster; the Envoy sidecar proxies applied to modules/services utilize an ingress gateway to access common Istio daemon. The steps for deploying and connecting the example service *matrycs-service* are listed in *Table 14*. The procedure consists of two parts. First, the application/service is locally deployed using

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Kubernetes. Next, the appropriate gateways on the cluster are enabled and the deployed service is exposed. In the example, the *REMOTE_CTX* refers to the name of the remote cluster's context.

Table 14: Common Istio deployment connection procedure.

1. Create the namespace and deploy the service

> kubectl create --context="${REMOTE_CTX}" namespace matrycs

> kubectl label --context="${REMOTE_CTX}" namespace matrycs istio-injection=enabled

> kubectl apply -f matrycs-service.yaml -l service=matrycs-service -n matrycs --context="${REMOTE_CTX}

2. Create ingress and egress gateway configuration file

Filename: ingress-egress.yaml

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  profile: empty
  components:
    ingressGateways:
    - namespace: external-istiod
      name: istio-ingressgateway
      enabled: true
    egressGateways:
    - namespace: external-istiod
      name: istio-egressgateway
      enabled: true
  values:
    gateways:
      istio-ingressgateway:
        injectionTemplate: gateway
      istio-egressgateway:
        injectionTemplate: gateway
```

3. Enable ingress and egress gateways

> istioctl install -f ingress-egress.yaml --context="${REMOTE_CTX}"

4. Create service gateway configuration file

Filename: matrycs-service-gateway.yaml

```
apiVersion: networking.istio.io/v1alpha3
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

kind: Gateway

metadata:

name: matrycs-service-gateway

spec:

selector:

istio: ingressgateway

servers:

[insert service configuration]

---

apiVersion: networking.istio.io/v1alpha3

kind: VirtualService

metadata:

name: matrycs-service

spec:

hosts:

- ""*

gateways:

- matrycs-service-gateway

http:

[insert service configuration]

5. Expose the service on the gateways

> kubectl apply -f matrycs-service-gateway.yaml -n matrycs --context="${REMOTE_CTX}"

6. Set the gateway URL

> export INGRESS_HOST=$(kubectl -n external-istiod --context="${REMOTE_CTX}" get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')

> export INGRESS_PORT=$(kubectl -n external-istiod --context="${REMOTE_CTX}" get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].port}')

> export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

5 Security Guidelines and Recommendations

To secure the MATRYCS ecosystem, apt security standards must be ensured by means of applying adequate measures throughout the complete lifecycle of the project, from development up to and including operational phases. The iterative nature of ensuring security, which is characterized by continuous identification and mitigation of risks, is captured in the essence of the End-to-End Security Framework; the feedback loop, described in *section 3.1*, upon identifying and evaluating security concerns triggers a potential recalibration of the framework. The framework, in addition to a security toolbox, consists of a set of practical security guidelines and recommendations for the DevSecOps process, which should recognize appropriate security practices, standards and processes. The guidelines should streamline the inclusion of best security practices on various implementation levels while also providing direct recommendations on realistic inclusion of relevant security processes.

In *Table 15*, a collection of proposed security guidelines and recommendations to be adopted by the MATRYCS consortium – specifically developers and operators of the MATRYCS modules, assets, and infrastructure – is available. The guidelines and recommendations have been identified upon reviewing the related work, which is denoted in the table, and focus on general as well as technical aspects covered in MATRYCS, i.e., services, systems, and networks.

Table 15: Security guidelines and recommendations.

Category	ID	Description	Source
General	GR_GE1	Regularly apply software/system updates.	[7]
	GR_GE2	Implement and configure fine-grained authorization and apply the principle of least privilege.	[7]
	GR_GE3	Do not reuse credentials.	[7]
	GR_GE4	Encrypt data at rest.	[7]
	GR_GE5	Store credentials securely.	[30]
	GR_GE6	Define access control policies.	[30]
	GR_GE7	Maintain optimal condition of computer hardware and software.	[30]
	GR_GE8	Implement auditing mechanisms and adopt related tools.	[31]
	GR_GE9	Perform continuous monitoring of the security status.	[31]
Service	GR_SE1	Avoid vulnerable programming languages and libraries.	[7]
	GR_SE2	Perform code analysis, also manual.	[7]

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

	GR_SE3	Use applicable software engineering practices, e.g., data validation and error handling.	[7]
	GR_SE4	Apply microservice design.	[7]
	GR_SE5	Apply comprehensive event and action auditing mechanisms.	[29]
Network	GR_NE1	Use standard and verified protocols, e.g., TLS.	[7]
	GR_NE2	Protect service-to-service communication using mutual TLS and internal PKI.	[7]
	GR_NE3	Apply principal propagation via security tokens, e.g., JWT.	[7]
	GR_NE4	Apply appropriate firewall policies.	[30]
	GR_NE5	Encrypt electronic communication.	[30]
	GR_NE6	Evaluate data access control and data security at rest, in transit and in use.	[31]
System	GR_SY1	Use trusted and reliable hardware with hardware security modules.	[7]
	GR_SY2	Use deployment models with strong isolation, i.e., virtualization.	[7]
	GR_SY3	Use trusted and reliable orchestration platforms with a secure service discovery/registry implementation.	[7]
	GR_SY4	Ensure physical protection of hardware.	[29]
	GR_SY5	Implement appropriate backup and disaster recovery mechanisms.	[29]
	GR_SY6	Use anti-viral software with automated updating.	[30]
	GR_SY7	Use uninterruptible power supply on critical systems.	[30]
	GR_SY8	Ensure sufficient safeguards for authentication, authorization, and identity/access management.	[31]

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

6 Conclusions

The MATRYCS project, which aims to provide an AI-powered framework for real-life building energy applications, heavily relies on modern information and communication technologies. Consequently, there is a need for a vertical security layer ensuring appropriate security standards throughout all of the phases, from development to operational. This is achieved by the development of the End-to-End Security Framework described in this deliverable, a composition of a software toolbox and practical set of security guidelines and recommendations to be applied during the DevSecOps process. The toolbox, built using virtualization technologies, bases on Keycloak and Istio tools that enable authentication, authorization, auditing and enforcement of various security/privacy aspects on application level. Furthermore, the framework will adopt a vulnerability detection and mitigation toolset. The integration of the framework with the MATRYCS technology stack is facilitated using a set of APIs and the related security plugin. Finally, the admin GUI provides graphical access to relevant functionalities aimed at MATRYCS system administrators.

In the following, the envisioned future activities carried out as a part of the T3.6 *End-to-End Security Framework* are listed. These activities will be reported in the relevant upcoming deliverables, i.e., D3.3 – *MATRYCS-GOVERNANCE (2nd technology release)* and D3.4 – *MATRYCS-GOVERNANCE (final technology release)*.

Future activities

- Selection and adoption of an appropriate vulnerability detection and mitigation toolset.
- Development of the End-to-End Security Framework plugin for extended security/privacy feature integration.
- Release of the final MATRYCS End-to-End Security Framework implementation.
- Finalization and harmonization of the integration aspects (e.g., required authentication and roles) with other technical WPs and module owners.
- Deployment of the End-to-End Security Framework on the selected public cloud infrastructure.
- Continuous monitoring as well as potential refinement of project's technical security aspects and support to consortium with respect to inclusion of security guidelines and integration of the End-to-End Security Framework.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

References

[1] 14:00-17:00, "ISO/IEC 27000:2018," *ISO*. https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/39/73906.html (accessed Nov. 02, 2021).

[2] S. Samonas and D. Coss, "The CIA Strikes Back: Redefining Confidentiality, Integrity and Availability in Security," *Journal of Information System Security*, vol. 10, no. 3, pp. 21–45, 2014.

[3] E. Asanghanwa and R. Ih, "Security for Intelligent, Connected IoT Edge Nodes," p. 10.

[4] J. R. Vacca, *Public Key Infrastructure: Building Trusted Applications and Web Services*. CRC Press, 2004.

[5] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf, and Y. A. Bangash, "An In-Depth Analysis of IoT Security Requirements, Challenges, and Their Countermeasures via Software-Defined Security," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10250–10276, Oct. 2020, doi: 10.1109/JIOT.2020.2997651.

[6] I. Indu, P. M. R. Anand, and V. Bhaskar, "Identity and access management in cloud environment: Mechanisms and challenges," *Engineering Science and Technology, an International Journal*, vol. 21, no. 4, pp. 574–588, Aug. 2018, doi: 10.1016/j.jestch.2018.05.010.

[7] T. Yarygina and A. H. Bagge, "Overcoming Security Challenges in Microservice Architectures," in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, Mar. 2018, pp. 11–20. doi: 10.1109/SOSE.2018.00011.

[8] M. Waseem, P. Liang, M. Shahin, A. Ahmad, and A. R. Nassab, "On the Nature of Issues in Five Open Source Microservices Systems: An Empirical Study," in *Evaluation and Assessment in Software Engineering*, New York, NY, USA, Jun. 2021, pp. 201–210. doi: 10.1145/3463274.3463337.

[9] S. Dilmaghani, M. R. Brust, G. Danoy, N. Cassagnes, J. Pecero, and P. Bouvry, "Privacy and Security of Big Data in AI Systems: A Research and Standards Perspective," in *2019 IEEE International Conference on Big Data (Big Data)*, Dec. 2019, pp. 5737–5743. doi: 10.1109/BigData47090.2019.9006283.

[10] Y. Mirsky *et al.*, "The Threat of Offensive AI to Organizations," *arXiv:2106.15764 [cs]*, Jun. 2021, Accessed: Nov. 02, 2021. [Online]. Available: http://arxiv.org/abs/2106.15764

[11] E. Krishnasamy, S. Varrette, and M. Mucciardi, "Edge Computing: An Overview of Framework and Applications," p. 20.

[12] I. Voras *et al.*, "Evaluating open-source cloud computing solutions," in *2011 Proceedings of the 34th International Convention MIPRO*, May 2011, pp. 209–214.

[13] A. Kurbatov, "Design and implementation of secure communication between microservices," Jan. 2021, Accessed: Nov. 02, 2021. [Online]. Available: https://aaltodoc.aalto.fi:443/handle/123456789/102405

[14] A. Khatri, V. Khatri, D. Nirmal, H. Pirahesh, and E. Herness, *Mastering Service Mesh: Enhance, secure, and observe cloud-native applications with Istio, Linkerd, and Consul*. Packt Publishing, 2020.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

[15] S. Shahzadi, M. Iqbal, Z. U. Qayyum, and T. Dagiuklas, "Infrastructure as a service (IaaS): A comparative performance analysis of open-source cloud platforms," in *2017 IEEE 22nd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, Jun. 2017, pp. 1–6. doi: 10.1109/CAMAD.2017.8031522.

[16] S. Ismaeel, A. Miri, D. Chourishi, and S. M. Reza Dibaj, "Open Source Cloud Management Platforms: A Review," in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, Nov. 2015, pp. 470–475. doi: 10.1109/CSCloud.2015.84.

[17] L. D. Kurup, C. Chandawalla, Z. Parekh, and K. Sampat, "Comparative Study of Eucalyptus, Open Stack and Nimbus," vol. 4, no. 6, p. 5, 2015.

[18] "Best Practices in Implementing a Secure Microservices," *CSA*. https://cloudsecurityalliance.org/artifacts/best-practices-in-implementing-a-secure-microservices-architecture/ (accessed Nov. 02, 2021).

[19] G. Conway, "Introduction to Cloud Computing," Jan. 2011. https://mural.maynoothuniversity.ie/2970/ (accessed Nov. 02, 2021).

[20] P. K. Chouhan, F. Yao, and S. Sezer, "Software as a service: Understanding security issues," in *2015 Science and Information Conference (SAI)*, Jul. 2015, pp. 162–170. doi: 10.1109/SAI.2015.7237140.

[21] "Software Reuse in the Era of Opportunistic Design." https://ieeexplore.ieee.org/document/8693072 (accessed Nov. 02, 2021).

[22] D. Pöhn and W. Hommel, "Universal Identity and Access Management Framework for Future Ecosystems," vol. 12, pp. 64–84, Mar. 2021, doi: 10.22667/JOWUA.2021.03.31.064.

[23] E. Maler, M. Machulak, and J. Richer, "User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization," Jan. 07, 2018. https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-grant-2.0.html (accessed Nov. 02, 2021).

[24] "What's a service mesh?" https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh (accessed Nov. 02, 2021).

[25] M. Klein, "Service mesh data plane vs. control plane," *Medium*, Oct. 10, 2017. https://blog.envoyproxy.io/service-mesh-data-plane-vs-control-plane-2774e720f7fc (accessed Nov. 02, 2021).

[26] R. Chandramouli, "Security strategies for microservices-based application systems," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-204, Aug. 2019. doi: 10.6028/NIST.SP.800-204.

[27] "The Best Programming Languages for Cybersecurity in 2021." https://flatironschool.com/blog/best-programming-languages-cyber-security/ (accessed Nov. 02, 2021).

[28] "CWE - Common Weakness Enumeration." https://cwe.mitre.org/index.html (accessed Nov. 02, 2021).

[29] R. Lehtinen and G. T. G. Sr, *Computer Security Basics*, 2nd edition. Sebastopol, CA: O'Reilly Media, 2006.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

[30] P. Schattner, C. Pleteshner, H. Bhend, and J. Brouns, "Guidelines for computer security in general practice," *Inform Prim Care*, vol. 15, no. 2, pp. 73–82, 2007, doi: 10.14236/jhi.v15i2.645.

[31] T. Grance and W. Jansen, "Guidelines on Security and Privacy in Public Cloud Computing," Dec. 2011, Accessed: Nov. 02, 2021. [Online]. Available: https://www.nist.gov/publications/guidelines-security-and-privacy-public-cloud-computing

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158