Cloud Orchestration at the Level of Application

Project Acronym: **COLA**

Project Number: **731574**

Programme: **Information and Communication Technologies**
Advanced Computing and Cloud Computing

Topic: **ICT-06-2016 Cloud Computing**

Call Identifier: **H2020-ICT-2016-1**
Funding Scheme: **Innovation Action**

Start date of project: 01/01/2017 Duration: 30 months

Deliverable:

D7.3 Design of application level security classification formats and principles

Due date of deliverable: 30/06/2018 Actual submission date: 25/06/2018

WPL:

Dissemination Level: PU

Version: 1.3

Status and Change History

Table 1 Status Change History

Status:	**Name:**	**Date:**	**Signature:**
Draft:	A. Michalas and N. Paladi	10/06/2018	A.Michalas/N. Paladi
Reviewed:	A. Marosi	18/06/2018	A. Marosi
Approved:	T. Kiss	24/06/2018	T. Kiss

Table 2 Document Change History

Version	Date	Pages	Author	Modification
V0.1	02/02	7	Hai-Van Dang	Document template
V0.2	02/02	13	Hai-Van Dang	First draft of Chapter 2
V0.2	07/02	13	Antonis Michalas	Review and changes to Chapter 2
V0.3	09/2	41	Hai-Van Dang	First draft of Chapter 3
V0.3	19/02	41	Antonis Michalas	Review and changes to Chapter 3
	27/02	11	Nicolae Paladi	Create template for deliverable 7.3
V0.4	09/03	32	Hai-Van Dang	Changes to section 3.3 – Attack vectors Add template Security Enabler Open Specification defined by SICS in Chapter 4
V0.5.1	16/03	52	Hai-Van Dang	Integrate the template and open specification for Image Verifier created by Nicolae in the current document
V0.5.2	20/03	66	Nicolae Paladi	Add updated use case partner requirements
V0.5.3	21/03	69	Nicolae Paladi	Add Open Specification for Crypto Engine
V0.6	22/03	72	Nicolae Paladi	Finalize draft Open Specification for Crypto Engine
V0.7	23/03	78	Antonis Michalas Hai-Van Dang	Finalize draft Open Specification for Credential Manager and Credential Store
V0.8	26/3	82	Nicolae Paladi	Move references from SICS enablers to end of document
V0.9	28/3	82	Hai-Van Dang	Add introduction and conclusion
V1.0	2/5	82	Antonis Michalas	Change to all sections
V1.1	3/5	81	Nicolae Paladi	Update 4.1.9 and 4.2.9
V1.2	7/5	105	Balint Kovacs	Update 4.3, 4.6 and 4.7
V1.3	8/5	106	Hai-Van Dang	Update table 6, section 4.5.7, and integrate all sections

Glossary

API	Application Programming Interface
AWS	Amazon Web Services
COLA	Cloud Orchestration at the Level of Application
UML	Unified Modelling Language
MiCADO	Microservice-based Cloud Application-level Dynamic Orchestrator
CM	Credential Manager
PM	Policy Manager
CSP	Cloud Service Provider
MAC	Message Authentication Code
HMAC	Hash-based Message Authentication Code
DoS	Denial of Service Attack
PII	Personally Identifiable Information
WN	Worker node
MN	Master node
HSTS	HTTP Strict Transport Security

Table 3 Glossary

List of Figures and Tables

Figure 1 MiCADO Architecture [1] 12
Figure 2 Interaction of the participating entities 17
Figure 3 Man-in-the-middle attack 21
Figure 4 Man-in-the-middle attack countermeasure 21
Figure 5 Password guessing attack 22
Figure 6 Password guessing countermeasure 23
Figure 7 Password reset MiTM attack 25
Figure 8 Password reset MiTM attack countermeasure 26
Figure 9 Certificate spoofing attack 27
Figure 10 Certificate spoofing countermeasure 28
Figure 11 Resource exhaustion as a result of impersonation attack 29
Figure 12 Resource exhaustion countermeasure 30
Figure 13 TOSCA file modification attack 31
Figure 14 TOSCA file modification attack countermeasure 31
Figure 15 Resource exhaustion due to TOSCA modification attack 32
Figure 16 Application sensitive information breach 33
Figure 17 Application sensitive information breach countermeasure 33
Figure 18 Application data breach 34
Figure 19 Open ports exploitation 34
Figure 20 Open ports exploitation countermeasure 35
Figure 21 Component interaction for the image integrity verifier 39
Figure 22 Component interaction for the credential manager in the use case CM-1 67
Figure 23 Component interaction for the credential manager in the use case CM-2 67
Figure 24 Component interaction for the credential manager in the use case CM-3 68
Figure 25 Component interaction for the credential manager in the use case CM-4 68
Figure 26 Component interaction for the credential manager in the use case CM-5 68
Figure 27 Initialize Credential Store 79
Figure 28 Write sensitive information to Credential Store 79
Figure 29 Read sensitive information from Credential Store 79
Figure 30 Write sensitive information to swarm 80
Figure 31 Read docker secret from swarm 80

Tables

Table 1 Status Change History 2

Table 2 Document Change History....3
Table 3 Glossary....4
Table 4 Communication Vulnerabilities of MiCADO....15
Table 5 Terms and definitions for authentication [6][21]....63
Table 6 Use case CM-1: Authentication....64
Table 7 Use case CM-1: Add new identity....65
Table 8 Use case CM-3: Change authenticator....65
Table 9 Use case CM-4: Reset authenticator....66
Table 10 Use case CM-5: Use case CM-5: Delete identity....66
Table 11 Credential table....68
Table 12 AccessLog table....69
Table 13 Protocol for authentication with lock-out functionality....69
Table 14 AccessConfig table....70
Table 15 Credential Manager - Test items....72
Table 16 Credential Manager - Test features....72
Table 17 Credential Manager - Features not to be tested....73
Table 18 Credential Manager - Test approach....73
Table 19 Use case CS-1: Initialize Credential Store....76
Table 20 Use case CS-2: Read/ write/ remove sensitive information....77
Table 21 Use case DS-1: Read docker secret from swarm....77
Table 22 Use case DS-2: Write secret to swarm and grant access right....78
Table 23 Use case DS-3: Remove docker secret....78

Table of Contents

Status and Change History 2
Glossary 4
List of Figures and Tables 5
Table of Contents 7
1 Introduction 10
2 Core components of MiCADO and data security requirements 12
2.1 MiCADO submitter 12
2.2 Cloud Orchestrator and Container Orchestrator 13
2.3 Policy Keeper 13
2.4 Monitoring System 14
2.5 External entities 14
2.6 Summary 14
3 Threat models and Attack vectors 17
3.1 Threat surface 17
3.2 Identified threat models 18
3.3 Attack vectors 20
3.3.1 Threat model 1: User impersonation 20
3.3.2 Threat model 2: TOSCA file modification 30
3.3.3 Threat model 3: Data Breach 32
3.3.4 Threat model 4: Open ports exploitation 34
4 Security Enablers Open Specification 36
4.1 Image Integrity Verifier Open specifications 36
4.1.1 Preface 36
4.1.2 Copyright 36
4.1.3 Legal notice 36
4.1.4 Terms and definitions 36
4.1.5 Overview 36
4.1.6 Basic concepts 37
4.1.7 Main interactions 37
4.1.8 Architectural drivers 39
4.1.9 Test plan 40
4.1.10 Re-utilised Technologies/Specifications 41
4.2 Crypto Engine: Open specifications 41
4.2.1 Preface 41
4.2.2 Copyright 42
4.2.3 Legal notice 42
4.2.4 Terms and definitions 42
4.2.5 Overview 42
4.2.6 Basic concepts 42

4.2.7 Main interactions 43
4.2.8 Architectural drivers 44
4.2.9 Test plan 46
4.2.10 Reused Technologies/Specifications 48
4.3 Security Policy Manager: Open specifications 48
4.3.1 Preface 48
4.3.2 Copyright 48
4.3.3 Legal notice 48
4.3.4 Terms and definitions 48
4.3.5 Overview 49
4.3.6 Basic concepts 49
4.3.7 Main interactions 49
4.3.8 Architectural drivers 56
4.3.9 Test plan 59
4.3.10 Re-utilised Technologies/Specifications 62
4.4 Credential Manager: Open specifications 62
4.4.1 Preface 62
4.4.2 Copyright 63
4.4.3 Legal notice 63
4.4.4 Terms and definitions 63
4.4.5 Overview 63
4.4.6 Basic concepts 63
4.4.7 Main interactions 64
4.4.8 Architectural drivers 71
4.4.9 Test plan 72
4.4.10 Re-utilised Technologies/Specifications 74
4.5 Crendential Store: Open specifications 74
4.5.1 Preface 74
4.5.2 Copyright 75
4.5.3 Legal notice 75
4.5.4 Terms and definitions 75
4.5.5 Overview 75
4.5.6 Basic concepts 76
4.5.7 Main interactions 76
4.5.8 Architectural drivers 81
4.5.9 Test plan 83
4.5.10 Re-utilised Technologies/Specifications 84
4.6 Zorp Firewall: Open specifications 85
4.6.1 Preface 85
4.6.2 Copyright 85
4.6.3 Legal notice 85
4.6.4 Terms and definitions 85
4.6.5 Overview 85
4.6.6 Basic concepts 86
4.6.7 Main interactions 86
4.6.8 Architectural drivers 89
4.6.9 Test plan 90

4.6.10 Re-utilised Technologies/Specifications 92
4.7 Zorp SSL: Open specifications 92
4.7.1 Preface 92
4.7.2 Copyright 92
4.7.3 Legal notice 93
4.7.4 Terms and definitions 93
4.7.5 Overview 93
4.7.6 Basic concepts 94
4.7.7 Main interactions 94
4.7.8 Architectural drivers 96
4.7.9 Test plan 97
4.7.10 Re-utilised Technologies/Specifications 99
5 Updated use case partner security requirements 100
5.1 Instrumentacion y Componentes S.A. (Inycom) Security Requirements 100
5.2 SAKER Security Requirements 101
5.3 Outlandish Security Requirements 101
6 Summary and Conclusions 103
7 References 105

1 Introduction

This document focuses on identifying security vulnerabilities and requirements, as well as describing corresponding counter measures to mitigate such attacks. To achieve that, we follow two approaches. First, we analyze the current MiCADO core architecture and secondly, we collect and analyze the requirements that were defined by the use case partners.

The main objectives of this document are the following:

- Identify possible vulnerabilities of the current MiCADO core architecture;
- Describe possible attack vectors on MiCADO;
- Present a concrete list of counter measures against the specified attacks;
- Analyze the security requirements that were defined by the use case partner;
- Illustrate security enablers that can be used to provide counter measures to possible attacks.

The security analysis is performed based on the current MiCADO core architecture enhanced with the specific components that were described in D6.2. Apart from that, to enhance the overall security of the infrastructure, counter measures may be implemented based on security components that were presented in D7.2. Therefore, this deliverable must be read in conjunction with D6.2 and D7.2:

1. D6.2 – "***Prototype and documentation of the monitoring service***" – contains the detailed specification for core components of MiCADO architecture.
2. D7.2 – "***MiCADO security architecture specification***" – presents the security architecture with a detailed description of all the security components.

The COLA Security Architecture will be used as input for D7.4 "***Security policy formats specification***", as well as subsequent deliverables in WP7.

The remaining of this deliverable is structured as follows:

- Chapter 2 – Core components of MiCADO and data security requirements

This chapter illustrates the core components of MiCADO architecture accompanied with the relevant security requirements;

- Chapter 3 – Threat models and attack vectors

This chapter elaborates on the security of the infrastructure. More precisely, the infrastructure threat surfaces are defined as well as a concrete list of threat models that may be used to attack such systems. Furthermore, a list of possible attacks and their counter measures are described;

- Chapter 4 - Security enablers open specifications

This chapter illustrates specifications for security enablers/ components which are described in D7.2. These security enablers may be implemented to enhance protection for the infrastructure against the attacks presented in Chapter 3;

- Chapter 5 - Use case partners security requirements

This chapter describes a concrete list of security requirements that were identified by the use case partners based on the specific needs of their systems;

- Section 6 – Summary and conclusion

This chapter concludes this deliverable.

2 Core components of MiCADO and data security requirements

In this section, we briefly describe the core components of MiCADO architecture. Furthermore, we elaborate on the importance of protecting data operated inside the system. For a more detailed description on MiCADO architecture, we refer to deliverable D6.2 [1].

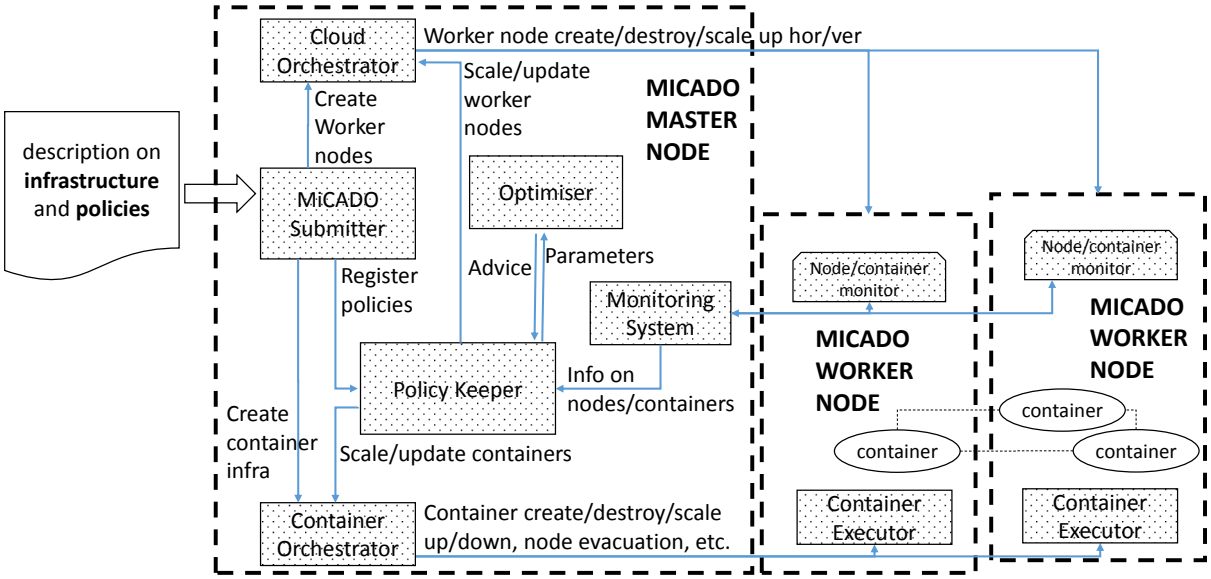


Figure 1 MiCADO Architecture [1]

MiCADO consists of one master node and several worker nodes. The master node can be deployed either locally or in the cloud while the worker nodes are created in the cloud and can be used by the users to run experiments. The master node currently contains five main components with different roles: MiCADO submitter, Cloud Orchestrator, Container Orchestrator, Policy Keeper and Monitoring system. The Optimiser component is an extension later.

2.1 MiCADO submitter

MiCADO submitter is an entry point where users[1] can input a TOSCA file describing the application topology and the relevant policies into MiCADO. The topology illustrates all the components of the application as well as their Docker images along with their relationship. In addition to that, the virtual machine configuration for worker nodes on which Docker images will be deployed is described. Meanwhile, policies are the set of rules which are used throughout the lifecycle of the application, such as scaling policies and security policies. For more information about the generated TOSCA file, please refer to deliverable D5.4 [2]. Although it is *not* necessary to keep confidentiality for information such as the configuration of a virtual machine or the public Docker images, such information still needs to be protected by making sure that it will not be tampered in transit. For instance, an adversary can try to change the user's Docker image into their own Docker image, such as a coin miner, and/or upgrade virtual machine configuration to take advantage of the existing cloud resources for their own benefit by avoid paying any cost. In addition, it is quite common that user needs to manage sensitive information which their applications need to use during the runtime (e.g.

[1] In the scope of this section, users mean entities who can deploy applications into MiCADO.

usernames and passwords, name of database, etc.) but they do not store inside Docker image files. Consequently, ***protecting TOSCA file's confidentiality in transit*** from the user to MiCADO is of paramount importance.

2.2 Cloud Orchestrator and Container Orchestrator

The two main components for scaling are the Cloud Orchestrator and the Container Orchestrator. Cloud Orchestrator, aims to scale up or down virtual machines (VM) while the Container Orchestrator does the same for Docker containers.

To deploy new VMs or delete unused VMs, Cloud Orchestrator sends requests to the Cloud Service Provider (CSP) where the user has been registered with. The user needs to expose their CSP account to Cloud Orchestrator inside MiCADO so that CSP accept requests from the Cloud Orchestrator. Information such as CSP user account is considered as sensitive and it must be kept private and protected from any potential unauthorized access. Cloud Orchestrator uses user account to prove its identity to the CSP and sends user's *VM configuration* for worker nodes demanded to the CSP. Then, the CSP launches a new VM based on the configuration required by the user.

Currently, the Cloud Orchestrator component is deployed using Occopus while the Container Orchestrator is implemented by using Docker Swarm on swarm mode [2]. Within the swarm mode, there are two roles for VM hosting Docker containers: *Swarm Manager* and *Swarm Workers*. A concrete set of such VMs forms a cluster which is called *swarm*. In MiCADO, the Master Node plays the role of Swarm Manager and the Worker Nodes act as Swarm Workers. As soon as the Cloud Orchestrator (i.e. Occopus), launches a new VM, that new VM uses the *swarm worker token* – a secret which is generated by the Swarm Manager and allows a VM to join an existing swarm. Therefore, the swarm worker token needs to be sent from the Master Node to the Worker Node and be protected in transit. Meanwhile, *swarm manager token*, that can be used by any machine to make itself become a Swarm Manager, must be kept confidentially inside the Swarm Manager.

In addition to that, in certain applications which are structured as a set of different components (i.e. different Docker containers), communication among them could be required. Container communication can be classified in three types: *(1)* ***Internal,*** which means communication between different containers inside a VM, *(2)* ***Across-VMs,*** which means communication between two containers that are running in different VMs, and *(3)* ***External,*** which means communication between a container and an external entity such as an external database. The data that are exchanged between containers in Across-VMs and External communication should be protected in transit.

2.3 Policy Keeper

The core component which is responsible for the auto-scaling feature of MiCADO is the Policy Keeper. Scaling policies are defined by users in TOSCA files that are injected into MiCADO through the MiCADO Submitter. After that, they are extracted and parsed from the TOSCA files and are sent to the Policy Keeper.

Policy Keeper receives monitoring information on worker nodes and containers from the Monitoring System and makes decisions regarding scaling up or down VMs and/or containers based on the defined scaling policies. Currently, this component is implemented using the Prometheus Executor.

2.4 Monitoring System

The consumption of resources while running an application in MiCACO infrastructure is collected by various entities in the Worker Node. This information is then transmitted to the Master Node. The Monitoring System in Master Node is implemented by Prometheus which actively requests data from the existing monitoring agents in WNs. Meanwhile, in Worker Node, Consul is a node discovery agent responsible for sending machine *health check information* to the Master Node. In addition to that, the Node Exporter collects monitoring data from virtual machines such as *CPU, diskstats,* etc. while Cadvisor *monitors microservices and Docker containers*. Although such *monitoring information* is not confidential, it should still be protected to prevent possible side channel attacks.

2.5 External entities

In addition to core components of MiCADO, there are a few external entities that have or might have connections with MiCADO.

Users: In this document, when we refer to users we refer to any entity that can deploy applications in MiCADO. This entity should be authenticated to MiCADO before being able to start the deployment.

Cloud Service Provider (CSP): This entity is selected by users and provides infrastructure as a service for users' application deployment. *It is assumed that one MiCADO infrastructure is deployed using resources from only one CSP*.

External repositories: Some application can store its data files, configuration or the databases in external repositories located outside of MiCADO. In such case, the application itself is deployed in MiCADO and it connects to external repositories during runtime.

Administrator: This entity is responsible for launching the MiCADO infrastructure.

Application users: This entity uses the application and is not related to MiCADO deployment. Therefore, we will not provide any analysis for the application users.

2.6 Summary

Based on the above description of MiCADO's core components and external entities, in Table 4 we summarize the data that is transmitted between MiCADO and any involved external entity, as well as between nodes (VMs) in MiCADO. Additionally, Table 4 also presents a description of all possible vulnerabilities that we found during this analysis. Furthermore, we classify data into the following three protection levels:

- **Level 1:** Information is considered public;
- **Level 2:** Information is not public, but disclosing this information would not cause any harm. However, tampering this information could cause risks;
- **Level 3:** Information is sensitive.

#	Communication direction	Message content	Vulnerabilities	Protection level
1	Administrator → CSP	**Init configuration file** to launch the MiCADO infrastructure	1. Tampering the file, for e.g. virtual machine identity, number of maximum worker nodes, installation command to install new services into the master nod 2. Accessing confidential information, i.e. cloud user password	3
2	User → MiCADO	**TOSCA file** to describe user's application	1. Tampering the file, for e.g. repository of application Docker images or VM configuration for worker nodes 2. Accessing confidential information, i.e. application's sensitive information if described	2 or 3 (2 if TOSCA file does not contain any sensitive information; 3 otherwise)
3	User → MiCADO	**MiCADO login credential**	1. Accessing login credential for impersonation attacks	3
4	MiCADO → User	New **template for security policies** section in TOSCA file	1. Tampering the file, for e.g. deleting some security policies	1
5	Master node → Worker node	Swarm worker token[2] Scaling containers request	1. Accessing the swarm worker token 2. Changing the request	3
6	Worker node → Master node	Monitoring information including heath check, cpu, diskstats, microservices and containers info, etc.	Changing the monitoring information	2
7	Worker node → External repository, if needed	Experimental data and results	Accessing data and results	3
8	Worker node ↔ Worker node	Application data	Accessing application data	3
9	MiCADO → CSP	Scaling VMs request	1. Changing VMs scaling request 2. Impersonating MiCADO to send requests to the CSP	3

Table 4 Communication Vulnerabilities of MiCADO

[2] Example:
- wget --retry-connrefused -qO /tmp/swarm_join {{variables.master_host_ip}}:2375/v1.26/swarm
- export TOKEN=$(grep -Eo 'SWMTKN-[[:alnum:]]*-[[:alnum:]]*-[[:alnum:]]*' /tmp/swarm_join | head -1)

From Table 4 it can be observed that it is essential to protect communications of MiCADO. Currently, not all communications in MiCADO are protected. More precisely, only the communication between container nodes in a swarm are secured by using Transport Security Layer (TLS) [4]. Other components, such as Prometheus, do not support any form of secure communication [17]. Meanwhile, security of communication between Occopus and CSP depends on the support provided by the CSP. The lack of security mechanisms for protecting the communication between MiCADO and any available external entity and/or between VM nodes in MiCADO makes the system vulnerable against common attacks like *eavesdropping, data modification and man-in-the-middle* [9].

In addition, access control to MiCADO as well as to the CSP are password-based. As a result, the system is susceptible to *password-based attacks* where an attacker that has access to a valid account can gain complete control of the system.

- `docker swarm join --token $TOKEN {{variables.master_host_ip}}:2377` in `cloud_init_worker.yaml` file

Example of swarm worker node token: SWMTKN-1-49nj1cmql0jkz5s954yi3oex3nedyz0fb0xx14ie39trti4wxv-8vxv8rssmk743ojnwacrr2e7c

3 Threat models and Attack vectors

In the previous section, we described the internal components of MiCADO and the external entities that will participate in our scenarios. Furthermore, we elaborated on the exchanged data inside MiCADO as well as between MiCADO and external entities, then we classified them into three protection levels. Continuing in the same direction, in this section, we identify threat surfaces of MiCADO which can attract several attacks. Based on the identified threat surfaces we describe a list of adversarial models that we need to consider. This concrete list of malicious behaviours, allowed us to describe several possible attack vectors and propose countermeasures by designing new protocols that will enhance the overall security of MiCADO.

3.1 Threat surface

To highlight MiCADO's main threat surfaces, we describe two basic scenarios that link MiCADO with the described external entities.

Scenario 1: Launch MiCADO infrastructure.

1. Administrator launches MiCADO Master Node in local host or in the cloud using an *init configuration file*. This file contains services and components installation for Master Node and cloud user credentials which will be later used by the Cloud Orchestrator. If MN is deployed in the cloud, the file also contains the underlying machine configuration.
2. Master Node might launch the default minimum number of Worker Nodes.

Scenario 2: Deploy an application in the launched MiCADO infrastructure.

1. User composes a TOSCA file that describes both the application and the WN configuration.
2. User sends the generated TOSCA file to MiCADO.
3. MiCADO deploys the user defined application in the Worker Node, autoscale the number of VMs and containers based on user defined scaling policies.
4. In case the user uses an external storage/database for application data/results, application containers in WNs connect to the external entity to send/request data.

Between the two scenarios, the second one occurs more often. Therefore, this is the scenario we consider to define the threat surfaces of MiCADO. In addition to that, we also assume that the Master Node is deployed in cloud.

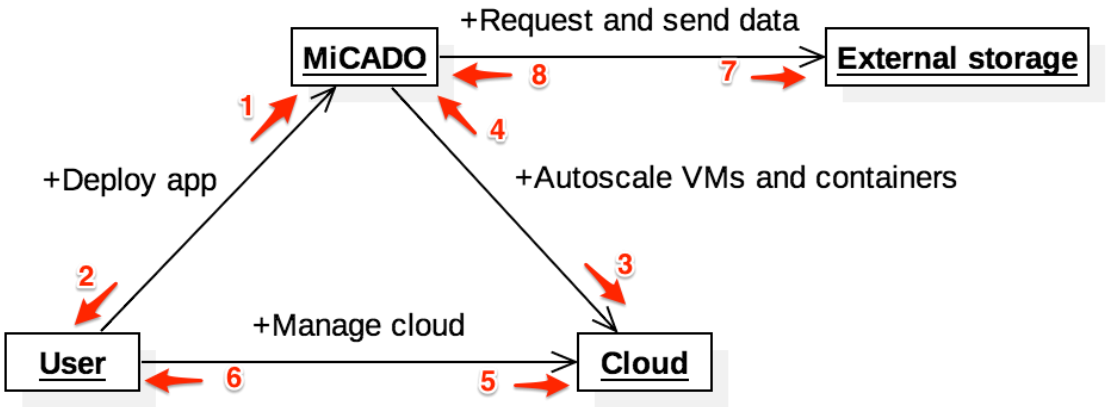


Figure 2 Interaction of the participating entities

Surface 1: [User →] MiCADO
Possible attacks towards this surface include buffer overflow [7].
Surface 2: [MiCADO →] User
Possible attacks towards this surface include SSL spoofing and attacks on the browsers cache [7].
Surface 3: [MiCADO →] Cloud
Possible attacks towards this surface include resource exhaustion and denial-of-service [7], [10-12].
Surface 4: [Cloud →] MiCADO
Possible attacks towards this surface include privacy breaches [13-17] and data tampering [7].
Surface 5: [User →] Cloud
Possible attacks towards this surface include impersonation.
Surface 6: [Cloud →] User
Possible attacks towards this surface include triggering unnecessary usage of cloud services and wrong bill delivering [7].
Surface 7: [MiCADO →] External storage
Possible attacks towards this surface include impersonation and data breach [18].
Surface 8: [External storage →] MiCADO
Possible attacks towards this surface include data tampering.

Among the aforementioned attack surfaces, we will be concentrating on surfaces 1, 2, 3, 7, 8. Surfaces 5 and 6 are out of scope because they involve a direct connection between the users and the cloud without any interaction with MiCADO. Furthermore, we skip surface 4 because we have assumed a trusted CSP.

3.2 Identified threat models

Based on the attack surfaces described above and transmitted data described in Chapter 2, we define the following threat models:

Threat Model 1: ***User impersonation***
There are several vulnerabilities that allow a malicious adversary to successfully perform an impersonation attack. The following are considered as the most common ones:

- Provide no authentication or access control;
- User chooses a weak password;
- Provide no protection for the communication channel between users and MiCADO.

Countermeasures:

- Provide strong authentication;
- Force user to choose a strong password;
- Protect communication channels through SSL/TLS.

Related Threat Surfaces: 1, 2

Threat Model 2: ***TOSCA file Modification***
TOSCA file is sent by users to MiCADO's Master Node. If the file is not properly protected, a malicious adversary can access or modify the information contained in the file.

Possible modification to the file includes the following:

- Change VM configuration for WN from a low-cost to a high-cost, or increase disk allocation of the underlying VM;

- Add malicious applications/services into the WN;
- Open certain ports in WN;
- Disable security features;
- Change the configuration of the application.

Countermeasures:
Communication between the user and MiCADO should be done over SSL/TLS. Additionally, a firewall must be used to prevent an attacker from opening certain ports.

Related Threat Surfaces: 1, 2

Threat Model 3: *Data Breach*
Sensitive information that can be exposed consist of the following:

- Cloud user credentials that are needed for the Cloud Orchestrator to send requests to the CSP;
- Sensitive information that may be accessed by an application during runtime (e.g. database user account and API token);
- Swarm manager token;
- Swarm worker token;
- Application experimental data and results;
- User's MiCADO credential.

Countermeasures:
Sensitive information could be stored in a central component which is called *Credential Store*. This information does not include the Swarm manager token and the worker tokens which are managed by the Swarm itself.

- Cloud user credentials are sent over TLS/SSL and stored in an encrypted form in the Credential Store;
- Application's sensitive information is stored in Docker Swarm or Credential Store instead of the Docker image or the source code of the underlying application;
- Swarm manager token is protected by the Swarm;
- Swarm worker token is sent over TLS/SSL;
- Application experimental data and results are sent over TLS/SSL;
- User login credentials are sent over TLS/SSL and managed by the Credential Manager.

Related Threat Surface: 1, 2, 3, 7, 8

Threat Model 4: *Open ports exploitation*
Attackers can execute a port scanning to identify open ports in both the MN and WNs. Then they can take advantage of the identified open ports and try to inject malicious code.

Countermeasures:
Only necessary ports should be open for public access. Other ports should be closed to prevent possible vulnerabilities. In addition to that, all communication towards MN and WNs should be protected.

Related Threat Surface: 1, 4, 8

Threat Model 5: *Virtual Container Alternation*
This includes:

- Change VM image which does not comply with the predefined security policies. This can be done by administrators with specific access rights;
- Modify, remove or replace the container image which may contain malicious software.

Countermeasures:
Integrity verification could be done frequently. This includes the following functions:

- VM image integrity verification;
- Container image integrity verification.

Related Threat Surfaces: 1, 4

Threat Model 6: *Cloud-init Config file Modification*
Modification could be the following:

- Add malicious Docker containers/services;
- Open ports in Master Node.

Countermeasures:
Communication between the administrator and the CSP could be done over SSL/TLS. However, this depends on the functionality offered by the underlying CSP.

Related Threat Surface: This attack is related to threat surface from administrator to CSP as described in the first scenario.

Combining Threat Models: An attacker can combine all the above attacking techniques in order to perform more sophisticated and possible powerful attacks.

3.3 Attack vectors

We assume that the CSP is running in a trusted state and it is SSL/TLS-enabled. Furthermore, we assume that the Docker image registry that is provided by the user is also trusted. As a result, we are not getting into details on Threat model 5 – Virtual Container Alternation and Threat model 6 – Cloud-init Config file Modification. However, for the remaining threat models, we provide a *concrete list of attack vectors* as well as a *set of countermeasures*. The proposed countermeasures are satisfied by designing new protocols that can increase the overall security of MiCADO.

3.3.1 Threat model 1: User impersonation

There are several cases that can lead to impersonation attacks. In the following paragraphs, we describe the most common ones.

Man-in-the-middle attack: Man-in-the-middle attack is a very common attack against http communication. The attacker acts as a proxy and intercepts the communication between the user and the server to read or even modify data. **Error! Reference source not found.** llustrates such an attack in which the attacker tries to access user's account.

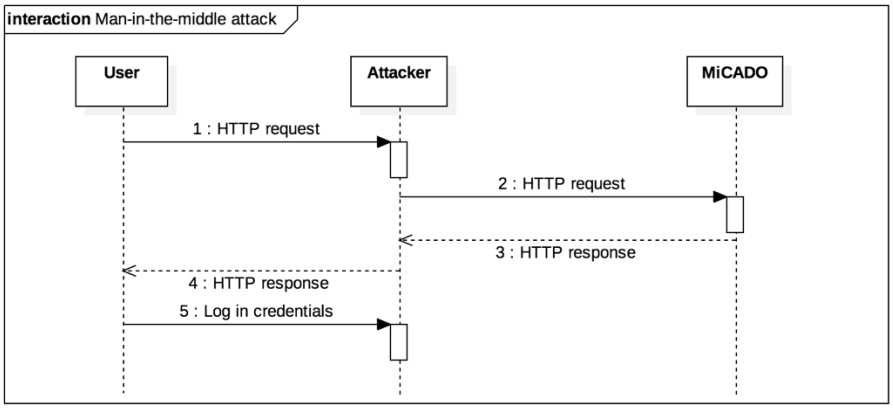


Figure 3 Man-in-the-middle attack

1. A user sends a request to MiCADO infrastructure;
2. An attacker forwards it to MiCADO;
3. Upon receiving the request, MiCADO sends its response with a form asking for user name and password;
4. The attacker forwards the response to the user;
5. The user enters user name and password to send back MiCADO;
6. The attacker overhears the communication between the user and MiCADO and gets access to the credential sent by the user. From now on, the attacker can impersonate user.

Countermeasure:

In order to prevent such man-in-the-middle attack, MiCADO shall support HTTPS communication instead of HTTP.

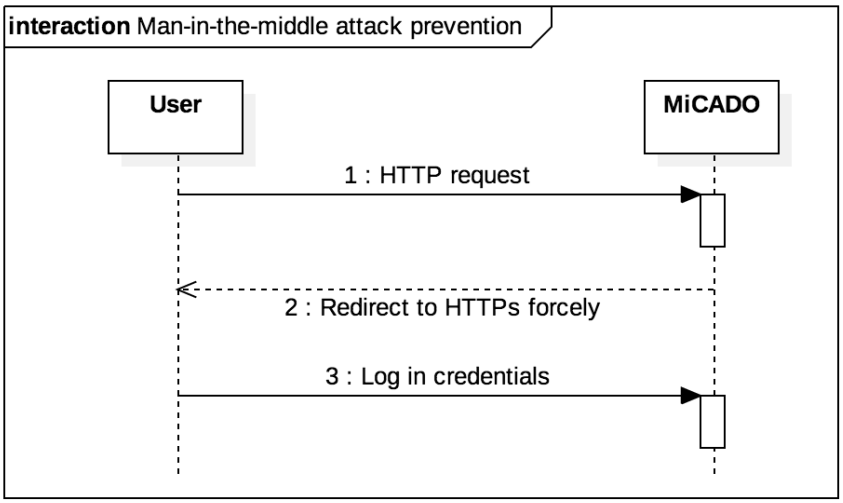


Figure 4 Man-in-the-middle attack countermeasure

1. A user sends a request to MiCADO infrastructure;
2. MiCADO redirects the user to HTTPS page for entering user name and password.

Brute Force attack or password guessing attack: Many systems rely on password-based authentication. The main reason for this is due to the ease of use as well as for providing users with a user-friendly authentication system. MiCADO would also be based on such authentication protocol which requires users to input their user name and password to log into the system. However, security of accounts is always a big concern due to possible vulnerabilities caused by developers' implementation and users' weak passwords [19]. Among the most popular password attacks, password guessing is considered as the most common. In such an attack, the attacker tries to guess users' password manually or automatically. This attack can be performed either offline or online [20]. Figure 5 demonstrates a very basic online guessing attack.

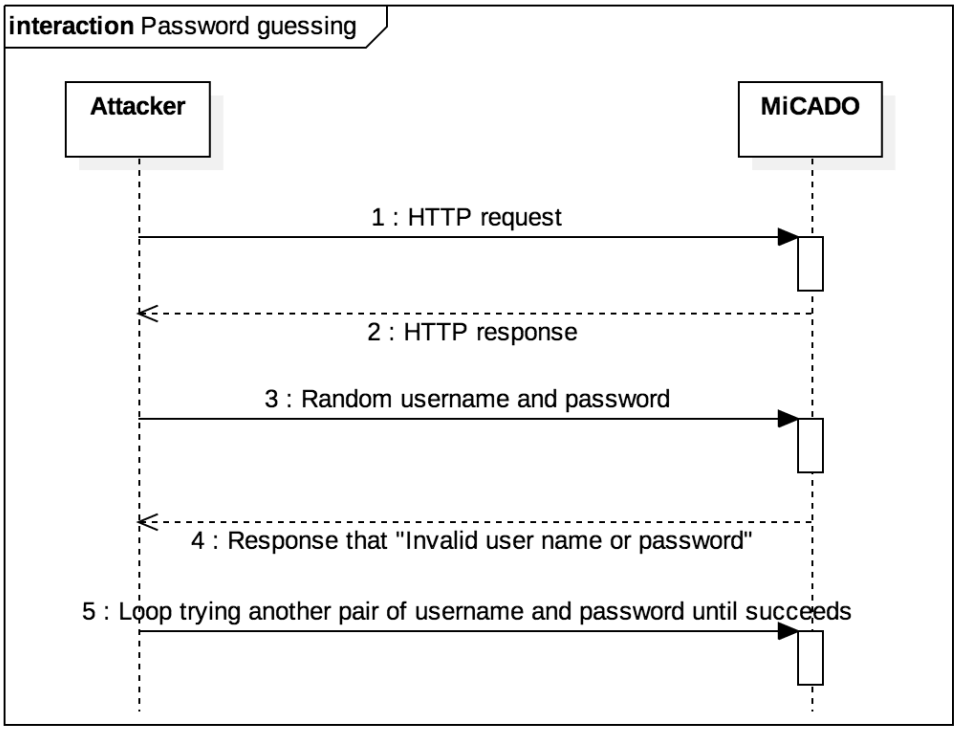


Figure 5 Password guessing attack

1. An attacker sends a request to MiCADO;
2. MiCADO replies by sending back a form that user needs to fill in here username and password;
3. The attacker inputs a random username and password and sends it to MiCADO;
4. MiCADO checks if the received credential match one of the records stored in the database;
5. Assuming that the credential does not match any records in the database, MiCADO sends a response "Invalid user name or password";
6. The attacker tries with another pair of random username and password until he succeeds to log in.

Countermeasure

Multiple countermeasures may be implemented to prevent password guessing attack. Figure 6 describes only a few of them.

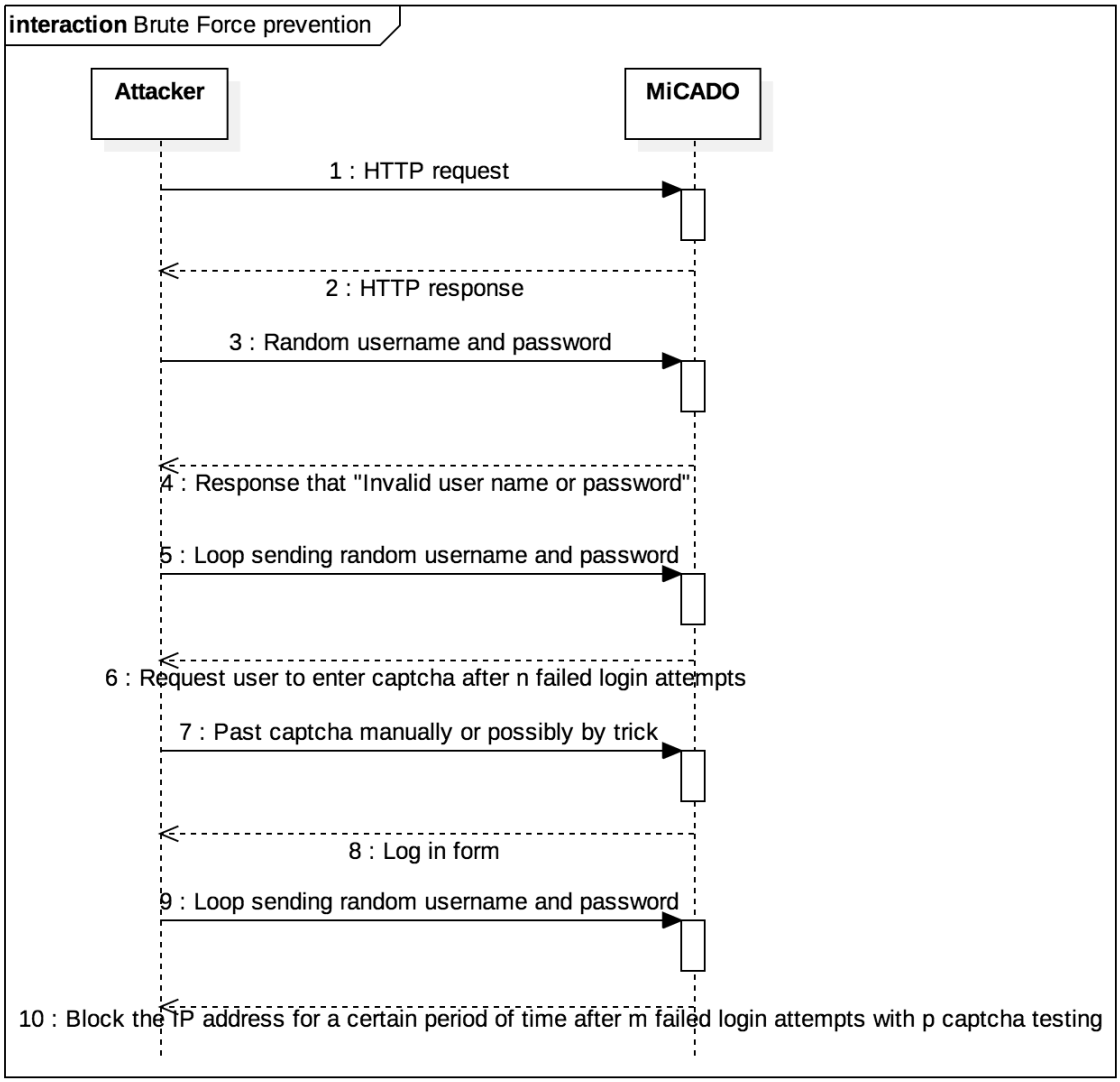


Figure 6 Password guessing countermeasure

1. An attacker sends a request to MiCADO;
2. MiCADO sends back HTTP response with a form for filling username and password;
3. The attacker completes the form by providing a random username and password;
4. MiCADO checks if the received credential matches one of the database records. If not, MiCADO sends a general response: “Invalid username or password”;
5. The attacker continues trying with random usernames and passwords;
6. After a fixed number of failed log-in attempts, MiCADO requests user to solve a captcha challenge;
7. Assuming that the attacker can pass the captcha challenge;
8. MiCADO replied by resending a fresh log-in form;
9. The attacker continues trying with random usernames and passwords;
10. After a fixed number of failed log-in attempts with a fixed number of captcha testing, the IP-address of the attacker will be blacklisted for a certain period of time (e.g. *h* hours). Therefore, for the next *h* hours the attacker will not be able to access the website

again (unless she changes IP). In addition, in case the attacker failed to log in using the same existed username, the corresponding account will be also blocked for a certain period.

As mentioned earlier, other countermeasures may be implemented to achieve further protection on MiCADO against password guessing attacks. Such countermeasures include the following:

- Setting constraints on password selection, covering both passwords chosen by users and default passwords generated by systems. For instance, following standards defined by NIST [21], secrets shall be at least 8 characters long if chosen by users, and at least 6 characters long if chosen randomly by the system;
- Comparing the prospective passwords against a list of possibly weak ones such as dictionary words, previously breached passwords which can be found from the internet, etc.;
- Using approved encryption and a protected channel to transmit account information;
- Using a suitable key derivation function such as Password-based Key Derivation Function 2 (PBKDF2) [5], that is based on Hash-based Message Authentication Code (HMAC) [3], to add salt and hash passwords;
- Storing passwords in a salted, hashed form;
- Provide two-factor authentication.

Details for such normative instructions can be found in Digital Identity Guidelines of NIST [21].

Password reset man-in-the-middle attack (PRMiTM) [8]**:** While choosing strong passwords increases security, it has one side effect. It has been observed that users tend to forget complicated passwords. Hence, password reset is an essential function that systems shall provide. A few common ways to facilitate password reset includes:

- Security questions;
- Code that is sent to users' mobile phones;
- Code that is sent to users' email;
- Reset link that is sent to users' email.

Figure 7 demonstrates an attack on the password reset protocol that is based on security questions.

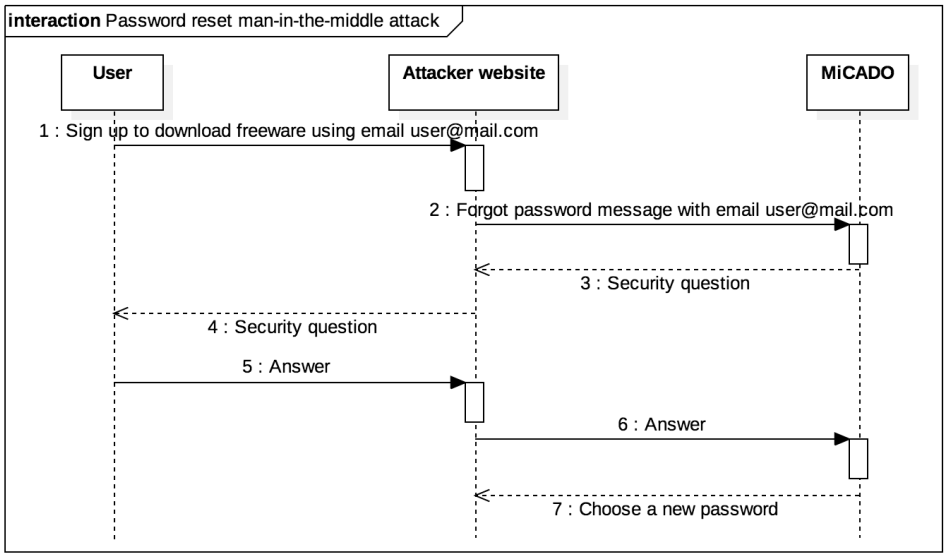


Figure 7 Password reset MiTM attack

Assuming that MiCADO supports recovering password provided that user can answer security questions.

1. A MiCADO user signs up on an attacker's website for downloading a freeware, using email;
2. An attacker uses the user's registered email, i.e. to log into MiCADO and selects the "Forgot password" function;
3. MiCADO displays the user's security questions and wait for answers;
4. The attacker forwards the same security questions to the user and asks her to provide the relevant answers;
5. The user may think that it is required to download a free software. Hence, the user fills in all answers;
6. The attacker uses the user's answers to fill in MiCADO page;
7. Upon checking the given answers, MiCADO allows the attacker to choose a new password.

Countermeasure:

There is a variant of countermeasures for protecting against such attacks. Figure 8 describes one among them.

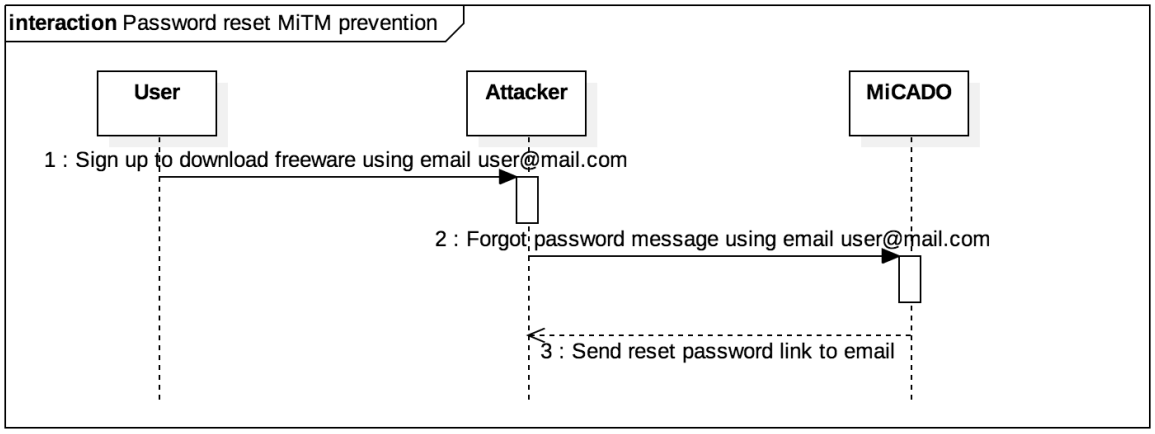


Figure 8 Password reset MiTM attack countermeasure

1. A user signs up on an attacker's website for downloading a freeware, using email her e-mail;
2. The attacker uses the user's registered email to log into MiCADO and selects the "Forgot password" function.
3. MiCADO sends the reset password link to the user's email . Without access to the user's email, the attacker cannot get the reset link. In addition to that, the user can be notified that someone is trying to reset her password.

Other countermeasures as described in the work of Nethanel G. et al. [8], are:

- Including necessary information such as sending website, explanation in password-reset message;
- Notifying users about password reset request;
- Limiting valid time for password reset code or link;
- Avoid relying on security questions.

Certificate spoofing man-in-the-middle (MiTM) attack: A common way to encounter man-in-the-middle attacks is relying on TLS/SSL communication. However, even applying TLS/SSL, it does not always guarantee secure communication. More precisely, attackers can exploit carelessness of standard users or bad habits of experienced users to deploy attacks such as certificate spoofing. Users usually do not check if https protocol is enabled when they access online services. Even experienced users who may care about certificates, they do not always check to make sure that certificates are signed by trusted entities or just self-signed. Such lack of security awareness, allows an attacker to successfully launch spoofing attacks. Figure 9 illustrates a basic certificate spoofing attack.

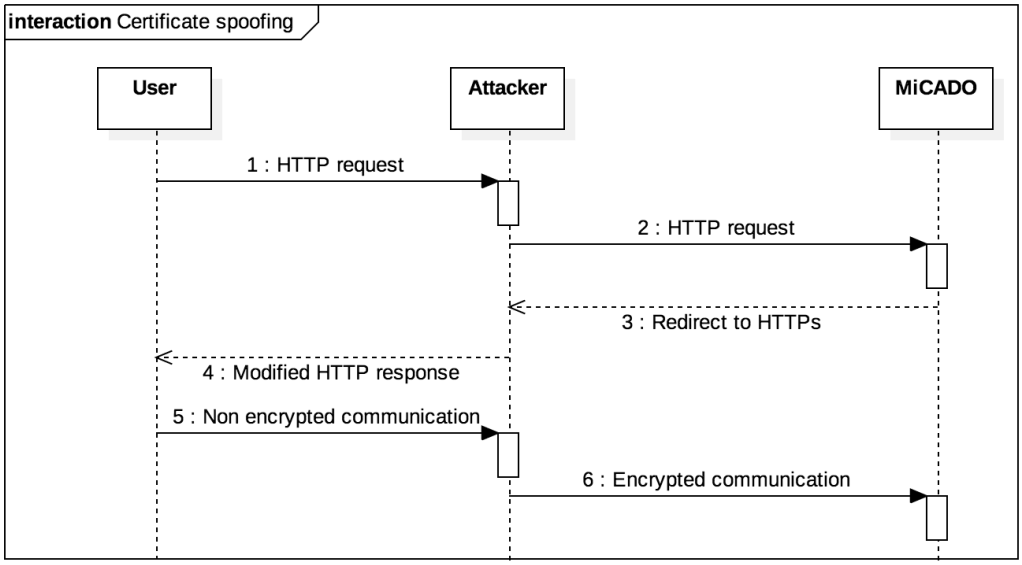


Figure 9 Certificate spoofing attack

Assuming that MiCADO supports both HTTP and HTTPS protocols. Whenever users prompt to http domain name, MiCADO redirects the user to https page.

1. A user sends HTTP request to MiCADO;
2. An attacker catches the request, and forwards it to MiCADO;
3. MiCADO redirects user to the corresponding HTTPS page;
4. The attacker drops the response from MiCADO and sends a modified HTTP response to the user;
5. The user, by trusting that the response comes from MiCADO, enters username and password and sends them back;
6. The attacker catches the sent information and uses it to communicate with MiCADO without the user's recognition.

For details about certificate spoofing attacks, we refer the reader to [22] and [23].

Countermeasure:

HSTS protocol, i.e. HTTP Strict Transport Security [24], defines a new http header containing a web-server-defined policy for users' browsers about how to handle future connections. For instance, the policy may include:

- Indicating if HSTS policy applies to all subdomains or only the main domain;
- Preventing users from accepting self-signed certificates;
- Indicating that the user should stay in https link even when an http link is clicked.

Figure 10 describes how we may apply HSTS protocol.

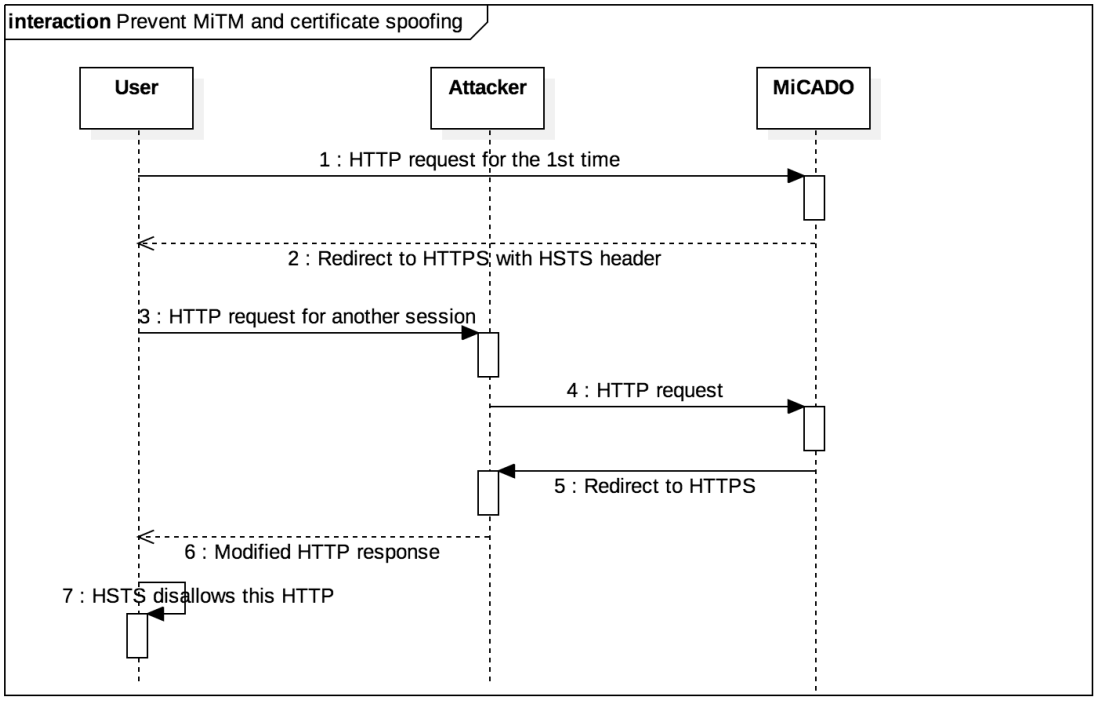


Figure 10 Certificate spoofing countermeasure

Assuming that the first time a user visits MiCADO, there's no attack yet.

1. A user sends a request to MiCADO for the first time;
2. MiCADO redirects the user to HTTPS page with HSTS (HTTP Strict Transport Security) header;
3. Later, the user re-visits MiCADO;
4. MiCADO redirects the user to an HTTPS page;
5. An attacker drops the response from MiCADO, and sends a modified http response to the user;
6. Due to the HSTS header, the user's browser does not allow http response.

Further countermeasures such as secure cookies can be found in OWASP Session Management Cheat Sheet [25].

Resource exhaustion as a result of impersonation attack: MiCADO provides dynamic orchestration for applications. It scales up or down cloud compute nodes to satisfy various needs of the applications. Users can benefit a lot from such functionality. For example, such models can prove cost effective since they tend to save money by utilizing only the needed cloud resources. However, as attackers can impersonate users in MiCADO, they can get control on which applications to be deployed as well as the configuration of the worker nodes. This can escalate user's bill on consuming cloud resources. Figure 11 illustrates such an attack.

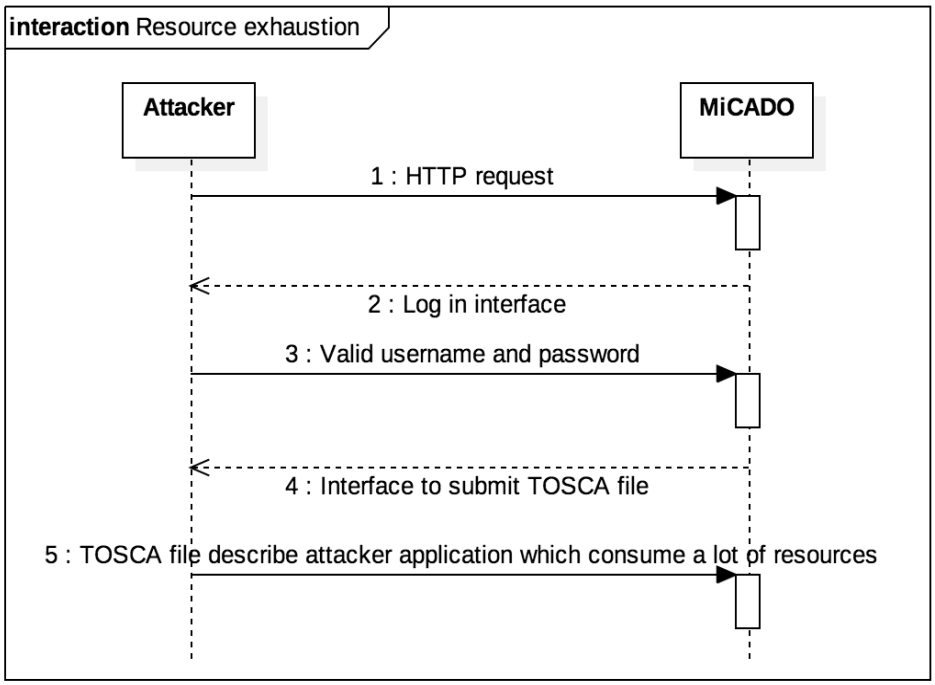


Figure 11 Resource exhaustion as a result of impersonation attack

1. An attacker accesses MiCADO;
2. MiCADO sends back a form to fill in user name and password;
3. Knowing a user's account, the attacker uses it to access MiCADO;
4. The attacker submits a TOSCA file containing their applications and configuration for the worker nodes;
5. The attacker's applications run into MiCADO infrastructures and can use as much cloud resources as possible.

Countermeasure:

Figure 12 illustrates heuristics to detect prospective large-scale consumption of cloud resources and notify users.

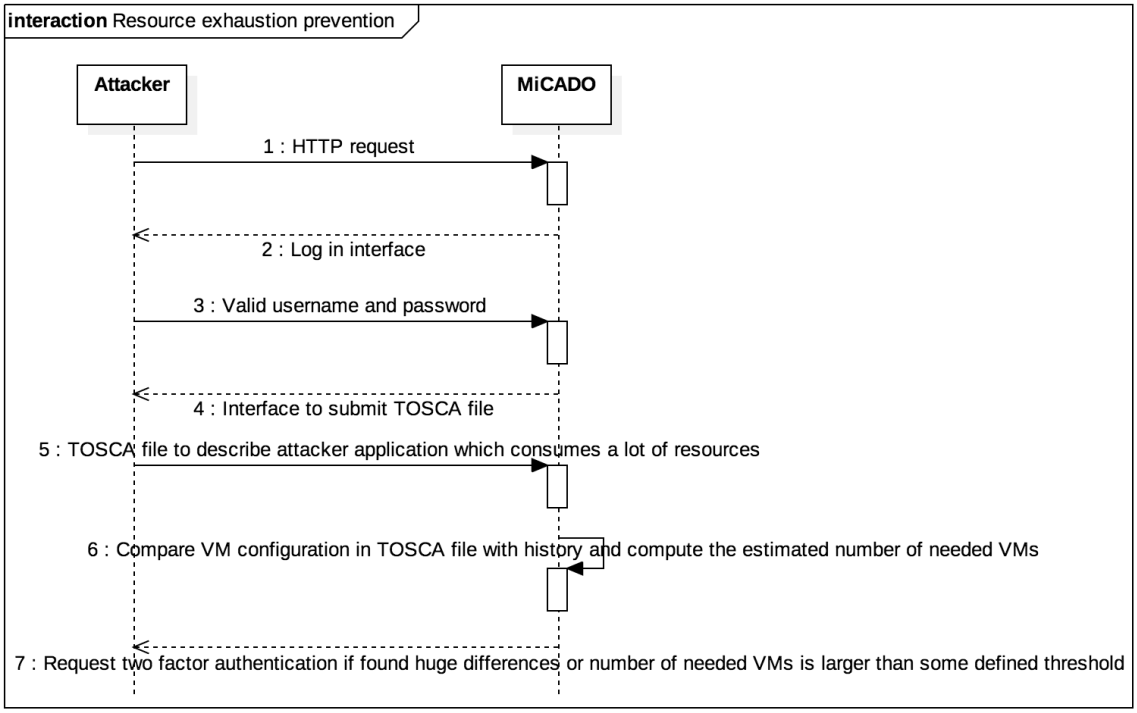


Figure 12 Resource exhaustion countermeasure

1. An attacker sends an access request to MiCADO;
2. MiCADO replies by sending back a form to fill in username and password;
3. Knowing a user's account, the attacker uses it to access MiCADO;
4. The attacker submits their TOSCA file containing their applications and configuration for worker nodes;
5. The attacker's applications would run into MiCADO's underlying infrastructures and it would use as much cloud resources as possible;
6. However, prior to deploying the applications, MiCADO compares VM configuration with historical data, which is a list of used VM configurations previously used by users, and computes the estimated number of needed VMs.
7. If the current VM has a more powerful configuration (memory, cpu, disk size) than the ones used in the past, or the estimated number of needed VMs (if computable) is larger than some defined threshold, MiCADO requests two factor authentication and/or sends a notification to the user's email.

3.3.2 Threat model 2: TOSCA file modification

TOSCA file modification attack due to lack of protected communication: TOSCA file is given as input by users as a way to describe the applications they wish to deploy in MiCADO's infrastructure. Apart from that, this TOSCA file also contains the configuration of the virtual machines for the worker nodes that will be launched. If users send such a TOSCA file through unprotected communication, attackers can tamper with the file as shown in Figure 13.

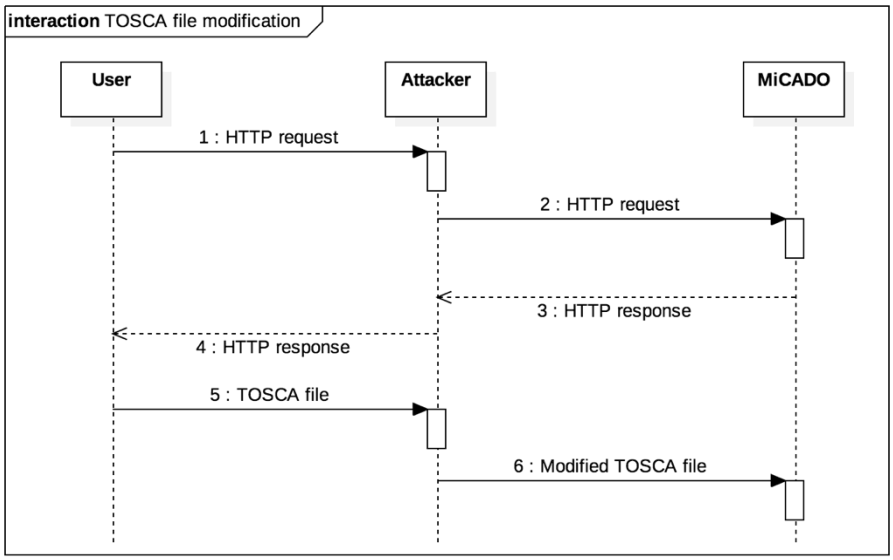


Figure 13 TOSCA file modification attack

1. A user sends an http request to MiCADO;
2. An attacker acts as a proxy and forwards it to MiCADO;
3. MiCADO sends back an http response;
4. The user submits the TOSCA file to MiCADO;
5. The attacker drops the file and sends a fake/malicious file to MiCADO.

Countermeasure:
Protecting communication between users and MiCADO with TLS/SSL channel can prevent such attacks.

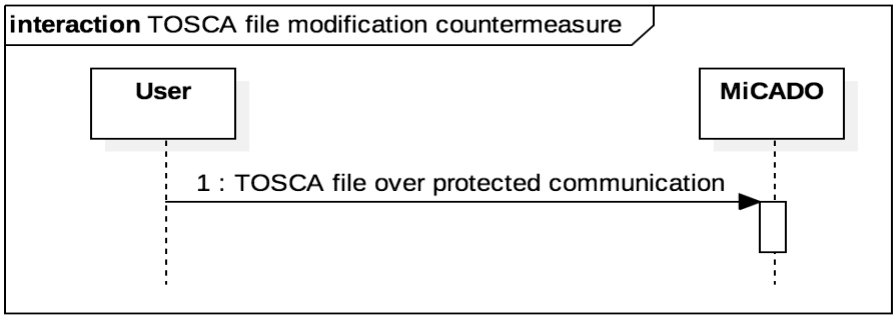


Figure 14 TOSCA file modification attack countermeasure

Resource exhaustion as a result of TOSCA file modification attack: A successful TOSCA file modification attack can lead to resource exhaustion attack in which the attacker leverages cloud resources as much as possible for their own benefit (i.e. to run their own applications). A resource exhaustion attack is shown in Figure 15.

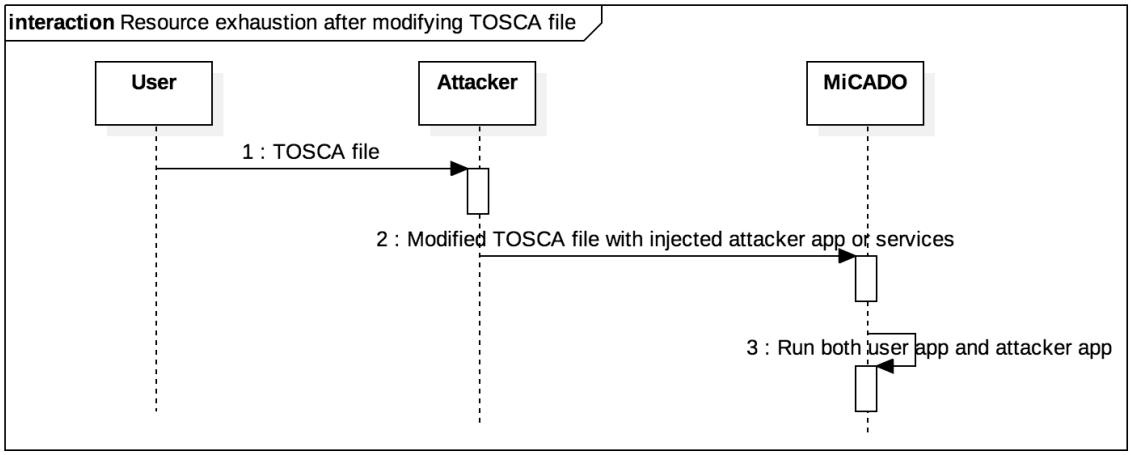


Figure 15 Resource exhaustion due to TOSCA modification attack

1. A user submits a TOSCA file to MiCADO;
2. An attacker drops the file and modifies it by adding description of their own applications or services and/or change machine configuration for worker nodes to more powerful ones;
3. Upon receiving the file, MiCADO runs all applications described in the file including the user and the attacker's applications.

Countermeasure: See Figure 12 Resource exhaustion .

3.3.3 Threat model 3: Data Breach

Application sensitive information breach: It is common that applications require some sensitive information to properly run (e.g. database credential, certificate, API token, etc.). A common way to store such information is either by injecting it into the application's source code or into the Docker image by defining them in the Docker file. However, such ways do not provide any form of protection since it cannot restrict who can access the information. More precisely, anyone who can access the source code (in case of hard-coding the sensitive information) or who can access the image (in the case of injecting sensitive information in the image file) can retrieve it. In addition to that, whenever developers want to perform an update of this sensitive parts, they need to re-build and re-deploy the applications. Hence, apart from weak security, this technique is also considered as inefficient. An application sensitive information breach is shown in Figure 16.

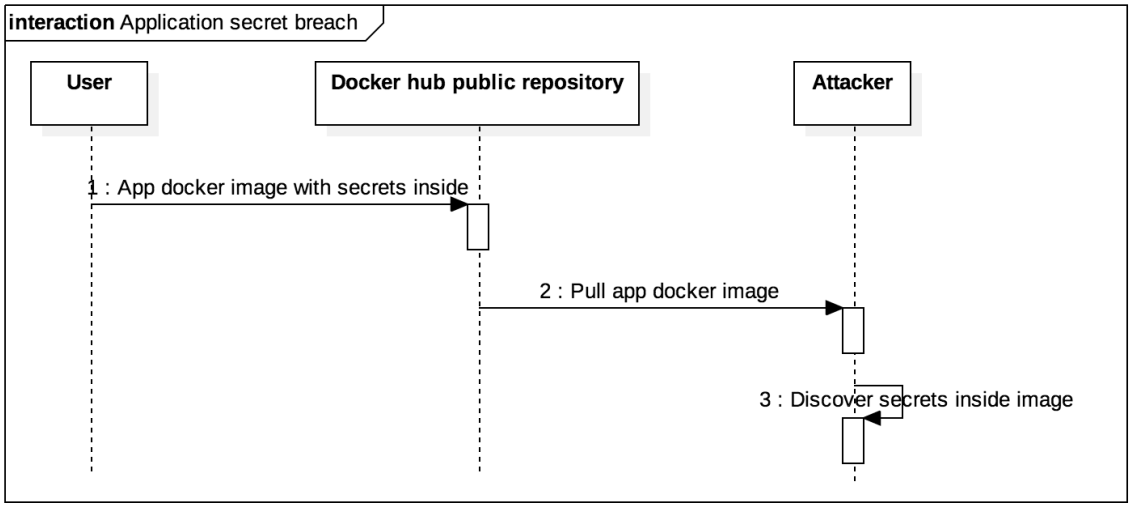


Figure 16 Application sensitive information breach

1. A user (developer) builds a Docker image for an application. This image also contains sensitive information that is required for the proper run of the application.
2. User uploads the image to the Docker hub;
3. An attacker pulls the image from the Docker hub;
4. The attacker can access the sensitive information contained in the image.

Countermeasure:

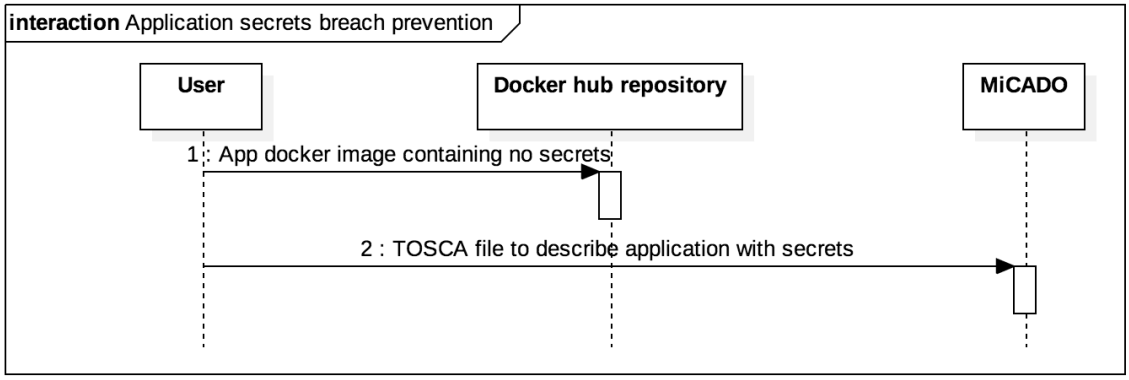


Figure 17 Application sensitive information breach countermeasure

1. A user (developer) builds a Docker image for an application without any sensitive information inside the image or the source code, and uploads it to the Docker hub;
2. The user uploads a TOSCA file, that contains application's sensitive information that is required to run the application inside MiCADO;
3. MiCADO securely stores the received sensitive information inside the infrastructure where application containers can access during run time.

Application data breach: It may happen that not all components of an application would be deployed in MiCADO's infrastructure. An application can invoke APIs from external entities (i.e. entities that stay outside of MiCADO – such as an external storage). It is vital to define whether the application containers inside MiCADO communicate via secure or insecure APIs

with the external entities. In the latter case, an attacker can take advantage of such an unprotected communication and intercept the transmitted data as shown in Figure 18.

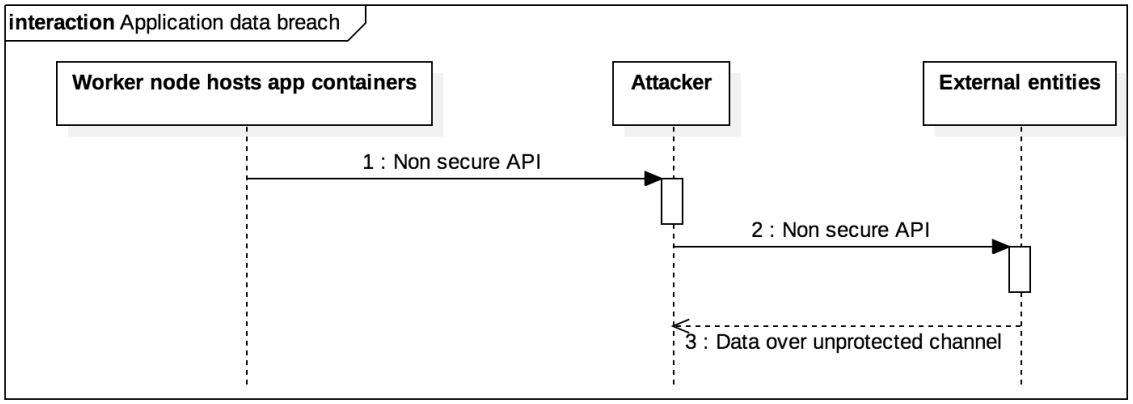


Figure 18 Application data breach

Countermeasure:

This attack depends on application developers, and it is out of MiCADO's control. The application developers shall ensure that their applications do not invoke any unprotected API.

In the other way, if MiCADO provides users with REST API, for instance to submit TOSCA files, it shall ensure that REST services only provide HTTPS endpoints. Other security requirement for REST API can be found in OWASP REST Security Cheat Sheet [26].

3.3.4 Threat model 4: Open ports exploitation

Attackers can take advantage of open ports in MiCADO's infrastructure, including both the master node and/or the worker nodes, to deploy certain attacks.

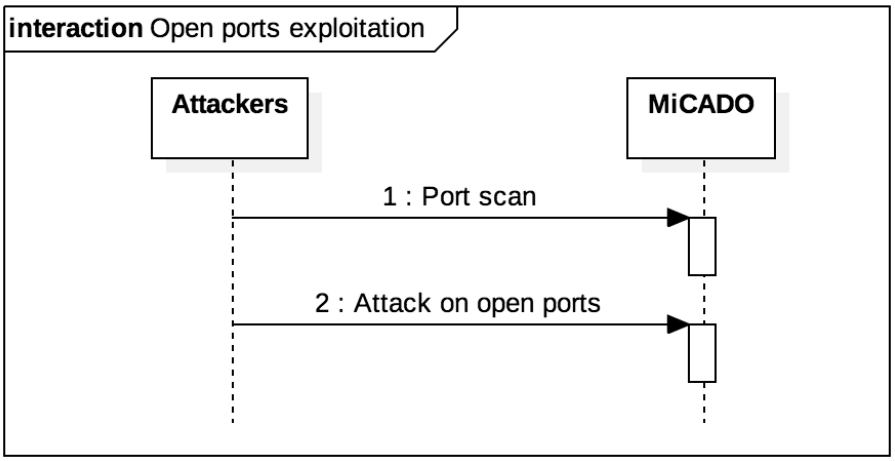


Figure 19 Open ports exploitation

Countermeasure:

To reduce potential vulnerabilities from open ports, it is important to control ports so that only the necessary ones are open. This can be done by utilizing firewalls for both the master node and the worker nodes. Components in the master node are not expected to be changed a lot; therefore, an administrator can decide which ports to open during MiCADO's infrastructure launch. In contrast, components in worker nodes may vary based on the deployed applications. As a result, identifying the ports that are essential in worker nodes may not be decided until the deployment of an application. In that case, the user may define the necessary ports directly in the corresponding TOSCA file.

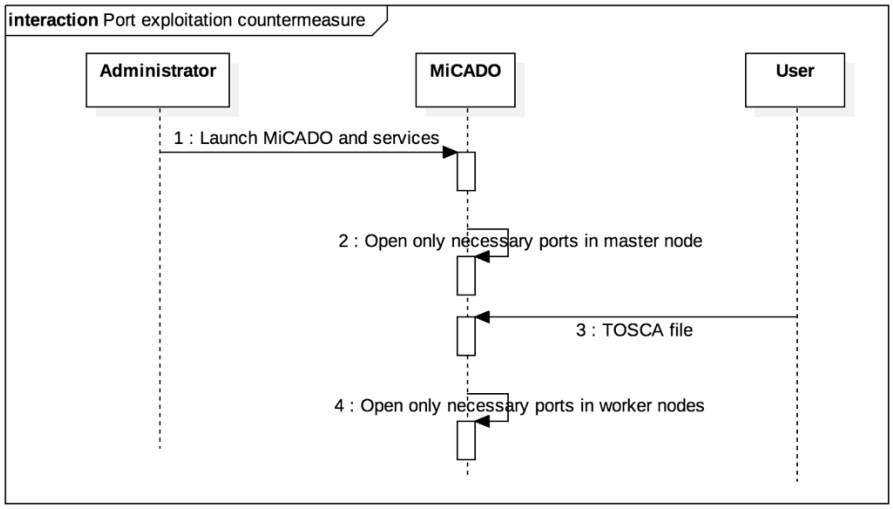


Figure 20 Open ports exploitation countermeasure

1. The administrator launches MiCADO's infrastructure and defines the open ports in the master node;
2. MiCADO opens the specified ports in master node by configuring the underlying firewall;
3. During the deployment phase, user sends a TOSCA file containing a list of open ports for worker nodes that are required for a proper run of the application;
4. MiCADO opens the specified ports in worker nodes by configuring the underlying firewall.

4 Security Enablers Open Specification

4.1 Image Integrity Verifier Open specifications

4.1.1 Preface

Execution environments based on lightweight virtualization – commonly known as containers – are widely used in modern cloud infrastructure deployments. They gained extensive popularity due to the low execution overhead, rapid instantiation, flexible management and process isolation that is sufficient for a vast majority of computation tasks outsourced to cloud infrastructure.

Container images are commonly stored in registries that contain multiple versions of images and additional 'layers' that allow to further customize the codebase. The continuous reuse of container images for instantiation in cloud infrastructure makes such registries an attractive target for malicious adversaries. A successful attack on the integrity of the container images can allow the adversary to inject malicious software (such as trojans, viruses, crypto-currency mining scripts, backdoors, etc.)

By verifying the integrity of image immediately prior to instantiation, the *Image Integrity Verifier* (IIV) enabler aims to reduce the risk to the integrity of the containers deployed by the COLA orchestration framework. Image integrity verification also allows to ensure reduce the security risks to the integrity and confidentiality of the workloads processed in the containers.

4.1.1.1 Status

An enabler prototype is under development. This is a preliminary specification and is subject to changes.

4.1.2 Copyright

4.1.3 Legal notice

N/A

4.1.4 Terms and definitions

Docker	Implementation of a lightweight virtualization framework
SDK	Software Development Kit
API	Application Programming Interface
IIV	Image Integrity Verifier
TEE	Trusted Execution Environment
TCB	Trusted Computing Base
TLS	Transport Layer Security

4.1.5 Overview

The IIV is used to verify the integrity of container images against an expected integrity value. While the integrity verification component can be designed as a generic solution, this open specification assumes an implementation based on Docker registry.

The functionality of the IIV revolves around determining whether one or more of image files are corrupted. Many attacks are focused on modification of critical files or configuration parameters. Especially, corrupted image files are considered as a substantial threat for cloud environment since a corrupted image can result in being unable to perform operations on a virtual machine/container. These operations include powering it on, taking a snapshot, and even modifying the virtual disks.

The enabler aims to be implementation-agnostic and one should be able to implement the principles of the enabler using different technologies. However, for the sake of clarity, the specification uses – where needed – concrete technologies and software applications.

Finally, it must be noted that the current API specification is likely to change both as the enabler itself matures, and as new features are added.

4.1.6 Basic concepts

Trusted Execution Environments (TEEs) guarantee isolated execution in the given adversary model, assuming correct implementation of the trusted computing base (TCB), e.g. the CPU and executed code. The TEE can be located on the same platform or on other platforms within the deployment.

Attested code and data in TEEs: Integrity of the code and data deployed in the TEEs is attested before any keys or key material is provisioned to the respective TEE. An appraiser under the control of the tenant performs the attestation of the TEE [28].

Trusted virtual images: an appraiser attests the integrity of the container images, including the software libraries in the images. Only verified container images are installed on the deployment. Authentication keys and other confidentiality and integrity sensitive cryptographic material is only stored in a verified image and never leaves its security perimeter [30].

Secure Communication Channels: The enabler protects communication channels container instances, as well as communication channels between container instances and external entities, when feasible. Communication security is ensured by verifying the image configuration against a baseline secure configuration, such that authentication credentials are correctly configured and are used to protect external communication [31], [32].

4.1.7 Main interactions

4.1.7.1 Use cases

In this section we describe two typical use cases for the *Integrity Verifier* enabler. Use cases are described in the "fully-dressed" format [29].

ID	IV-1
Title	Attest integrity of a container image prior to deployment
Description	Administrator obtains a quote of the container image TCB, verifies its authenticity and verifies that it matches the expected values.
Primary Actor	Administrator

Preconditions	The TCB of the container image is measured in a chain of trust originating in a hardware root of trust and reliably stored in an enclave, i.e. an isolated execution environment.
Post-condition	Administrator has obtained a statement of whether the container image TCB measurements match the expected values
Main success scenario	1. Administrator requests a quote of the container image TCB 2. Integrity verification component platform produces the quote 3. Integrity verification compoennt signs the quote and returns to the administrator. 4. Administrator verifies quote signature against the known public key of the Integrity Verification Component 5. Administrator matches quote against expected values for intended container image.
Extensions	4a1. The signature verification step shows that the quote signature is invalid. 4a2. Administrator either retries operation or excludes the container image from the list of container images in the registry.
Frequency of Use	At each deployment of the container image or as required for audit purposes.
Status	Design phase
Owner	RISE SICS

4.1.7.2 Components and interaction overview

Figure 21 displays the interactions of the IIV with the other components in the COLA architecture, in particular in relation with the Docker registry component[3].

According to the architecture of the Docker registry, the registry instance processes the requests for image instantiation, identifies the requested image in the repository and submits it for instantiation.
In the vanilla configuration, the submit request is sent to a broadcaster component, which instantiates the container image on one or more worker nodes.

To add integrity guarantees to the deployment, the image integrity verifier interacts with the broadcaster prior to the container images being submitted to the worker nodes. In particular, if the deployment request contains the integrity verification bit set, the broadcaster submits the container images to the IIV for verification. The IIV produces an integrity measurement of the image and verifies it against an expected value. In case the instance integrity value and the expected value math, the signed result of the verification, along with the signed image are returned to the broadcaster. Otherwise, the IIV returns only the signed result of the verification and not the corrupt image. This is done to prevent both accidental instantiation and delays in response time in the case when the image is corrupt. Finally, the broadcaster verifies the signature and submits the image for instantiation to the worker nodes according to the same flow as the vanilla configuration.

[3] Docker Registry: https://docs.docker.com/registry/

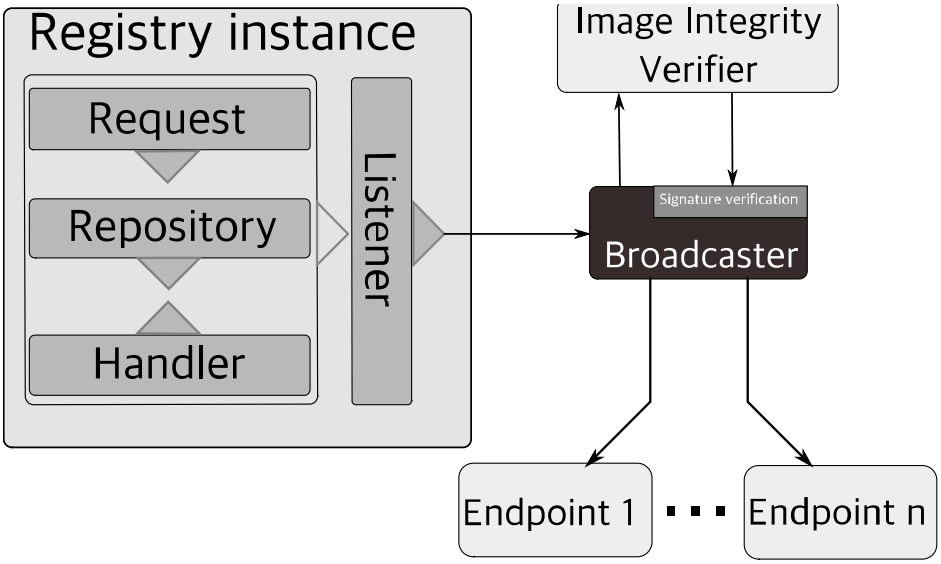


Figure 21 Component interaction for the image integrity verifier

4.1.7.3 Security requirements traceability

The IIV addresses the following requirements outlined in D7.1 COLA security requirements: CNSR-2, CNSR-6

4.1.7.4 Architecture objectives traceability

The IIV addresses the following security architecture objectives outlined in D7.2 MiCADO security architecture specification: O4.1, O4.4, O6.2

4.1.8 Architectural drivers

4.1.8.1 High-Level functional requirements

Authentication All communication between the orchestrator and integrity verifier must the authenticated; a secure signature verification mechanism must be in place.
Component integrity: Integrity of container images must be verified prior to deployment; the cryptographic material required for their deployment access must be protected strong encryption.
Confidentiality protection of domain secrets: Network domain secrets – such as VPN session keys – should not be revealed in plaintext even if the adversary succeeds in compromising the software stack on the host.
Confidentiality and integrity of network communication: All network communication in the tenant domain must be confidentiality and integrity protected.

4.1.8.2 Technical constraints

Designed for Docker container image verification.

4.1.8.3 Business constraints

No business constraints have been found at this point.

4.1.8.4 API specifications

1. ***Attest container image integrity***
    a. **Input**
        i. Tuple list <Image identity, Integrity Quote>
    b. **Output**
        i. Tuple list < Image identity, Attestation Result>
    c. **Comment**
        Attestation allows to verify whether the code and data executing in the enclave has not been modified and its fingerprint is identical to the expected values. This is a simple matching operation with a binary answer – the fingerprint either matches or does not. If the fingerprint matches, the container image is executing the expected code and data (assuming trust in the implementation of the platform). If the fingerprint does not match, nothing can be stated about the container image integrity.

4.1.9 Test plan

1. Test Items

#	Item to Test	Test Description
1	Image Integrity verifier	Test the functionality of measuring and verifying integrity values for container images.

2. Test features

#	Function to Test	Test Description
1	Fingerprint image	Test whether the function correctly generates a fingerprint of the container image.
2	Evaluate image integrity	Evaluate whether a given image matches the expected fingerprint.

3. Features not to be tested

Some features are not tested at this phase because they will be delayed for developing later or they belong to another test phase.

#	Feature not to be tested	Test Description
1	Image delta verification	Test whether two updates to a base container image are identical.
2	Encrypted image delta verification	Test whether two encrypted updates to a base container image are identical.

4. Approach

#	Function to Test	Test data description	Metrics to be collected	Pass/Fail criteria
1	Generate 256-bit image fingerprint	Data involves: input command, container image.	Correct/ Incorrect "Correct" means function produces a uniformly distributed 256-bit hash of the container image using the sha-256 algorithm; Incorrect otherwise	Precision = # of incorrect/ # of test runs Pass if precision = 1 Fail if precision<1
2	Evaluate image integrity	Data involves: input command, input container image, input expected fingerprint, input mismatching fingerprint.	Correct/ Incorrect "Correct" means the function produces the equivalence of the input container image and the expected fingerprint and produces an error in the case of the mismatching fingerprint; Incorrect otherwise	As above

4.1.10 Re-utilised Technologies/Specifications

Re-utilized technologies are presented in the table below:

Component	Role	Availability
Docker Registry	Verified Image Store	Open Source
OpenSSL	Cryptographic library	Open Source

The utilized components are modified where necessary for the purposes of the enabler.

4.2 Crypto Engine: Open specifications

4.2.1 Preface

Security components require a range of cryptographic functions, such as hash functions, symmetric key encryption schemes, public key encryption schemes, etc. The wide spectrum of use cases, architectures, and use case requirements, highlight the need to provide a diverse collection of encryption libraries rather than several fixed algorithms. A limited set of algorithms that cannot be easily configured can endanger the security of the entire deployment in case a severe vulnerability is found. However, a fragmented architecture where each security component relies on a distinct library is hard to maintain and update later. As a consequence, we introduce the *Crypto Engine*, a middle layer that provides the functionality of cryptographic libraries for security components. Based on industry best practice, the functionality of the

Crypto Engine includes standard or widely-adopted algorithms and may be updated in the future.

The Crypto Engine provides essential cryptographic functions for security components. It brings more flexibility and facilitates the maintenance of the MiCADO orchestration system.

4.2.1.1 Status

An enabler prototype is under development. This is a preliminary specification and is subject to changes.

4.2.2 Copyright

4.2.3 Legal notice

N/A

4.2.4 Terms and definitions

PKI	Public Key Infrastructure
RSA	Rivest-Shamir-Adleman public key encryption scheme
SHA	Secure Hashing Algorithm
MAC	Message authentication code

4.2.5 Overview

Crypto Engine enabler is a collection of cryptographic algorithms. It provides a list of cryptographic functions for other security components to perform certain cryptographic operations such as encryption, decryption, hashing, etc. Furthermore, it is responsible for generating new credentials, new keys, tokens, nonces, etc. In the context of the COLA security architecture, the Crypto Engine will be available as a separate micro-service and will be considered as one of the standard micro-services offered by COLA. The Crypto Engine is intended to support the following list of functions:

- Key Generation Orchestration;
- Operations using Symmetric Cipher Suites;
- Operations using Asymmetric Ciphers Suites;
- Digital Signature;
- Cryptographic Hash Function;
- Message Authentication Code (MAC);
- Token Generation.

4.2.6 Basic concepts

Symmetric-key algorithms are algorithms for cryptography that use the same cryptographic key to encrypt or decrypt ciphertext. The keys for encryption and decryption may be identical or may be obtained through a simple transformation. The keys, represent a shared secret between communicating parties and can be used to maintain a private information link [1].

Asymmetric cryptography or public key cryptographic systems use pairs of keys: *public keys* that do not carry any secret information, and *private keys* that must be maintained secret by the owner. Public key cryptographic systems accomplish two functions:

- Authentication, where the public key verifies that a holder of the private key encrypted the message.
- Encryption, where only the private key holder can decrypt the message encrypted with the public key.

Hash function – a function that takes as input an arbitrarily long string of bits or bytes and produces a fixed-sized result. The resulting output is also known as *digest* or *fingerprint*. The ideal hash function behaves like a random mapping from all possible input values to the set of all possible output values. An attack on a hash function is a non-generic method of distinguishing the hash function from an ideal hash function. Various hash functions exist and a full review of the existing hash functions is out of the scope of this document.

Message authentication code (MAC) – construction that detects tampering with messages. A MAC takes two arguments, a fixed-size key K and an arbitrarily sized message m, and produced a fixed-size MAC value. An ideal MAC function is a random mapping from all possible inputs to n-bit outputs.

4.2.7 Main interactions

4.2.7.1 Use cases

The following use cases address several main aspects of the functionality of the crypto engine. They do not represent an exhaustive list of use cases and should be seen as a sample that should be expanded further.

ID	CE-1
Title	Generate a public key cryptography key pair
Description	The Administrator – directly through the Security Policy Manager requests the generation of an RSA keypair.
Primary Actor	Administrator | the Security Policy Manager
Preconditions	Administrator defined a crypto security policy and configured the crypto engine according to the crypto security policy.
Post-condition	A keypair with at least 2048-bit security has been generated
Main success scenario	1. Primary actor selects key type: RSA | ECDSA 2. Primary actor issues command to generate key and store it in a pre-defined location. 3. Crypto Engine verifies request corresponds to security policy and generates X.509 certificate.
Extensions	1a. Primary actor selects encryption mode for the key to be generated
Frequency of Use	At each deployment of a component or workload.
Status	Design phase
Owner	RISE SICS

ID	CE-2
Title	Create X.509 certificates
Description	The Administrator – directly through the Security Policy Manager requests the generation of a X.509 certificate.
Primary Actor	Administrator | Security Policy Manager

Preconditions	Administrator defined a crypto security policy and configured the crypto engine according to the crypto security policy.
Post-condition	A certificate with at least 2048-bit security has been generated
Main success scenario	1. Primary actor selects certificate type: RSA | ECDSA 2. Primary actor selects certificate validity period 3. Primary actors select Certificate Authority to sign certificate 4. Primary actor issues command to generate key and store it in a pre-defined location. 5. Crypto Engine verifies request corresponds to security policy and generates X.509 certificate.
Extensions	1a. Primary actor selects encryption mode for the private key to be generated
Frequency of Use	At each deployment of a MiCADO component. Potentially at each deployment of a MiCADO workload.
Status	Design phase
Owner	RISE SICS

4.2.7.2 Security requirements traceability

The Crypto Engine directly addresses the following requirements outlined in D7.1 COLA security requirements: SR12, SR13, CNSR-3, CNSR-9, CSSR-1

Furthermore, the Crypto Engine supports a set of additional requirements outlined in D7.1 COLA security requirements: SR01, SR02, SR11, CNSR-7.

4.2.7.3 Architecture objectives traceability

The Crypto Engine directly addresses the following security architecture objectives outlined in D7.2 MiCADO security architecture specification: O3.1, O5.2

Furthermore, the Crypto Engine supports a set of additional security objectives outlined in D7.2 MiCADO security architecture specification: O1.1, O3.3, O2.2, O5.1, O4.1.

4.2.8 Architectural drivers

4.2.8.1 High-Level functional requirements

The high-level functional requirements towards the crypto engine component are based on: the requirements towards cryptographic security of: (a) the primitive operations performed by the crypto engine; (b) the cryptographic primitives produced by the Crypto Engine. The functional requirements are formulated below:

- The Crypto Engine should encrypt messages with keys that are at least 128-bit long. Less secure keys as well as deprecated algorithms MUST be rejected.
- The Crypto Engine should only accept the combination of parameters that allow symmetric-key encryption security in the chosen-plaintext attack model.
- The Crypto Engine should only encrypt messages using public-key cipher suites with keys that are at least 2048-bit long.

- The Crypto Engine should only accept the combination of parameters that allow public-key encryption security in the chosen-ciphertext attack model.
- The hash functions provided by the Crypto Engine must produce results that are preimage-resistant.

4.2.8.2 Technical constraints

The crypto engine is designed to run on platforms with the x86 32-bit or 64-bit architecture. Other architectures may be supported in a future release.

4.2.8.3 Business constraints

No business constraints have been found at this point.

4.2.8.4 API specifications

1. ***Generate public-private keypair***
    a. **Input**
        i. Function invocation – genKey
        ii. Parameters [crypto library, key type, encryption algorithm, encryption mode]
    b. **Output**
        i. Tuple list <public key, private key>
    c. **Comment**
    The choice of the crypto library could be pre-defined by the administrator in the crypto security policy.

2. ***Generate X.509 certificate***
    a. **Input**
        i. Function invocation – genCert
        ii. Parameters [crypto library, encryption algorithm, validity period, certificate authority, certificate storage location]
    b. **Output**
        i. X509 certificate
    c. **Comment**
    The choice of the crypto library, validity period and certificate authority could be pre-defined by the administrator in the crypto security policy.

3. ***Encrypt content using a symmetric cipher suite***
    a. **Input**
        i. Struct <Plaintext message, Encryption Key>
        ii. Parameters [crypto library, encryption algorithm, encryption mode]
    b. **Output**
        i. Tuple list < Result, Ciphertext message>
    c. **Comment**
    N/A
4. ***Decrypt content using a symmetric cipher suite***
    a. **Input**

i. Struct <Ciphertext message, Decryption Key>
ii. Parameters [crypto library, encryption algorithm, encryption mode]

b. Output

i. Tuple list < Plaintext message>

c. Comment

N/A

4.2.9 Test plan

1. Test Items

#	Item to Test	Test Description
1	Key generator	Test whether the component can generate keys according to specifications.
2	Nonce generator	Test whether the component can generate random numbers according to specifications
3	Encryption library	Test whether the component can encrypt and decrypt messages according to specifications

2. Test features

#	Function to Test	Test Description
1	Generate 128-bit symmetric key	Test whether the function correctly generates a 128-bit symmetric key with sufficient entropy.
2	Generate 256-bit symmetric key	Test whether the function correctly generates a 256-bit symmetric key with sufficient entropy.
3	Generate 2048-bit asymetric key	Test whether the function correctly generates a 2048-bit public-private keypair with sufficient entropy.
4	Generate random nonce	Test whether the function generates random numbers with sufficient entropy.
5	Encrypt and decrypt data	Test whether the function works properly and correctly encrypts and decrypts sample inputs using a given key, encryption algorithm and encryption mode.
6	Generate X509 certificate	Test whether the function correctly generates a well-formed X509 certificate.

3. Features not to be tested

Some features are not tested at this phase because they will be delayed for developing later or they belong to another test phase.

#	Feature not to be tested	Test Description

1	Symmetric Searchable encryption	Test whether the component correctly implements symmetric searchable encryption.
2	Asymmetric Searchable encryption	Test whether the component correctly implements asymmetric searchable encryption.
3	Probabilistic encryption	Test whether the component correctly implements probabilistic encryption.

4. Approach

#	Function to Test	Test data description	Metrics to be collected	Pass/Fail criteria
1	Generate 256-bit symmetric key	Data involves: input command, key type.	Correct/ Incorrect "Correct" means function produces a uniformly distributed 256-bit sequence; Incorrect otherwise	Precision = # of incorrect/ # of test runs Pass if precision = 1 Fail if precision<1
2	Generate 2048-bit asymetric key	Data involves: input command, key type.	Correct/ Incorrect "Correct" means function produces a uniformly distributed 2048-bit sequence; Incorrect otherwise	As above
3	Generate random nonce	Data involves: input command, nonce size	Correct/ Incorrect "Correct" means function produces a uniformly distributed sequence of a given size; Incorrect otherwise	As above
4	Encrypt and decrypt data	Data involves: input command, input data, encryption/decryption key, encryption/decryption cipher and mode	Correct/ Incorrect "Correct" means function produces a pseudorandom sequence (for encryption) that equals the input plaintext when decrypted (for decryption); Incorrect otherwise	As above
5	Generate X509 certificate	Data involves: input command, certificate input data, encryption/decryption cipher and mode	Correct/ Incorrect "Correct" means function produces a valid X509 certificate with	As above

			correct input data. Incorrect otherwise	

4.2.10 Reused Technologies/Specifications

The Crypto Engine comprises functionality from a range of widely used cryptographic libraries. While many of the cryptographic libraries provide the same functionality, they differ in computation performance, feature set and support for new hardware features. The cryptographic libraries that can be included in the crypto engine are as follows:

Component	Feature	Availability
OpenSSL	Multipurpose	Open Source
WolfSSL	Multipurpose	Proprietary, Code Available Open Source
mbedTLS	Support for Intel SGX	Open Source

4.3 Security Policy Manager: Open specifications

4.3.1 Preface

The Security Policy Manager is a component of MiCADO, that is in charge of security requirements for each of the business processes within the COLA framework and has connectors to perform the configuration of the security enablers to fullfill those requirements.

In the MiCADO architecture, the various security enablers require a single entry point for providing security functions to other services within the architecture. Security components are managed through a number of configuration files and APIs, which makes them hard to implement in the various different components of MiCADO without locking the project to a single implementation of the security enabler. The SPM remedies that by providing a comprehensive, implementation-agnostic REST API to wrap security functions by a standardized interface.

4.3.1.1 Status

An enabler prototype is under development. This is a preliminary specification and is subject to changes.

4.3.2 Copyright

4.3.3 Legal notice

N/A

4.3.4 Terms and definitions

TOSCA	Topology and Orchestration Specification for Cloud Applications is a specification format that provides a language to describe service components and their relationships using a service topology in a cloud environment.

4.3.5 Overview

The exact design will be available and updated after the competition of the Design phase.

4.3.6 Basic concepts

The exact design will be available and updated after the competition of the Design phase.

4.3.7 Main interactions

The exact design will be available and updated after the competition of the Design phase.

4.3.7.1 Use cases

ID	SPM-1
Title	Provision a new worker node
Description	Create a new worker node in the cloud infrastructure and put security safeguards in place
Primary Actor	Cloud Orchestrator
Preconditions	The SPM, Cloud Orchestrator, Zorp SSL and PKI components are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	A new worker node is provisioned with security in place
Main success scenario	1. The Cloud Orchestrator notifies the SPM that a new worker node is to be provisioned. 2. SPM instructs PKI to generate a new keypair using the internal CA for the newly created worker and assignes a token to the new worker. 3. SPM notifies the Cloud Orchestrator to add the token as a parameter for provisioning the new worker. 4. SPM notifies Zorp SSL Master of the newly created token. 5. Upon initial startup Zorp SSL Worker connects to Zorp SSL Master using its token to acquire the TLS keypair, saves them locally and starts listening for incoming requests using the new keypair. 6. Zorp SSL Master accepts connection from Zorp SSL Worker, verifies its IP address and token and serves the newly created keypair from PKI. 7. Zorp SSL Master removes the token from its list.
Extensions	
Frequency of Use	This may happen infrequently, whenever the Optimiser component decides to provision a new worker node due the heavy workload of the application.
Status	Design phase
Owner	BalaSys

ID	SPM-2
Title	Decomission a worker node
Description	Tear down an existing worker node and destroy all security identifiers associated with them to avoid replay attacks
Primary Actor	Cloud Orchestrator

Preconditions	The SPM, Cloud Orchestrator, Zorp SSL and PKI components are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	The affected worker node is destroyed and its security identifiers revoked
Main success scenario	1. The Cloud Orchestrator notifies SPM that a new worker node is to be decommissioned. 2. SPM instructs PKI to revoke the keypair associated with the worker node within the internal CA. 3. SPM notifies Zorp SSL Master to refresh its revocation list.
Extensions	
Frequency of Use	This may happen infrequently, whenever the Optimiser component decides to tear down a worker node due the reduced workload of the application.
Status	Design phase
Owner	BalaSys

ID	SPM-3
Title	Provision a new MiCADO application
Description	Upon provisioning a new MiCADO application, verify that the TOSCA description has correctly configured security policies and implement them
Primary Actor	MiCADO administrator
Preconditions	All core components of the MiCADO master node are initialized and running
Post-condition	The requested application is provisioned with all configured security policies implemented and functional
Main success scenario	1. The user initiates a TOSCA descriptor submission by sending an authenticated HTTPS request to Zorp Firewall. 2. Zorp verifies the credentials and group membership of the user and if applicable forwards the request to the TOSCA Submitter. 3. The TOSCA submitter verifies the format of the descriptor and passes relevant information to the various Manager components of the MiCADO master. 4. SPM verifies the syntax and semantics of the security policies within the TOSCA descriptor (and instructs the submitter to relay an error message to the user if validation fails). 5. SPM instructs the Submitter if a network topology change within the application is required to implement security policies (e.g. for firewalling). 6. SPM generates the configuration for Zorp Firewall to implement network security policies. 7. The return value is sent back to the Submitter that forwards it to the user. 8. SPM persists the state of the application's security configuration.
Extensions	
Frequency of Use	Infrequently, when deploying a new MiCADO application

Status	Design phase
Owner	BalaSys

ID	SPM-4
Title	Change an existing MiCADO application
Description	Upon changing the configuration of an existing MiCADO application, verify that the TOSCA description has correctly configured security policies and implement them
Primary Actor	MiCADO administrator
Preconditions	All core components of the MiCADO master node are initialized and running
Post-condition	The requested application is provisioned with all configured security policies implemented and functional
Main success scenario	1. The user initiates a TOSCA descriptor submission by sending an authenticated HTTPS request to Zorp Firewall. 2. Zorp verifies the credentials and group membership of the user and if applicable forwards the request to the TOSCA Submitter. 3. The TOSCA submitter verifies the format of the descriptor and passes relevant information to the various Manager components of the MiCADO master. 4. SPM verifies the syntax and semantics of the security policies within the TOSCA descriptor (and instructs the submitter to relay an error message to the user if validation fails). 5. SPM looks up the persisted state of the application and calculates changes necessary for implementing the new configuration. 6. SPM instructs the Submitter if a network topology change within the application is required to implement security policies (e.g. for firewalling). 7. SPM generates the configuration for Zorp Firewall to implement network security policies. 8. The return value is sent back to the Submitter that forwards it to the user. 9. SPM persists the state of the application's security configuration.
Extensions	
Frequency of Use	Infrequently, when changing the configuration of a MiCADO application
Status	Design phase
Owner	BalaSys

ID	SPM-5
Title	Remove an existing MiCADO application
Description	Upon removing an existing MiCADO application, remove persistent state
Primary Actor	MiCADO administrator
Preconditions	MiCADO users's registered credentials have been stored in CBS. Zorp has obtained credential from a subscriber who accesses to MiCADO.
Post-condition	The MiCADO application and its persistent state is removed

Main success scenario	1. The user initiates a TOSCA descriptor submission by sending an authenticated HTTPS request to Zorp Firewall. 2. Zorp verifies the credentials and group membership of the user and if applicable forwards the request to the TOSCA Submitter. 3. The TOSCA submitter verifies the format of the descriptor and passes relevant information to the various Manager components of the MiCADO master. 4. SPM looks up the persisted state of the application and discards it. 5. The return value is sent back to the Submitter that forwards it to the user.
Extensions	
Frequency of Use	Infrequently, when tearing down a MiCADO application
Status	Design phase
Owner	BalaSys

ID	SPM-6
Title	Add a new MiCADO credential
Description	Add a new MiCADO user with its authenticator via CLI
Primary Actor	MiCADO administrator
Preconditions	The Administrator has SSH access to the master node, SPM and CM are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	A new credential is added with its authenticator and roles set up correctly and can be used to authenticate user access.
Main success scenario	1. The Administrator logs in to the master node via SSH. 2. The Administrator runs a command to add the new credential with username and role as its parameter. 3. The command line tool asks the Administrator to supply the authenticator interactively or via a command line parameter. 4. The command line tool initiates a REST call to the SPM to add the new user. 5. SPM initiates a REST call to CM to add the new user. 6. CM replies to SPM if the addition was successful which relays the answer to the command line tool. 7. The Administrator is notified textually of the result and the return code of the command line tool is set accordingly.
Extensions	The use case can be extended to support graphical management of users.
Frequency of Use	Infrequently, when adding new MiCADO users
Status	Design phase
Owner	BalaSys

ID	SPM-7
Title	Remove an existing MiCADO credential

Description	Remove an existing MiCADO user and its authenticator via CLI
Primary Actor	MiCADO administrator
Preconditions	The Administrator has SSH access to the master node, SPM and CM are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	The existing credential is removed and cannot be further used to authenticate user access.
Main success scenario	1. The Administrator logs in to the master node via SSH. 2. The Administrator runs a command to remove the existing credential with username as its parameter. 3. The command line tool asks the Administrator for confirmation of the removal interactively or via a command line parameter. 4. The command line tool initiates a REST call to the SPM to remove the user. 5. SPM initiates a REST call to CM to remove the new user. 6. CM replies to SPM if the removal was successful which relays the answer to the command line tool. 7. The Administrator is notified textually of the result and the return code of the command line tool is set accordingly.
Extensions	The use case can be extended to support graphical management of users.
Frequency of Use	Infrequently, when removing MiCADO users
Status	Design phase
Owner	BalaSys

ID	SPM-8
Title	Reset the authenticator of an existing MiCADO credential
Description	Change the authenticator of an existing MiCADO user via CLI
Primary Actor	MiCADO administrator
Preconditions	The Administrator has SSH access to the master node, SPM and CM are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	The authenticator of the relevant user is changed with other attributes kept unchanged, the previous authenticator is invalidated
Main success scenario	1. The Administrator logs in to the master node via SSH. 2. The Administrator runs a command to change the credential with username as its parameter. 3. The command line tool asks the Administrator to supply the authenticator interactively or via a command line parameter. 4. The command line tool initiates a REST call to the SPM to change the user's authenticator. 5. SPM initiates a REST call to CM to change the authenticator. 6. CM replies to SPM if the change was successful which relays the answer tot he command line tool. 7. The Administrator is notified textually of the result and the return code of the command line tool is set accordingly.
Extensions	The use case can be extended to support graphical management of users.

Frequency of Use	Infrequently, when changing MiCADO users' authenticator by the administrator
Status	Design phase
Owner	BalaSys

ID	SPM-9
Title	Add a new cloud credential
Description	Add a new cloud user with its authenticator via CLI
Primary Actor	MiCADO administrator
Preconditions	The Administrator has SSH access to the master node, SPM and CS are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	A new credential is added with its authenticator and can be used to perform cloud operations.
Main success scenario	1. The Administrator logs in to the master node via SSH. 2. The Administrator runs a command to add the new credential with cloud endpoint and username as its parameter. 3. The command line tool asks the Administrator to supply the authenticator interactively or via a command line parameter. 4. The command line tool initiates a REST call to the SPM to add the new credential. 5. SPM initiates a REST call to CS to add the new credential. 6. CS replies to SPM if the addition was successful which relays the answer to the command line tool. 7. The Administrator is notified textually of the result and the return code of the command line tool is set accordingly.
Extensions	The use case can be extended to support graphical management of cloud credentials.
Frequency of Use	Infrequently, when adding cloud credentials by the Administator
Status	Design phase
Owner	BalaSys

ID	SPM-10
Title	Remove an existing cloud credential
Description	Remove an existing MiCADO user and its authenticator via CLI
Primary Actor	MiCADO administrator
Preconditions	The Administrator has SSH access to the master node, SPM and CS are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	The existing credential is removed and cannot be further used to perform cloud operations.
Main success scenario	1. The Administrator logs in to the master node via SSH. 2. The Administrator runs a command to remove the existing credential with username as its parameter. 3. The command line tool asks the Administrator for confirmation of the removal interactively or via a command line parameter.

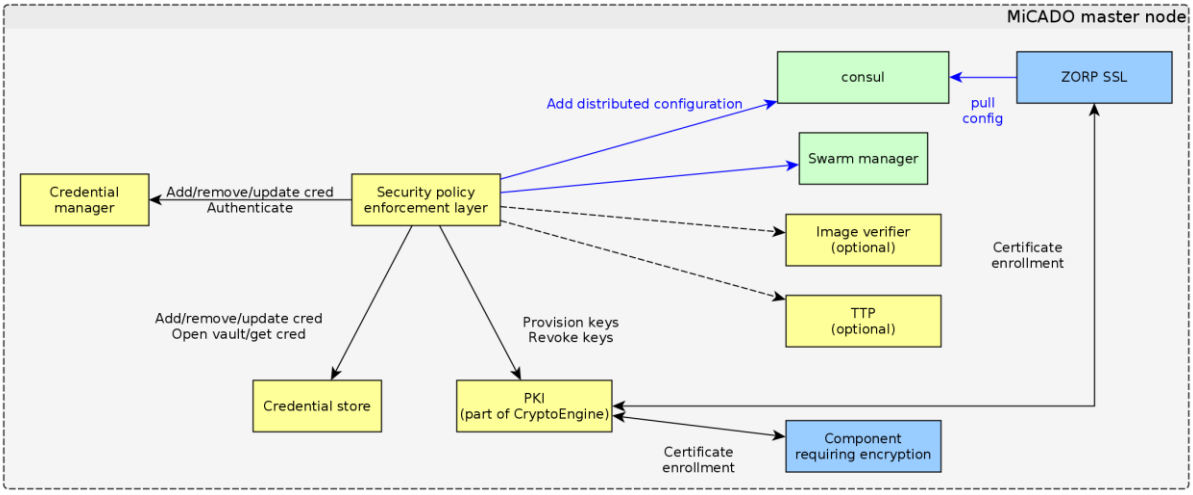
	4. The command line tool initiates a REST call to the SPM to remove the user. 5. SPM initiates a REST call to CS to remove the new user. 6. CS replies to SPM if the removal was successful which relays the answer tot he command line tool. 7. The Administrator is notified textually of the result and the return code of the command line tool is set accordingly.
Extensions	The use case can be extended to support graphical management of users
Frequency of Use	Infrequently, when removing cloud credentials by the administrator
Status	Design phase
Owner	BalaSys

ID	SPM-11
Title	Change an existing cloud credential
Description	Change the authenticator of an existing MiCADO user via CLI
Primary Actor	MiCADO administrator
Preconditions	The Administrator has SSH access to the master node, SPM and CS are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	The authenticator of the relevant user is changed with other attributes kept unchanged, the previous authenticator is invalidated.
Main success scenario	1. The Administrator logs in to the master node via SSH. 2. The Administrator runs a command to change the credential with username as its parameter. 3. The command line tool asks the Administrator to supply the authenticator interactively or via a command line parameter. 4. The command line tool initiates a REST call to the SPM to change the user's authenticator. 5. SPM initiates a REST call to CS to change the authenticator. 6. CS replies to SPM if the change was successful which relays the answer to the command line tool. 7. The Administrator is notified textually of the result and the return code of the command line tool is set accordingly.
Extensions	The use case can be extended to support graphical management of users.
Frequency of Use	Infrequently, when changing the authenticator of a cloud credential by the administrator
Status	Design phase
Owner	BalaSys

ID	SPM-12
Title	Retrieve an existing cloud credential
Description	Retrieve an existing cloud credential by the Cloud Orchestrator for performing scaling operations
Primary Actor	Cloud Orchestrator
Preconditions	The Cloud Orchestrator, SPM and CM are successfully initialized and running as a Docker container on the MiCADO master node.

Post-condition	The credential is returned to the Cloud Orchestrator in plain text format.
Main success scenario	1. The Cloud Orchestrator initiates a REST call to the SPM to retrieve a cloud credential‘s authenticator. 2. SPM initiates a REST call to CS to retrieve the authenticator. 3. CS returns the credential to the SPM which relays the answer to the CO. 4. The credential is discarded by the SPM and CO after performing the operation.
Extensions	The use case can be extended to support graphical management of users.
Frequency of Use	Frequently, when performing automatic cloud operations by the Cloud Orchestrator
Status	Design phase
Owner	BalaSys

4.3.7.2 Components and interaction overview



4.3.7.3 Security requirements traceability

Zorp Firewall addresses the following requirements outlined in D7.1 COLA security requirements: SR05, SR06, SR10, CNSR-1, CNSR-2, CNSR-3, CNSR-4, CNSR-5, CNSR-6, CNSR-7, CNSR-8, CNSR-9, CNSR-10

4.3.7.4 Architecture objectives traceability

The CM addresses the following security architecture objective outlined in D7.2 MiCADO security architecture specification: O1.1, O4.2, O4.3, O4.4, O6.1, O6.2

4.3.8 Architectural drivers

4.3.8.1 High-Level functional requirements

Verify and implement MiCADO application security policies: Provisioned applications may contain security policies that are provided by the MiCADO infrastructure. These policies need to be verified upon submission of a new TOSCA descriptor and in case of validation failure a

user-comprehensible error status should be relayed back to the submitting administrator. If verification is successful, the translation of the policies to concrete configuration elements is done by the SPM.
Standardize PKI and credential management interfaces: The SPM acts as a wrapper above all security enablers' native interfaces to make sure, that the security services provided to other microservices are comprehensive and implementation-agnostic.

4.3.8.2 Technical constraints

No technical constraints identified currently.

4.3.8.3 Business constraints

No know business constraint.

4.3.8.4 API specifications

1. ***Provision new cryptographic keys***
    a. **Input**
        i. KeyProvisioningRequest <CommonName, IP Address>
    b. **Output**
        i. Return value ALREADY_EXISTS or Access token that can be used to retrieve the generated keypair
    c. **Comment** Provisioning cryptographic keys aims to generate a new keypair for authentication by the internal Certification Authority and provide a means to securely retrieve them. These are then used for securing internal communication within the distributed architecture.

2. ***Revoke cryptographic keys***
    a. **Input**
        i. KeyRevocationRequest <CommonName>
    b. **Output**
        i. Return value NOT_EXIST, SUCCESS
    c. **Comment** Revoking cryptographic keys aims to make an existing keypair unfit for futher authentication within the distributed architecture. This happens if a key is compromised, or a worker node is automatically decommissioned by the scaling logic.

3. ***Provision new MiCADO application***
    a. **Input**
        i. ToscaNewApplication <Application name, Relevant TOSCA parts>
    b. **Output**
        i. Return value ALREADY_EXISTS or VALIDATION_FAILURE and exact cause or SUCCESS and network topology information
    c. **Comment** The new application request aims to verify if the security policies within the newly submitted TOSCA description are syntactically and

semantically correct and if yes, translate the policies into concrete configuration elements of the MiCADO infrastructure. In case of any validation error, a user-comprehensible error message should be returned.

4. ***Change an existing MiCADO application***
    a. **Input**
        i. ToscaChangeApplication <Application name, Relevant TOSCA parts>
    b. **Output**
        i. Return value NOT_EXISTS or VALIDATION_FAILURE and exact cause or SUCCESS and network topology information
    c. **Comment** The change application request aims to verify if the security policies within the submitted TOSCA description are syntactically and semantically correct and if yes, change the existing MiCADO security infrastructure to reflect the changes to the descriptor.

5. ***Remove an existing MiCADO application***
    a. **Input**
        i. ToscaRemoveApplication <Application name>
    b. **Output**
        i. Return value NOT_EXISTS or SUCCESS
    c. **Comment** The change application request aims to remove any persistent state data to be removed alongside with the application.

6. ***Add a new MiCADO credential***
    a. **Input**
        i. MCredentialNew <Identity, Role, Authenticator>
    b. **Output**
        i. Return value ALREADY_EXISTS or SUCCESS
    c. **Comment** Adding new credentials enables the Administrator to grant access to stakeholders of different roles to the MiCADO infrastructure.

7. ***Remove MiCADO credential***
    a. **Input**
        i. MCredentialRemove <Identity >
    b. **Output**
        i. Return value NOT_EXIST or SUCCESS
    c. **Comment** Removing credentials allows the Administator to revoke access to the MiCADO infrastructure.

8. ***Reset MiCADO credential‘s authenticator***
    a. **Input**
        i. MCredentialChangeCred <Identity, Authenticator>
    b. **Output**
        i. Return value NOT_EXIST or SUCCESS

c. **Comment** The credential reset functionality allows the Administrator to reset the state of the accounts with forgotten or compromised authenticators.

9. Add a new cloud credential

a. **Input**
   i. CCredentialNew <Endpoint, Identity, Authenticator>
b. **Output**
   i. Return value ALREADY_EXISTS or SUCCESS
c. **Comment** Adding new credentials enables the Administrator to grant access to cloud services for the Cloud Orchestrator component.

10. Remove cloud credential

a. **Input**
   i. CCredentialRemove <Endpoint, Identity>
b. **Output**
   i. Return value NOT_EXIST or SUCCESS
c. **Comment** Removing credentials allows the Administator to revoke access to cloud services.

11. Change cloud credential's authenticator

a. **Input**
   i. CCredentialChangeCred <Endpoint, Identity, Authenticator>
b. **Output**
   i. Return value NOT_EXIST or SUCCESS
c. **Comment** The credential reset functionality allows the Administrator to reset the authenticator of compromised cloud accounts.

12. Retrieve cloud credential

a. **Input**
   i. CCredentialGet <Identity, Authenticator, Endpoint>
b. **Output**
   i. Return value NOT_EXIST, WRONG_PASS or SUCCESS and credential in key-value format
c. **Comment** The credential retrieval functionality allows the Cloud Orchestration component to keep sensitive credentials in a secure container. This way data at rest is in an encrypted storage, while data in motion is minimized for the duration of any cloud operation.

4.3.9 Test plan

1. Test items

#	Item to Test	Test Description

1	Security Policy Manager (SPM)	Test whether the component can communicate with CM, CS, CO, PKI, TOSCA Submitter and Zorp SSL, and works properly or not
2	Credential Store (CS)	Test whether the component can communicate with SPM, and works properly or not
3	Credential Manager (CM)	Test whether the component can communicate with SPM, and works properly or not
4	Container Orchestrator (CO)	Test whether the component can communicate with SPM, and works properly or not
5	TOSCA Submitter	Test whether the component can communicate with SPM, and works properly or not
6	Public Key Infrastructure (PKI)	Test whether the component can communicate with SPM, and works properly or not
7	Zorp SSL	Test whether the component can communicate with SPM, and works properly or not

2. Test features

#	Function to Test	Test Description
1	Provision new cryptographic keys	Test whether the function works properly and returns correct response
2	Revoke cryptographic keys	Test whether the function works properly and returns correct response
3	Provision new MiCADO application	Test whether the function works properly and returns correct response
4	Change existing MiCADO application	Test whether the function works properly and returns correct response
5	Remove existing MiCADO application	Test whether the function works properly and returns correct response
6	Add new MiCADO credential	Test whether the function works properly and returns correct response
7	Remove existing MiCADO credential	Test whether the function works properly and returns correct response
8	Reset authenticator of a MiCADO credential	Test whether the function works properly and returns correct response
9	Add new cloud credential	Test whether the function works properly and returns correct response
10	Remove an existing cloud credential	Test whether the function works properly and returns correct response

11	Change an existing cloud credential	Test whether the function works properly and returns correct response
12	Retrieve and existing cloud credential	Test whether the function works properly and returns correct response

3. Approach

#	Function to Test	Test data description	Metrics to be collected	Pass/Fail criteria
1	Provision new cryptographic keys	Data involves two cases: already existing keypair and new keypair	Correct/ Incorrect	Precision = # of incorrect/ # of test runs Pass if precision = 1 Fail if precision<1
2	Revoke cryptographic keys	Data involves cases: non-existent keypair and existing keypair	Correct/ Incorrect	As above
3	Provision new MiCADO application	Data involves cases: descriptor with incorrect security policy configuration, descriptor with non-existent security policy, valid descriptor, already existing application	Correct/ Incorrect	As above
4	Change existing MiCADO application	Data involves cases: descriptor with incorrect security policy configuration, descriptor with non-existent security policy, valid descriptor, non-existent application	Correct/ Incorrect	As above
5	Remove existing MiCADO application	Data involves cases: non-existent application, existing application	Correct/ Incorrect	As above
6	Add new MiCADO credential	Data involves cases: non-existent user and existing user	Correct/ Incorrect	As above
7	Remove existing	Data involves cases: non-existent user and existing user	Correct/ Incorrect	As above

	MiCADO credential			
8	Reset authenticator of a MiCADO credential	Data involves cases: non-existent user and existing user	Correct/ Incorrect	As above
9	Add new cloud credential	Data involves cases: non-existent credential and existing credential	Correct/ Incorrect	As above
10	Remove an existing cloud credential	Data involves cases: non-existent credential and existing credential	Correct/ Incorrect	As above
11	Change an existing cloud credential	Data involves cases: non-existent credential and existing credential	Correct/ Incorrect	As above
12	Retrieve and existing cloud credential	Data involves cases: non-existent credential and existing credential	Correct/ Incorrect	As above

4.3.10 Re-utilised Technologies/Specifications

Component	Role	Availability
Flask Python Framework	Software library to provide easy development of REST APIs	Open Source

The utilized components are modified where necessary for the purposes of the enabler.

4.4 Credential Manager: Open specifications

4.4.1 Preface

MiCADO infrastructure is not publicly available. More precisely, only authorized users are eligible to access the services provided by MiCADO. Therefore, authorized users need to authenticate themselves prior to the deployment of their applications in the underlying infrastructure. In addition to that, the administrator also needs to be authenticated to perform *any* management actions. Furthermore, the Credential Manager is combined with Zorp to provide authentication for MiCADO.

4.4.1.1 Status

An enabler prototype is under development. This is a preliminary specification and is subject to changes.

4.4.2 Copyright

4.4.3 Legal notice

N/A

4.4.4 Terms and definitions

Subscriber or Claimant	Any entity that needs to be authenticated. In the scope of MiCADO framework, a subscriber refers to a user or an administrator
Identity	An attribute that uniquely identifies a subscriber (e.g., username)
Authenticator or Token	Information that the subscriber owns and uses for authentication (e.g., password)
Credential	An object containing an identity (e.g. username) of a subscriber binding with its authenticator (e.g. password) which the subscriber possesses
Verifier	Entity that verifies the subscriber's authenticator
Authentication protocol	A protocol that runs between the subscriber and the verifier and authenticates the subscriber to the verifier (i.e. subscriber is a legitimate user)

Table 5 Terms and definitions for authentication [6][21]

4.4.5 Overview

In the scope of MiCADO framework, authentication is explicitly performed by the master node. After a user has been successfully authenticated she is allowed to use the underlying infrastructure. Hence, in our authentication protocol we only consider the following two entities:

- Subscriber/claimant who can be any MiCADO user or administrator;
- The verifier which is MiCADO's master node. More precisely, the verifier is implemented through a combination of Zorp and the Credential Manager component. Apart from that, the Credential Manager manages the backend storage for users' credentials which are used to authenticate the subscribers.

4.4.6 Basic concepts

Zorp is an open source software that provides access control and token management. When a user/administrator connects to MiCADO, Zorp acts as a gateway that tries to authenticate the user prior allowing it to the application deployment/performing management. Zorp, connects to the Credential Manager to verify the user's identity. If the verification is successful, Zorp allows the user to deploy applications in MiCADO infrastructure, or the administrator to execute management tasks.

Credential Manager (CM) is the mediator between Zorp and the credentials backend storage. It plays the role of verifier. In order to authenticate a user, CM receives the user's credentials from Zorp, then connects to the backend user storage to request for the user's stored authenticator. After that, CM performs a verification to check if the received information from Zorp matches the stored authenticator.

Credential Backend Storage (CBS) stores all MiCADO users's authenticators along with identity information (e.g. username). There are various types of authenticators such as passwords, certificates, pin numbers etc., and different types of backend storage such as files, relational database.

4.4.7 Main interactions

4.4.7.1 Use cases

In this section, we describe a set of typical use cases for the *Credential Manager* enabler. The use cases are described in the "fully-dressed" format [29].

Table 6 Use case CM-1: Authentication

ID	CM-1
Title	Authenticate a user/ an administrator prior to allowing the user's application deployment/ the administrator's management actions
Description	CM obtains the subscriber's (user/ administrator) credential from Zorp, compare it with the stored credential in the backend storage (CBS).
Primary Actor	Zorp Credential Manager (CM)
Preconditions	MiCADO users's registered credentials have been stored in CBS. Zorp has obtained credential from a subscriber who accesses to MiCADO.
Post-condition	Zorp obtains a statement of whether the subscriber is authenticated successful or not
Main success scenario	1. Zorp sends the subscriber's credential to CM 2. CM connects to CBS to query for the authenticator and its role based on the identity contained in the received credential. 3. If CBS returns no result, meaning that there does not exist the queried identity in CBS, return NOT_EXIST 4. CBS returns CM an authencator corresponding to the queried identity 5. CM compares the authenticator in the received credential from Zorp and the authenticator sent back by CBS 6. If comparison is not matched, return WRONG_PASS. Otherwise, go to Step 8. 7. Return ROLE _VALUE (user/admin) that is queried in Step 3. 8. The return value is sent back to Zorp 9. Zorp relies on the return value to allow access to suitable services or not
Extensions	The use case can be extended to support lock-out mechanism after a fixed number of log-in fails. In addition to specific authentication result in the main scenario, the function may include a general message to be returned to users. That would prevent malicious users from knowing the true reason of any failed authentication.
Frequency of Use	At each user/ admin's log in
Status	Design phase
Owner	UoW

Table 7 Use case CM-1: Add new identity

ID	CM-2
Title	Add a new identity and its role
Description	CM adds a new credential and its role
Primary Actor	Credential Manager (CM) Security Policy Manager (SPM) CryptoEngine (CE)
Preconditions	SPM is keeping a new identity and its role that are added into MiCADO by the administrator. Verifying that the user who request to add new identity is an administrator has been done.
Post-condition	CM adds the new identity with a default value for authenticator, along with its role to credential storage backend (CBS)
Main success scenario	1. SPM sends the identity and its role to CM 2. CM queries for the identity in the credential from CBS 3. If CBS returns the corresponding identity, return EXISTED. Otherwise, go to step 4. 4. CM calls a random generator from CE to generate a random value for the authenticator default value. 5. CM adds the credential <identity, default authenticator> and its role to CBS. Return DEFAULT_PASS which is the generated default authenticator.
Extensions	When administrator adds a new identity, it is required to input the identity's email. After successful new identity insertion, the default authenticator is sent to the identity's email.
Frequency of Use	At each user/ admin's log in
Status	Design phase
Owner	UoW

Table 8 Use case CM-3: Change authenticator

ID	CM-3
Title	Change an authenticator
Description	CM changes the authenticator of an existing credential to a new authenticator value
Primary Actor	Credential Manager (CM) Security Policy Manager (SPM)
Preconditions	SPM is keeping the credential and its new authenticator. Verifying that the new authenticator is satisfied with password-policies has been done.
Post-condition	CM update the authenticator if the credential is verified.
Main success scenario	1. SPM sends the credential and its new authenticator to CM 2. CM verifies the credential. If verification returns NOT_EXIST or WRONG_PASS, return NOT_EXIST or WRONG_PASS. Otherwise, go to step 3.

	3. Update the credential's authenticator with the new authenticator. Return SUCCESS.
Extensions	
Frequency of Use	At each user/ admin's request
Status	Design phase
Owner	UoW

Table 9 Use case CM-4: Reset authenticator

ID	CM-4
Title	Reset an authenticator by administrator
Description	Administrator resets the authenticator of an existing identity
Primary Actor	Credential Manager (CM) Security Policy Manager (SPM) Crypto Engine (CE)
Preconditions	SPM is keeping the identity. Administrator has been verified.
Post-condition	CM update the authenticator if the credential is verified.
Main success scenario	1. SPM sends the identity to CM 2. CM queries for the identity. If it does not exist, return NOT_EXIST. Otherwise, go to step 3. 3. CM uses a random generator provided by CE to generates a random value. Check to ensure that the generated value is different from NOT_EXIST. 4. Set the corresponding authenticator to the generated value. Return DEFAULT_PASS which is the generated value.
Extensions	Extending the function for reset an authenticator by the subscriber itself. An email is sent to the subscriber to notify about the reset of password.
Frequency of Use	At each admin's request
Status	Design phase
Owner	UoW

Table 10 Use case CM-5: Use case CM-5: Delete identity

ID	CM-5
Title	Delete an identity
Description	CM deletes an identity
Primary Actor	Credential Manager (CM) Security Policy Manager (SPM)
Preconditions	The user who issues this request is authenticated as an administrator. He/she does not delete themselves.
Post-condition	CM deletes the identity
Main success scenario	1. SPM sends the identity CM 2. CM queries for the identity. If it does not exist, return NOT_EXIST. Otherwise, go to step 3. 3. CM deletes the identity out of CBS. Return SUCCESS

Extensions	A notification email is sent to the removed identity.
Frequency of Use	At each admin's request
Status	Design phase
Owner	UoW

4.4.7.2 Components and interaction overview

The following figures illustrate the interactions of the CM with the other components in the MiCADO architecture, in particular in relation with Security Policy Manager and Zorp. For the first stage of implementation, user's identity is defined by username and authenticator is defined by password. CBS is implemented as a simple file in CM's filesystem. Therefore, interaction between CM and CBS is considered as self-interaction of CM.

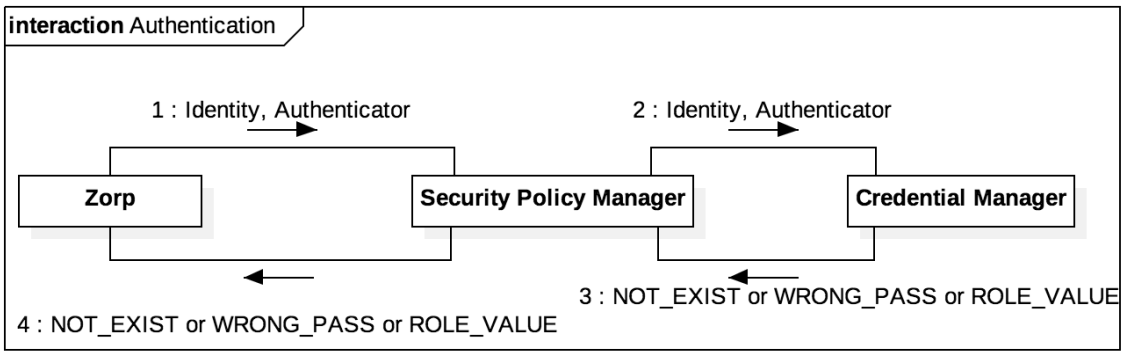


Figure 22 Component interaction for the credential manager in the use case CM-1

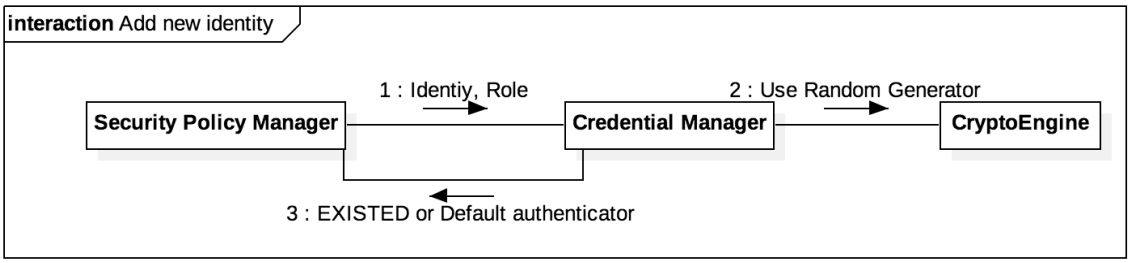


Figure 23 Component interaction for the credential manager in the use case CM-2

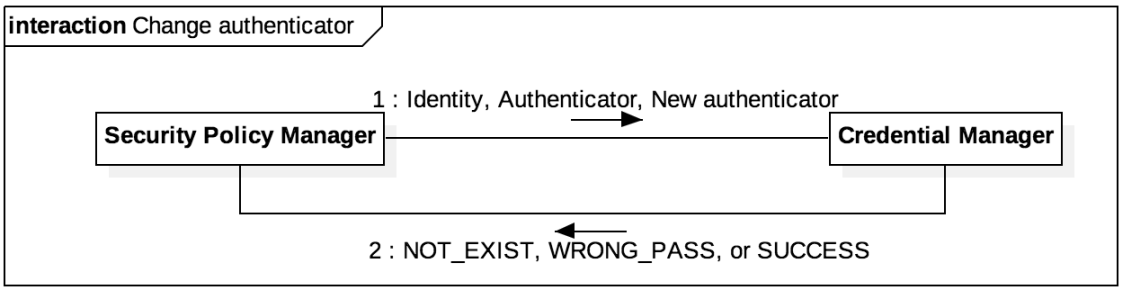


Figure 24 Component interaction for the credential manager in the use case CM-3

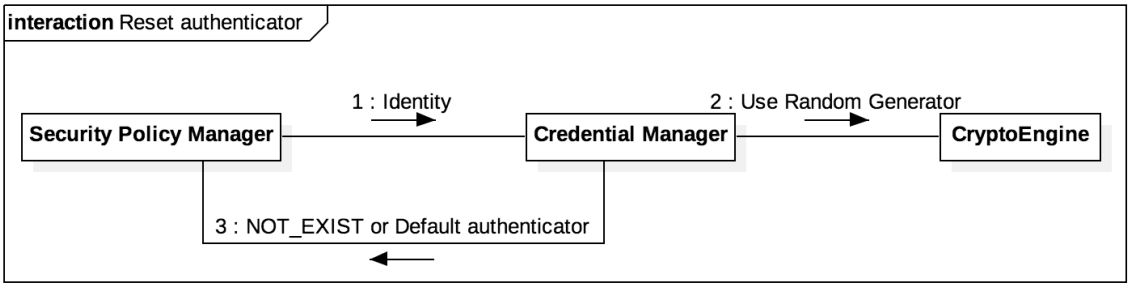


Figure 25 Component interaction for the credential manager in the use case CM-4

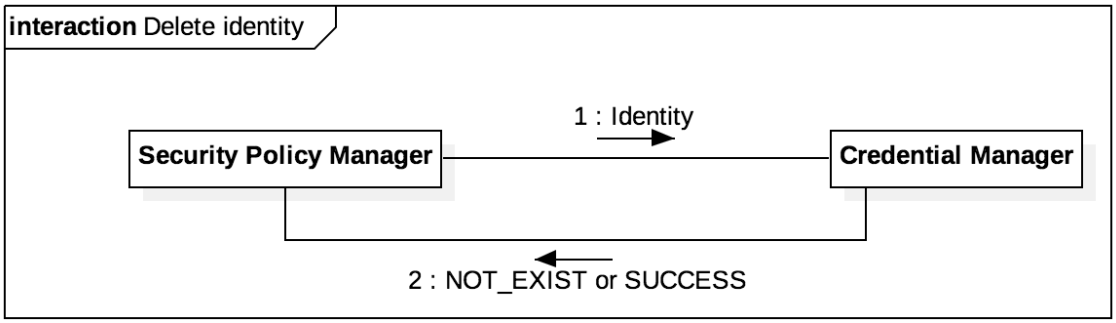


Figure 26 Component interaction for the credential manager in the use case CM-5

4.4.7.3 Database design

As stated, for the first version of security components in MiCADO, the Credential Backend Storage (CBS) is implemented using a text file. However, in order to prepare for any extension in future, we still provide database design for data as belows.

1. **Table Credential**

Description: This table contains data about all users in MiCADO.

Table 11 Credential table

#	Table name	Field name	Type	Primary key	Foreign key	IsNull	Description

1	Credential	Id	Integer	Y	N	N	Identity
2	Credential	Username	String	Y	N	N	Identity
3	Credential	Password	String	N	N	N	Authenticator
4	Credential	CreateDate	Date	N	N	Y	Date of creation
5	Credential	CreateBy	Integer	N	Y	Y	Identity of admin who creates this user
6	Credential	Email	String	N	N	Y	Email of user
7	Credential	Phone	String	N	N	Y	Phone number of user
8	Credential	Role	Byte	N	N	N	0 = user 2 = administrator
9	Credential	State	Byte	N	N	N	0 = active 2 = deleted

2. Table AccessLog

Description: This table contains log information about users's accesses to MiCADO.

Table 12 AccessLog table

#	Table name	Field name	Type	Primary key	Foreign key	IsNull	Description
1	AccessLog	Id	Integer	Y	N	N	Id
2	AccessLog	UserId	Integer	N	Y	N	Identity of user
3	AccessLog	StartTime	Time	N	N	Y	Starting time of the recent continuous log-in attempts Default value = 0
4	AccessLog	NoFails	Integer	N	N	Y	Number of fails from StartTime
5	AccessLog	LockStatus	Byte	N	N	Y	0 = unlocked 2 = locked
6	AccessLog	LockStartTime	Time	N	N	Y	Time when the account is locked Default value = 0
7	AccessLog	IpAddress	String	N	N	N	IP address. This value could be used in future to lock access from an IP.

In order to demonstrate how the table AccessLog is used, we describe a protocol for user authentication with lock-out functionality in case of continuous failed log in with the same identity. This is an extension for the use case CM-1.

Table 13 Protocol for authentication with lock-out functionality

1. User enters credential for log in
2. System checks if LockStatus = LOCKED or UNLOCKED. If LOCKED, go to step 3. Otherwise, go to step 4.
3. Get the current time. If CurrentTime – LockStartTime > DurationForLock, reset LockStatus = unlocked and NoFails = 0. Go to step 4. Otherwise, return BEING_LOCKED.
4. Verify the credential as in the use case CM-1. If return value is ROLE_VALUE, meaning authentication is successful, reset NoFails = 0. Return ROLE_VALUE. Otherwise, go to step 5.

5. If return value is NOT_EXIST, return NOT_EXIST. If return value is WRONG_PASS, go to step 6.
6. Get the current time. If CurrentTime - StartTime < DurationForAttempts (this is queried from the AccessConfig table), increase NoFails by 1 and go to step 6. Otherwise, set NoFails = 1, StartTime = CurrentTime and return WRONG_PASS.
7. Compare NoFails with MaxFails. If NoFails > MaxFails, set LockStatus = locked. Set LockStartTime = CurrentTime. Return LOCKED.

3. Table AccessConfig

Description: This table contains configuration settings related to user's authentication. Only administrator can access and set value in this table. The latest row in the table indicates the latest configuration that is in use.

Table 14 AccessConfig table

#	Table name	Field name	Type	Primary key	Foreign key	IsNull	Description
1	AccessConfig	Id	Integer	Y	N	N	Increasing auto number. The last row contains the updated configuration.
1	AccessConfig	DurationForAttempts	Time	N	N	Y	Duration (in minutes) for counting failed log in attempts. For instance, DurationForAttempts=60 means that during 60 minutes, failed log in attempt will be counted.
2	AccessConfig	MaxFails	Byte	N	N	Y	Maximum of allowed fails in fixed time defined by DurationForAttempts
3	AccessConfig	DurationForLock	Time	N	N	Y	Duration (in minutes) for locking-out a credential
4	AccessConfig	CreatedBy	Id	N	Y	Y	Identity of admin who creates it
5	AccessConfig	CreatedDate	Date	N	N	Y	Date of creation

4.4.7.4 Security requirements traceability

The CM addresses the following requirements outlined in D7.1 COLA security requirements: CNSR-1, CNSR-3.

4.4.7.5 Architecture objectives traceability

The CM addresses the following security architecture objective outlined in D7.2 MiCADO security architecture specification: O4.2, O5.1

4.4.8 Architectural drivers

4.4.8.1 High-Level functional requirements

Authentication: All subscribers, including users and administrators, must be authenticated prior to application deployment/management in the infrastructure.
Secure Credential Backend Storage: CBS is deployed so that only CM can access it. In the scope of MiCADO, CM is deployed as a container in the master node and CBS is implemented using a file. The CBS file, should be contained in a separate volume of CM. Then, only the host (the master node) and CM can access the file.
Confidentiality and integrity of network communication: All network communication between subscribers and MiCADO must be confidentiality and integrity protected.

4.4.8.2 Technical constraints

CM is responsible for verifying a subscriber's authenticator given its credential. In order to complete the authentication process, it is required to have session management and access control implemented. Zorp is responsible for these tasks.

4.4.8.3 Business constraints

No business constraints have been found at this point.

4.4.8.4 API specifications

1. ***Verify authenticator***
    - **d. Input**
        - **i.** Credential <Identity, Authenticator>
    - **e. Output**
        - **i.** Return value NOT_EXIST, WRONG_PASS or ROLE_VALUE
    - **f. Comment** Authenticator verification aims to verify whether the inputted credential matches with any authenticator stored in the backend database or not. This is a simple comparison that returns a binary answer – matched or not matched. If matched, the subscriber is allowed to access to MiCADO services. Otherwise, the subscriber is not allowed to do anything else.
2. ***Add new identity***
    - **a. Input**
        - **i.** <Identity, Role>
    - **b. Output**
        - **i.** Return value EXISTED or DEFAULT_PASS
    - **c. Comment** This API aims to add a completely new identity. This API is only for administrator's usage.
3. ***Change authenticator***
    - **a. Input**
        - **i.** <Identity, Authenticator, New authenticator>
    - **b. Output**
        - **i.** Return value NOT_EXIST or WRONG_PASS or SUCCESS
    - **c. Comment** The authenticator is updated only if the identity and authenticator verification is successful.
4. ***Reset authenticator***
    - **a. Input**

i. <Identity>

b. **Output**

i. Return value NOT_EXIST or DEFAULT_PASS

c. **Comment**

5. ***Specify credential backend storage (extension)***

a. **Input**

i. Backend storage information <Type>

b. **Output**

i. Return value 0 or 1 (0 means failed provisioning, 1 means successful provisioning)

c. **Comment** In case MiCADO supports multiple types of CBS, (e.g., storing credentials in a file/Consul/Credential Store), it can provide the administrator with options to select a specific type of CBS.

4.4.9 Test plan

This test plan is created for the unit level of testing and is under development. Other levels such as integration testing level, system level, and acceptance level are not concerned yet.

1. Test items

Table 15 Credential Manager - Test items

#	Item to Test	Test Description
1	Security Policy Manager	Test whether the component can communicate with CM, and works properly or not
2	Credential Manager	Test whether the component can communicate with SPM, and works properly or not

2. Test features

Table 16 Credential Manager - Test features

#	Function to Test	Test Description
1	Verify authenticator	Test whether the function works properly and returns correct response
2	Add new identity	Test whether the function works properly and returns correct response
3	Change authenticator	Test whether the function works properly and returns correct response
4	Reset authenticator	Test whether the function works properly and returns correct response
5	Delete identity	Test whether the function works properly and returns correct response
5	Integration of #1 and #2	Test whether the two functions corporate smoothly to deliver the function of adding a new identity or not
6	Integration of #1 and #3	Test whether the two functions corporate smoothly to deliver the function of changing authenticator or not
7	Integration of #1 and #4	Test whether the two functions corporate smoothly to deliver the function of resetting authenticator or not

8	Integration of #1 and #5	Test whether the two functions corporate smoothly to deliver the function of deleting an identity or not

3. Features not to be tested

Some features are not tested at this phase because they will be delayed for developing later or they belong to another test phase.

Table 17 Credential Manager - Features not to be tested

#	Feature not to be tested	Test Description
1	Lock-out mechanism	Test whether the function works properly and returns correct response
2	Verifying password strength	Test whether the function works properly and returns correct response
3	Reset authenticator by user himself	Test whether the function works properly and returns correct response
4	Collision of random authenticator	Test whether new random generated authenticator matches with any of other generated ones in the past
5	Forcing user to change the default authenticator	Test whether users changed their default authenticator from the first log-in or not
6	Testing for credentials transported over protected channel	Test whether credentials are transported with POST method through HTTPS protocol or not. This test should involve all sensitive requests, such as log in request, TOSCA file submission.
7	Testing for bypassing authentication	Test whether user can bypass authentication by means such as directing to another page which is not under access control, parameter modification, session Id prediction, SQL injection.
8	Test for non-specific announcement for failed login	Test whether user knows if username or password fails or not.
9	Test for default credentials	Test whether user is using common default credentials or not. For e.g., common usernames are admin, qa, test, root. Common passwords are blank password, pass123, 123, nopass, password.

4. Approach

Table 18 Credential Manager - Test approach

#	Function to Test	Test data description	Metrics to be collected	Pass/Fail criteria
1	Verify authenticator	Data involves not existed identity, existed identity with wrong authenticator, existed identity with matched authenticator	Correct/ Incorrect	Precision = # of incorrect/ # of test runs Pass if precision = 1 Fail if precision<1

2	Add new identity	Data involves not existed identity, existed identity	Correct/ Incorrect	As above
3	Change authenticator	Data involves not existed identity, existed identity with wrong authenticator, existed identity with matched authenticator but empty new authenticator, existed identity with matched authenticator and non-empty new authenticator	Correct/ Incorrect	As above
4	Reset authenticator	Data involves not existed identity, existed identity	Correct/ Incorrect	As above
5	Delete identity	Data involves not existed identity, existed identity	Correct/ Incorrect	As above
6	Integration of #1 and #2	Combination data from #1 and #2	Correct/ Incorrect	As above
7	Integration of #1 and #3	Combination data from #1 and #3	Correct/ Incorrect	As above
8	Integration of #1 and #4	Combination data from #1 and #4	Correct/ Incorrect	As above
9	Integration of #1 and #5	Combination data from #1 and #5	Correct/ Incorrect	As above

4.4.10 Re-utilised Technologies/Specifications

Re-utilized technologies are presented in the table below:

Component	**Role**	**Availability**
Zorp	Access control and token management	Open Source

The utilized components are modified where necessary for the purposes of the enabler.

4.5 Crendential Store: Open specifications

4.5.1 Preface

MiCADO infrastructure itself requires certain private information to run. For instance, the Cloud Orchestrator (CO) requires cloud credential from the user to communicate with the CSP. Without providing valid cloud credential, CSP will not allow CO to request for scale up or down of the cloud resources. In addition to that, CO also requires swarm worker token that is used to configure new worker nodes to join into the swarm.

Apart from that, it is common that applications need to access some private information during runtime. This allows the applications to complete several tasks such as database account, external storage account, API key, SSL certificate, etc. Such private information is not recommended to be hard-coded into the source code, or stored in Docker images of the applications.

Both categories of this sensitive information (i.e. application and infrastructure sensitive information), may be stored inside the MiCADO infrastructure, and Credential Store is built up to take charge of this task.

4.5.1.1 Status

An enabler prototype is under development. This is a preliminary specification and is subject to changes.

4.5.2 Copyright

4.5.3 Legal notice

N/A

4.5.4 Terms and definitions

Docker secret	A piece of data that is encrypted at rest in a Docker swarm and can be securely transmit to swarm services
Hashicorp vault	An open source that provides secure storage and access controls to secret data (https://www.vaultproject.io)

4.5.5 Overview

There is one major difference between application and infrastructure sensitive information: sensitive information provisioning. The infrastructure sensitive information only need to be provisioned to an internal component in the master node; therefore, they will be stored in ***Credential Store***. Meanwhile, the application sensitive information need to be accessed by swarm application services in worker nodes. Swarm services are created based on the user's application that should not acknowledge about any specific deployment of components inside the infrastructure. For such reason, Credential Store aims to mainly store infrastructure sensitive information. For application sensitive information, there will be two options for application developers. For the first option, their sensitive information will be stored as ***Docker secrets*** in the Swarm Manager that are easily accessed by authorized swarm services. The majority of the application developers would know about this mechanism offered by Docker. Thus, they can implement a function inside their application that will give them access to this private information. For the second option, their sensitive information will be stored in the Credential Store of the master node. The developers can then use the provided API to access their private information.

In this section, we mainly describe the Credential Store which will be built on the top of some open source software. Although we have not decided a specific open source for implementation, we rely on Hashicorp Vault [27] to describe basic concepts as well as use cases of this enabler.

In addition to that, we also provide some valuable insights regarding the Docker Secret which may be more appropriate for swarm services to access private information compared to the Credential Store. The reason is that developers may be familiar with Docker and it is easier for them to retrieve secrets from swarm instead of using the APIs provided by MiCADO.

4.5.6 Basic concepts

Storage backend is responsible for storing sensitive information in encrypted form and it is considered as a non-trusted entity. In the first stage of the implementation, the storage backend is implemented using a simple text file.
Server is an instance that provides APIs for clients. Through these APIs, clients can manage secrets.
Client is an instance that uses APIs to interact with the server in order to manage secrets.
Secret is a piece of information that the client requests the server to store securely in the storage backend.
Vault is a tool for storing secrets and allowing securely access to secrets. This tool runs in the server.
Initialization is the process that configures the vault for client use.
Authentication is a way for the server to authenticate a client prior allowing the client to manage secrets.
Client token is granted to the client by the server after successful authentication. It is used for verifying the client's identity for future request without re-authentication.
Root token is generated by the server after initialization. With root token, the client can do anything in the vault.
Keys are generated by the server after initialization. The server will use keys to open the decryption key that helps to decrypt data from the storage backend. However, the server does not store keys. Instead, only client who can access the secrets is able to keep the keys.
Unseal is the process that provides vault with the client's keys so that can successfully access the decryption key.

In MiCADO, Security Policy Manager (SPM) plays the client role and Credential Store does the server role.

4.5.7 Main interactions

4.5.7.1 Use cases

In this section, we describe use cases for the *Credential Store* enabler. The use cases are described in the "fully-dressed" format [29].

Table 19 Use case CS-1: Initialize Credential Store

ID	CS-1
Title	Initialize Credential Store
Description	The Security Policy Manager (SPM) requests to initialize the Credential Store.
Primary Actor	The Credential Store (CS) or the server. The Security Policy Manager (SPM) or the client.
Preconditions	CS and SPM are started as Docker containers in the master node already. CS is configured as a vault server with a file for the storage backend.
Post-condition	The vault in CS is initialized successfully.
Main success scenario	1. SPM sends "init" request to CS 2. CS initializes the vault, return the root token and keys to SPM 3. SPM saves the root token and keys into its filesystem
Extensions	3a. Root tokens and keys may be stored in encrypted form

Frequency of Use	This happens only once when the infrastructure is initialized
Status	Design phase
Owner	UoW

Table 20 Use case CS-2: Read/ write/ remove sensitive information

ID	CS-2
Title	Read/ write/ remove sensitive information
Description	SPM reads sensitive information from/ write sensitive information to/ remove sensitive information from CS
Primary Actor	Security Policy Manager (SPM) Credential Store (CS)
Preconditions	CS has been started and initialized SPM has the keys which will be used to unseal CS
Post-condition	The sensitive information is read to SPM/ written to CS/ removed from CS
Main success scenario	1. SPM uses the keys to unseal CS 2. CS changes the vault status from “sealed“ to “unsealed“ 3. SPM reads/ writes/ removes the sensitive information 4. CS reads or removes/ writes the sensitive information from/ to the backend file 5. SPM seals CS
Extensions	
Frequency of Use	Write sensitive information: one time when the infrastructure is launched for sensitive information such as cloud user credential, swarm worker token; Read sensitive information: multiple times when Cloud Orchestrator sends requests; Remove sensitive information: possibly not supported now.
Status	Design phase
Owner	UoW

Table 21 Use case DS-1: Read Docker secret from swarm

ID	DS-1
Title	Read a secret from swarm
Description	Swarm application services reads a secret from swarm
Primary Actor	Swarm application service (SAS)
Preconditions	The secret is written to swarm SAS is granted right to access the secret SAS is running and knows the secret name
Post-condition	The secret is read to SAS
Main success scenario	1. SAS uses the secret name to open the file contained the secret. This file has been provisioned to SAS as soon as SAS was granted right to access the secret.
Extensions	
Frequency of Use	Possibly once, depending on application
Status	Design phase

Owner	UoW

Table 22 Use case DS-2: Write secret to swarm and grant access right

ID	DS-2
Title	Write a secret to swarm and grant access right to swarm application services
Description	User writes a secret into swarm and grant access right to swarm application services
Primary Actor	Security Policy Manager (SPM) Container Orchestrator (CO)
Preconditions	User writes a secret and swarm application services that will be granted access to that secret into TOSCA file which is submitted to MiCADO. TOSCA submitter parses the secret along with swarm application services names and passes to Security Policy Manager (SPM).
Post-condition	The secret is written to swarm
Main success scenario	1. SPM passes the secret and the swarm application services names to the Container Orchestrator (CO) 2. CO creates the secret in swarm 3. CO adds the secret to the swarm application services based on received names
Extensions	
Frequency of Use	Once
Status	Design phase
Owner	UoW

Table 23 Use case DS-3: Remove Docker secret

ID	DS-3
Title	Remove a secret from swarm
Description	User removes a secret from swarm
Primary Actor	Security Policy Manager (SPM) Container Orchestrator (CO)
Preconditions	
Post-condition	The secret is removed from swarm
Main success scenario	1. SPM passes the secret name to the Container Orchestrator (CO) 2. CO removes the secret from swarm
Extensions	
Frequency of Use	Once
Status	Design phase
Owner	UoW

4.5.7.2 Components and interaction overview

The following figures illustrate the interactions of the Credential Store (CS), Security Policy Manager and Container Orchestrator in order to provide sensitive information storage service for MiCADO.

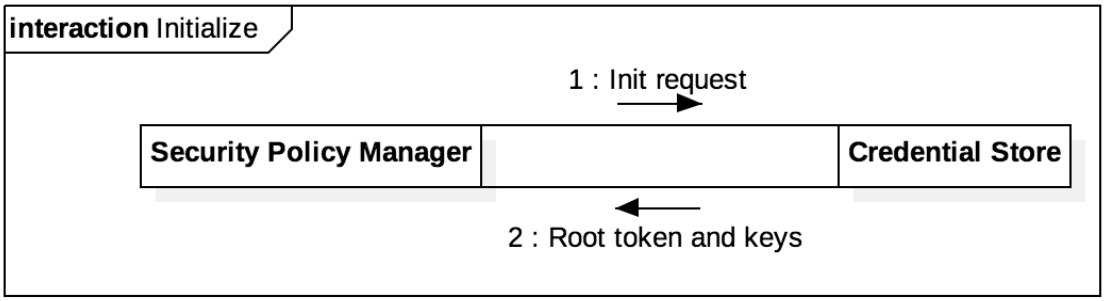


Figure 27 Initialize Credential Store

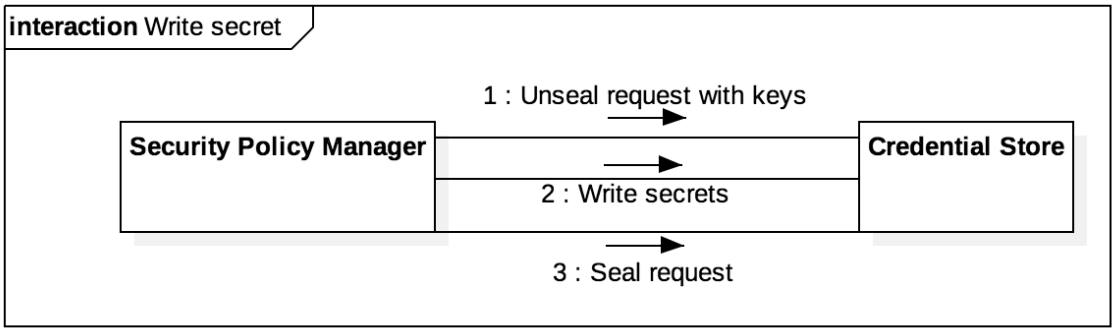


Figure 28 Write sensitive information to Credential Store

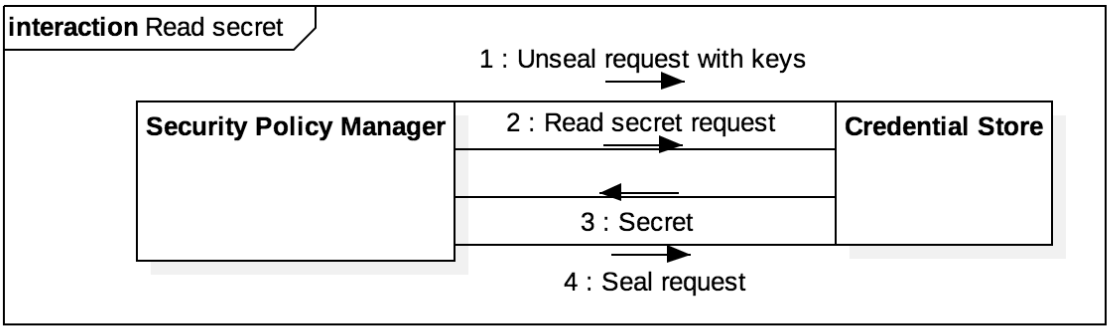


Figure 29 Read sensitive information from Credential Store

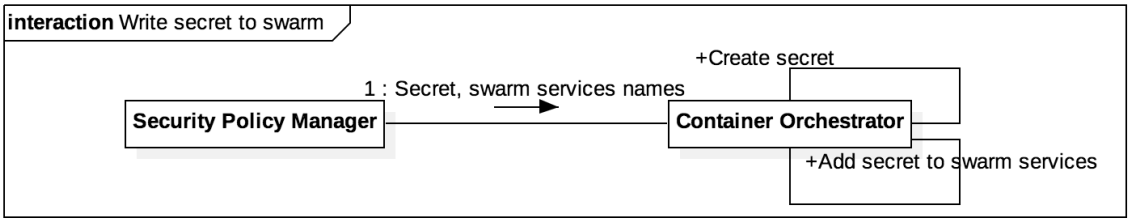


Figure 30 Write sensitive information to swarm

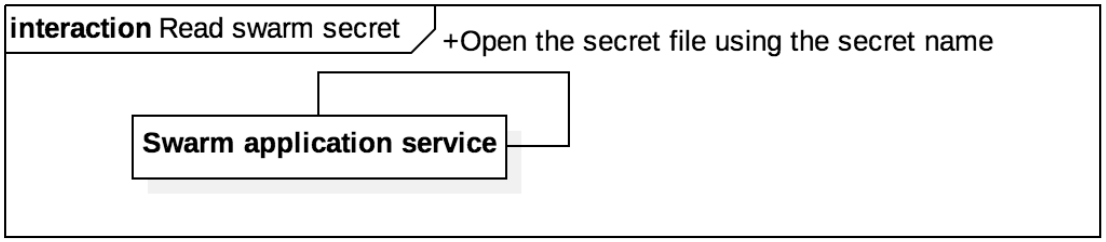


Figure 31 Read Docker secret from swarm

4.5.7.3 Database design

As stated, the Storage Backend is implemented using a text file. However, in order to prepare for any extension in future, we still provide database design for storing sensitive information as belows.

2. Table InfraSecrets

Description: This table contains data about infrastructure sensitive information in MiCADO that are kept secure by the Credential Store.

#	Table name	Field name	Type	Primary key	Foreign key	IsNull	Description
1	InfraSecrets	Id	Integer	Y	N	N	Id
2	InfraSecrets	SecretName	String	N	N	N	Name of secret
3	InfraSecrets	SecretValue	String	N	N	N	Value of secret

3. Table InfraSecretsAccess

Description: This table contains description about access rights to infrastructure sensitive information in MiCADO.

#	Table name	Field name	Type	Primary key	Foreign key	IsNull	Description
1	InfraSecretsAccess	Id	Integer	Y	N	N	Identity
2	InfraSecretsAccess	SecretId	String	N	Y	N	Identity of secret
3	InfraSecretsAccess	AccessRight	Byte	N	N	Y	Bitwise 0 = no components

							2 = Cloud Orchestrator 4 = Container Orchestrator

4. Table AppSecrets

Description: This table contains data about application sensitive information in MiCADO that are kept secure by Docker Swarm or Credential Store.

#	Table name	Field name	Type	Primary key	Foreign key	IsNull	Description
1	AppSecrets	Id	Integer	Y	N	N	Id
2	AppSecrets	SecretName	String	N	N	N	Name of secret
3	AppSecrets	SecretValue	String	N	N	N	Value of secret

5. Table AppSecretsAccess

Description: This table contains description about access rights to application sensitive information in MiCADO.

#	Table name	Field name	Type	Primary key	Foreign key	IsNull	Description
1	AppSecretsAccess	Id	Integer	Y	N	N	Identity
2	AppSecretsAccess	SecretId	String	N	Y	N	Identity of secret
3	AppSecretsAccess	AccessRight	String	N	N	Y	Swarm application service name

4.5.7.4 Security requirements traceability

The CM addresses an extension for the requirements outlined in D7.1 COLA security requirements.

4.5.7.5 Architecture objectives traceability

The CM addresses the following security architecture objective outlined in D7.2 MiCADO security architecture specification: O5.1

4.5.8 Architectural drivers

4.5.8.1 High-Level functional requirements

Secure Storage Backend Sensitive information are stored securely in a file so that only CS can access to it. However, without keys provided by CM, CS cannot decrypt to retrieve the sensitive information.

4.5.8.2 Technical constraints

There are two options for storing application sensitive information: Docker Swarm or Credential Store. For the first option, secrets provisioning is done automatically by swarm server. In the latter, MiCADO must take care of provisioning that is currently out of scope and may be extended later.

4.5.8.3 Business constraints

No business constraints have been found at this point.

4.5.8.4 API specifications

1. ***Initialize Credential Store***
    a. **Input**
        i. URL (both address and port) of Credential Store
    b. **Output**
        i. Root token and keys, or Error
    c. **Comment** Credential Store has been deployed as a Docker container in the master node in advance with a file as storage backend. Initialization aims to generate root token and keys which are granted to SPM. Root token and keys will be stored in filesystem of SPM Docker container.
2. ***Write sensitive information to Credential Store***
    a. **Input**
        i. <URL of Credential Store, List of sensitive information in form of <key, value>>
    b. **Output**
        i. No value is returned
    c. **Comment** SPM uses keys to unseal and then write sensitive information to CS. After that, SPM unseals CS.
3. ***Read sensitive information from Credential Store***
    a. **Input**
        i. <URL of Credential Store, List of sensitive information names>
    b. **Output**
        i. List of sensitive information values
    c. **Comment** SPM uses secret keys to unseal and then read sensitive information from CS using the inputted names. After that, SPM unseals CS.
4. ***Write sensitive information (Docker secret) to Docker Swarm***
    a. **Input**
        i. List of sensitive information and Docker service names which are allowed to access the sensitive information: <key, value, list of Docker service names>.
    b. **Output**
        i. No value is returned
    c. **Comment** SPM communicate with Docker daemon to add sensitive information as Docker secrets into swarm and grant access rights to the corresponding Docker services.
5. ***Read sensitive information (Docker secret) from Docker Swarm***
    a. **Input**
        i. Name of sensitive information
    b. **Output**
        i. Value of sensitive information is returned
    c. **Comment** Docker service uses the name of the information to access its value.

4.5.9 Test plan

This test plan is created for the unit level of testing and is under development. Other levels such as integration testing level, system level, and acceptance level are not concerned yet.

1. Test items

#	Item to Test	Test Description
1	Security Policy Manager (SPM)	Test whether the component can communicate with CO and CS, and works properly or not
2	Credential Store (CS)	Test whether the component can communicate with SPM, and works properly or not
3	Container Orchestrator (CO)	Test whether the component can communicate with SPM, and works properly or not
4	Docker service	Test whether the Docker service in worker node can access the sensitive information which it is granted or not

2. Test features

#	Function to Test	Test Description
1	Initialize Credential Store	Test whether the function works properly and returns correct response
2	Write sensitive information to Credential Store	Test whether the function works properly and returns correct response
3	Read sensitive information from Credential Store	Test whether the function works properly and returns correct response

3. Features not to be tested

Some features are not tested at this phase because they will be delayed for developing later or they belong to another test phase.

#	Function to Test	Test Description
1	Write sensitive information to Docker Swarm	Test whether the function works properly and returns correct response
2	Read sensitive information from Docker Swarm	Test whether the function works properly and returns correct response

4. Approach

#	Function to Test	Test data description	Metrics to be collected	Pass/Fail criteria
1	Initialize Credential Store	Data involves two cases: correct URL of Credential Store, incorrect URL of Credential Store	Correct/ Incorrect ("correct" means that Credential Store is initialized successful if providing URL is correct, and vice versa)	Precision = # of incorrect/ # of test runs Pass if precision = 1 Fail if precision<1
2	Write sensitive information to Credential Store	Data involves cases: no information, one piece of information, multiple pieces of information	Correct/ Incorrect	As above
3	Read sensitive information from Credential Store	Data involves cases: not existed information name, existed information name, and combination.	Correct/ Incorrect	As above
4	Write sensitive information to Docker Swarm	Data involves cases: no information, one piece of information, multiple pieces of information	Correct/ Incorrect	As above
5	Read sensitive information from Docker Swarm	Data involves cases: accessing the sensitive information that the Docker service is allowed to access, accessing the sensitive information that it is not allowed to access	Correct/ Incorrect	As above

4.5.10 Re-utilised Technologies/Specifications

Re-utilized technologies are presented in the table below:

Component	Role	Availability
Vault	Secret store	Open Source
Docker daemon	Docker secret store	Open Source

The utilized components are modified where necessary for the purposes of the enabler.

4.6 Zorp Firewall: Open specifications

4.6.1 Preface

While MiCADO master node can be deployed locally or in cloud, worker nodes are deployed in cloud. Attackers can attack against both master node and worker nodes. Consequently, all shall be protected by restricting access and open ports, that can be done by installing firewalls. Zorp firewall is a piece of open-source software, that can play such role.

4.6.1.1 Status

An enabler prototype is under development. This is a preliminary specification and is subject to changes.

4.6.2 Copyright

4.6.3 Legal notice

N/A

4.6.4 Terms and definitions

TLS	Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network.
CM	Credential Manager, a component that stores user credentials and roles and exposes and authentication interface via a REST API
TOSCA	Topology and Orchestration Specification for Cloud Applications is a specification format that provides a language to describe service components and their relationships using a service topology in a cloud environment.

4.6.5 Overview

Zorp can perform the following tasks on the master node:

- Perimeter network access control;
- Application protocol enforcement;
- TLS offloading;
- Authentication and authorization;
- URL-based routing.

The role of Zorp on the master node is to provide the highest possible level of network security when the user accesses the master node. As the master node contains all management functions, its security is of paramount importance.

Zorp will pre-filter incoming packets using the builtin Linux netfilter infrastructure and then fully process the accepted packets as a proxy. Additional security features include adding an encryption layer (TLS offloading), adding authentication to protocols that support it (e.g.

HTTP) and can also perform authorization based on the URL to be accessed. Based on the URL, Zorp can also forward the request to different microservices in the MiCADO master, for example to the TOSCA submitter, or the Credential Manager user interface. By using Zorp for this task, the security features do not have to be implemented one-by-one in the microservices and also expose less endpoints that can be abused as attacker entry points.

By implementing the 2-factor authentication support, the security of the management user interface could be raised substantially.

By implementing access control delegation (OAuth2 or SAML), the user administration of the master nodes used in enterprise enviroments would be eased considerably. In low-security environments, this also eases deployment as users could utilize a 3rd party provider (e.g. Google) for providing access control information.

4.6.6 Basic concepts

Authentication: All subscribers, including users and administrators, must be authenticated prior to application deployment/management in the infrastructure.
Authorization: All subscribers, including users and administrators, must be granted access to protected resources within the MiCADO architecture only based on their corresponding role, unathorized access MUST be prevented.
URL-based routing: To reduce the number of open ports and provide uniform security features to all user-facing MiCADO microservices, only one graphical management interface should be opened towards all subscribers, the reverse proxy (Zorp in this case) will examine the URL and forward the request to the corresponding microservice of the MiCADO master.
Perimeter network access control: To perform filtering on incoming requests to the MiCADO master node, all incoming traffic is handed to the firewall microservice for examination, except explicitly enabled well-known traffic that is only subject to packet filtering (swarm, etc).
Application protocol enforcement: To prevent exploitation of possible application server programming errors, the formal requirements of all graphical management protocols (HTTP and TLS) must be met, all traffic must compy with their corresponding RFCs, violations must result in termination of the connection.
TLS offloading: To provide a uniform level of transport security, the TLS layer of all traffic to the user-facing microservices of the MiCADO master node must be enforced at the perimeter, only secure versions and ciphers must be allowed for key exchange and negotiation.

4.6.7 Main interactions

4.6.7.1 Use cases

ID	ZM-1
Title	Access MiCADO dashboard
Description	The user initates a web request towards the dashboard component on the MiCADO master node via its URL

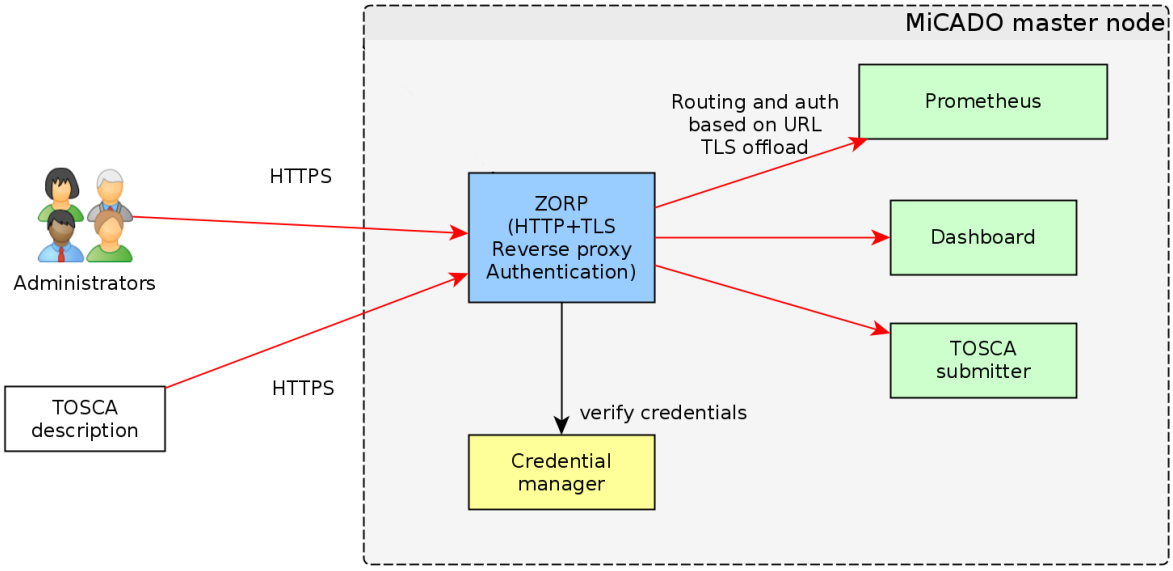
Primary Actor	The user.
Preconditions	The CM and the Dashboard are started as Docker containers on the master node. The user is previously added to the CM with the appropriate authenticator.
Post-condition	The web interface is presented to the authenticated user successfully.
Main success scenario	1. The user sends an HTTPS request to the master node. 2. Zorp presents an authentication form to the user. 3. The user supplies its MiCADO credentials. 4. Zorp initiates a REST call to the CM to verify the user's credentials. 5. The CM confirms user credentials (if correct) and presents the users roles in its answer. 6. Zorp uses its predefined ruleset to determine the final target of the request based on the URL 7. Zorp forwards the initial request to the dashboard microservice 8. Zorp forwards successive calls the the dashboard without futher authentication based on the verification of a Cookie token until timeout occurs.
Extensions	Dashboard may implement a logout link to invalidate the Cookie and terminate the user session.
Frequency of Use	This may happen frequently, whenever the user would like to monitor the status of the MiCADO infrastructure
Status	Design phase
Owner	BalaSys

ID	ZM-2
Title	TOSCA description submission
Description	The user would like to create or change a MiCADO application by initiating web request towards the dashboard component on the MiCADO master node via its URL and submitting a TOSCA descriptor
Primary Actor	The user.
Preconditions	The CM and the TOSCA submitter are started as Docker containers on the master node. The user is previously added to the CM with the appropriate authenticator.
Post-condition	The web interface is presented to the authenticated user successfully.
Main success scenario	1. The user sends an HTTPS request to the master node. 2. Zorp presents a basic authentication request (HTTP response code 401) to the user. 3. The user supplies its MiCADO credentials. 4. Zorp initiates a REST call to the CM to verify the user's credentials. 5. The CM confirms user credentials (if correct) and presents the users roles in its answer. 6. Zorp verifies if the user has the administrator role and permits submission if applicable. 7. Zorp uses its predefined ruleset to determine the final target of the request based on the URL 8. Zorp forwards the initial request to the submitter microservice

Extensions	
Frequency of Use	This may happen infrequently, whenever the user would like to provision or change a MiCADO application
Status	Design phase
Owner	BalaSys

ID	ZM-3
Title	Password change
Description	The user initializes a password change
Primary Actor	The user.
Preconditions	The CM is started as Docker container on the master node. The user is previously added to the CM with the appropriate authenticator. The user has successfully performed authentication.
Post-condition	The user's password is successfully changed.
Main success scenario	1. The authenticated user sends a password change request to the master node. 2. Zorp presents a web page to the user where they have to input their current password and the desired new password and password confimation. 3. The user supplies its old and new MiCADO credentials. 4. Zorp initiates a REST call to the CM to change the user's credentials. 5. The CM confirms the change of the user credentials (if correct). 6. Zorp redirects the user to the login page to re-authenticate with the new credentials.
Extensions	
Frequency of Use	This may happen infrequently, whenever the user would like to provision or change a MiCADO application
Status	Design phase
Owner	BalaSys

4.6.7.2 Components and interaction overview



4.6.7.3 Security requirements traceability

Zorp Firewall addresses the following requirements outlined in D7.1 COLA security requirements: SR05, SR06, SR10, CNSR-1, CNSR-2, CNSR-3, CNSR-4, CNSR-5, CNSR-6, CNSR-7, CNSR-8, CNSR-9, CNSR-10

4.6.7.4 Architecture objectives traceability

The CM addresses the following security architecture objective outlined in D7.2 MiCADO security architecture specification: O1.1, O4.2, O4.3, O4.4, O6.1, O6.2

4.6.8 Architectural drivers

4.6.8.1 High-Level functional requirements

Authentication: All subscribers, including users and administrators, must be authenticated prior to application deployment/management in the infrastructure.
Authorization: All subscribers, including users and administrators, must be granted access to protected resources within the MiCADO architecture only based on their corresponding role, unathorized access MUST be prevented.
URL-based routing: To reduce the number of open ports and provide uniform security features to all user-facing MiCADO microservices, only one graphical management interface should be opened towards all subscribers, the reverse proxy (Zorp in this case) will examine the URL and forward the request to the corresponding microservice of the MiCADO master.
Perimeter network access control: To perform filtering on incoming requests to the MiCADO master node, all incoming traffic is handed to the firewall microservice for examination, except explicitly enabled well-known traffic that is only subject to packet filtering (swarm, etc).
Application protocol enforcement: To prevent exploitation of possible application server programming errors, the formal requirements of all graphical management protocols (HTTP

and TLS) must be met, all traffic must compy with their corresponding RFCs, violations must result in termination of the connection.
TLS offloading: To provide a uniform level of transport security, the TLS layer of all traffic to the user-facing microservices of the MiCADO master node must be enforced at the perimeter, only secure versions and ciphers must be allowed for key exchange and negotiation.
Key provisioning: The keypair for TLS encryption must be supplied via Ansible when provisioning the master node, otherwise the self-signed, auto-generated "snakeoil" keypair is used. Automatic key provisioning is not in scope for the prototype.

4.6.8.2 Technical constraints

No known technical constraints at this time.

4.6.8.3 Business constraints

No business constraints have been found at this point.

4.6.8.4 API specifications

No strict API is described for user interaction, the use cases describe standard procedures. For interaction with the CM, the CM defines the wire format. The programmatic interface in Zorp is defined by the AbstractAuthenticationBackend[4] class.

4.6.9 Test plan

1. Test items

#	Item to Test	Test Description
1	Zorp Firewall	Test whether the component can communicate with CM, and works properly or not
2	Credential Manager	Test whether the component can communicate with Zorp, and works properly or not

2. Test features

#	Function to Test	Test Description
1	Authentication	Test whether the function works properly and returns correct response
2	Change authenticator	Test whether the function works properly and returns correct response
3	User session termination	Test whether the function works properly and returns correct response
4	User access control	Test whether the function works properly and returns correct response
5	URL-based request routing	Test whether the function works properly and returns correct response

[4] https://github.com/Balasys/zorp/blob/master/pylib/Zorp/AuthDB.py

6	Application protocol enforcement	Test whether the function works properly and returns correct response
7	Network access control	Test whether the function works properly and returns correct response

3. Features not to be tested

Some features are not tested at this phase because they will be delayed for developing later or they belong to another test phase.

#	Feature not to be tested	Test Description
1	Automatic TLS keypair provisioning	Test whether the function works properly and returns correct response
2	2-factor authentication	Test whether the function works properly and returns correct response
3	Authentication delegation	Test whether the function works properly and returns correct response
6	Testing for credentials transported over protected channel within MiCADO	Test whether credentials are transported with POST method through HTTPS protocol or not. This test should involve all sensitive requests, such as log in request, TOSCA file submission within the MiCADO master node.
7	Testing for bypassing authentication	Test whether user can bypass authentication by means such as directing to another page which is not under access control, parameter modification, session Id prediction, SQL injection.
8	Test for default credentials	Test whether user is using common default credentials or not. For e.g., common usernames are admin, qa, test, root. Common passwords are blank password, pass123, 123, nopass, password.

4. Approach

#	Function to Test	Test data description	Metrics to be collected	Pass/Fail criteria
1	Verify authenticator	Data involves not existing identity, existing identity with wrong authenticator, existing identity with matched authenticator via HTTP basic authentication	Correct/ Incorrect	Precision = # of incorrect/ # of test runs Pass if precision = 1 Fail if precision<1
2	Change authenticator	Data involves existing identity with wrong authenticator, existing identity with matched authenticator and non-empty new authenticator and non-verified identity	Correct/ Incorrect	As above

		supplying new authenticator		
3	User session termination	Data involves not existing session, existing session	Correct/ Incorrect	As above
4	User access control	Data involves unauthenticated identity with protected resources, existing authenticated identity with permitted resources and existing authenticated identity with prohibited resources	Correct/ Incorrect	As above
5	URL-based request routing	Data involves not existing route, existing route (URL-target server mapping)	Correct/ Incorrect	As above
6	Application protocol enforcement	Data involves compliant and non-RFC compliant HTTP and TLS	Correct/ Incorrect	As above
7	Network access control	Data involves permitted and prohibited traffic via manual testing	Correct/ Incorrect	As above

4.6.10 Re-utilised Technologies/Specifications

Component	Role	Availability
Zorp	Access control and token management	Open Source

The utilized components are modified where necessary for the purposes of the enabler.

4.7 Zorp SSL: Open specifications

4.7.1 Preface

Secure communication within a distributed architecture is a complex task, that requires great flexibility and tight integration with existing components. To be able to secure the communication between the master and worker nodes of MiCADO Zorp is deployed in a specialized way to provide encryption and traffic encapsulation to the master node's components in a seamless way.

4.7.1.1 Status

An enabler prototype is under development. This is a preliminary specification and is subject to changes.

4.7.2 Copyright

4.7.3 Legal notice

N/A

4.7.4 Terms and definitions

TLS	Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network.
CA	In cryptography, a certificate authority or certification authority (CA) is an entity that issues digital certificates.
PKI	A public key infrastructure (PKI) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption. In this case it refers to the corresponding MiCADO component.
SPM	Security Policy Manager, see section 4.3

4.7.5 Overview

Zorp can perform the following tasks on the worker node:

- TLS wrapping;
- Nontransparent proxying;
- Traffic multiplexing;
- Automatic PKI provisioning (e.g. certificate enrollment).

The role of Zorp on the worker node is to provide confidentiality, integrity, and availability of the internal traffic of the MiCADO architecture. It uses TLS as a transport security implementation (hence the outdated name Zorp SSL). It applies encryption to passing traffic on-demand and on-the-fly and ensures mutual authentication of conversing endpoints using mutual TLS authentication via x509 keypairs. Its configuration is static, but the endpoints are registered via the distributed key-value store of Consul, that is already used and updated in the Docker Swarm architecture. Keypairs are generated and distributed automatically during worker node deployment and updated automatically when approaching expiry.

By implementing the OCSP and OCSP stapling functionalities, revocation of a key could take immediate effect as opposed to distributing or pulling revocation lists periodically. This would shorten the timeframe where a successful attacker could impersonate a node.

Zorp SSL Master is the component that resides on the MiCADO master node and serves as a dispatcher to other microservices when trying to connect to the worker nodes. Zorp SSL Worker is its counterpart on the worker node.

4.7.6 Basic concepts

TLS wrapping: The confidentiality, integrity and authenticity of all messages between the MiCADO master node and the MiCADO worker are ensured by adding a layer of transport security (TLS) on top of management traffic and is secured by mutual TLS authentication to an internal Certification Authority.
Nontransparent proxying: To reduce the complexity of the network architecture, a nontransparent HTTP proxying solution is put in place to add structure and encapsulation to the distributed architecture of the MiCADO master-worker systems, with this solution only one port is needed to be opened for collecting diagnostic data while keeping a clean routing architecture.
Automatic PKI provisioning: All newly created worker nodes must be able to acquire TLS keypairs from the master node using a preshared, random-generated secret that is passed on to them during initial provisioning via the Cloud Orchestrator.

4.7.7 Main interactions

4.7.7.1 Use cases

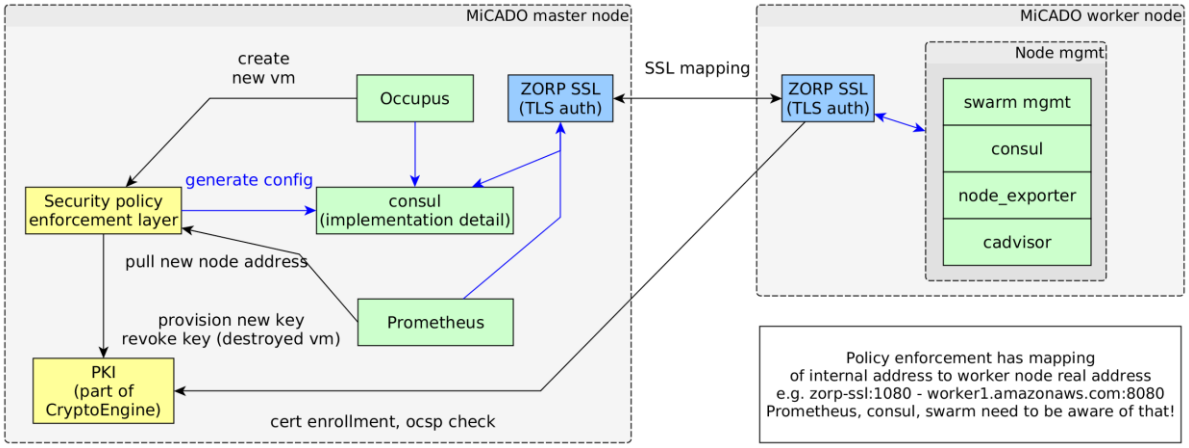
ID	ZS-1
Title	Gather performance data from cAdvisor
Description	The Prometheus component of the MiCADO master initializes a request to gather performance data from the cAdvisor component on the MiCADO worker node
Primary Actor	The Prometheus component.
Preconditions	The worker node is successfully provisioned. Zorp SSL is running on the master node as a Docker container.
Post-condition	The performance metrics are supplied to the Prometheus component successfully.
Main success scenario	1. The Prometheus component initiates a request to the worker node's cAdvisor listening port via Zorp SSL as the proxy server. 2. Zorp SSL Master forwards the request to the appropriate worker node, based on addressing information within the HTTP proxy request. 3. Zorp SSL Master wraps the request in a TLS layer and presents a client certificate to the worker node. 4. Zorp SSL Master verifies the server certificate presented by the worker node to the internal CA. 5. Zorp SSL Worker verifies the client certificate to the internal CA and accepts the connection if applicable. 6. Zorp SSL Worker analyzes the request and forwards the traffic to the cAdvisor component based on its listening port. 7. Both Zorp instances forward the response to the original caller.
Extensions	TLS security can be extended by implementing OCSP lookup and OCSP stapling for online revocation checking.
Frequency of Use	Frequent, Prometheus gathers performance metrics often and periodically to be able to serve as a base for scaling decisions.
Status	Design phase
Owner	BalaSys

ID	ZS-2
Title	Gather performance data from Node_exporter
Description	The Prometheus component of the MiCADO master initializes a request to gather performance data from the Node exporter component on the MiCADO worker node
Primary Actor	The Prometheus component.
Preconditions	The worker node is successfully provisioned. Zorp SSL is running on the master node as a Docker container.
Post-condition	The performance metrics are supplied to the Prometheus component successfully.
Main success scenario	1. The Prometheus component initiates a request to the worker node's cadvisor listening port via Zorp SSL as the proxy server. 2. Zorp SSL Master forwards the request to the appropriate worker node, based on adressing information within the HTTP proxy request. 3. Zorp SSL Master wraps the request in a TLS layer and presents a client certificate to the worker node. 4. Zorp SSL Master verifies the server certificate presented by the worker node to the internal CA. 5. Zorp SSL Worker verifies the client certificate to the internal CA and accepts the connection if applicable. 6. Zorp SSL Worker analyzes the request and forwards the traffic tot he node_exporter component based on its listening port. 7. Both Zorp instances forward the response to the original caller.
Extensions	
Frequency of Use	Frequent, Prometheus gathers performance metrics often and periodically to be able to serve as a base for scaling decisions.
Status	Design phase
Owner	BalaSys

ID	ZS-3
Title	Initialize new worker
Description	The user initates a web request towards the dashboard component on the MiCADO master node via its URL
Primary Actor	Security Policy Manager
Preconditions	The SPM, Cloud Orchestrator, Zorp SSL and PKI components are successfully initialized and running as a Docker container on the MiCADO master node.
Post-condition	The worker node is successfully provisioned, Prometheus is able to gather performance statistics.
Main success scenario	1. The Cloud Orchestrator notifies the SPM that a new worker node is to be provisioned. 2. SPM instructs PKI to generate a new keypair using the internal CA for the newly created worker and assignes a token to the new worker. 3. SPM notifies the Cloud Orchestrator to add the token as a parameter for provisioning the new worker. 4. SPM notifies Zorp SSL Master of the newly created token.

	5. Upon initial startup Zorp SSL Worker connects to Zorp SSL Master using its token to acquire the TLS keypair, saves them locally and starts listening for incoming requests using the new keypair. 6. Zorp SSL Master accepts connection from Zorp SSL Worker, verifies its IP address and token and serves the newly created keypair from PKI. 7. Zorp SSL Master removes the token from its list.
Extensions	
Frequency of Use	This may happen infrequently, whenever the Optimiser component decides to provision a new worker node due the heavy workload of the application.
Status	Design phase
Owner	BalaSys

4.7.7.2 Components and interaction overview



4.7.7.3 Security requirements traceability

Zorp Firewall addresses the following requirements outlined in D7.1 COLA security requirements: SR05, SR06, SR10, CNSR-1, CNSR-2, CNSR-3, CNSR-4, CNSR-5, CNSR-6, CNSR-7, CNSR-8, CNSR-9, CNSR-10

4.7.7.4 Architecture objectives traceability

The CM addresses the following security architecture objective outlined in D7.2 MiCADO security architecture specification: O1.1, O4.2, O4.3, O4.4, O6.1, O6.2

4.7.8 Architectural drivers

4.7.8.1 High-Level functional requirements

TLS wrapping: The confidentiality, integrity and authenticity of all messages between the MiCADO master node and the MiCADO worker are ensured by adding a layer of transport security (TLS) on top of management traffic and is secured by mutual TLS authentication to an internal Certification Authority.

Nontransparent proxying: To reduce the complexity of the network architecture, a nontransparent HTTP proxying solution is put in place to add structure and encapsulation to the distributed architecture of the MiCADO master-worker systems, with this solution only one port is needed to be opened for collecting diagnostic data while keeping a clean routing architecture.

Automatic PKI provisioning: All newly created worker nodes must be able to acquire TLS keypairs from the master node using a preshared, random-generated secret that is passed on to them during initial provisioning via the Cloud Orchestrator.

4.7.8.2 Technical constraints

All services that use the Zorp SSL component for secure master-worker communication must be able to use an HTTP proxy.

4.7.8.3 Business constraints

No known business constraint.

4.7.8.4 API specifications

1. ***Provision new keypair***
    a. **Input**
        i. URL (both address and port) of Zorp SSL Master API, new worker node name and IP address, access token and keypair in key-value format
    b. **Output**
        i. Success, or Error
    c. **Comment** Zorp SSL has been deployed as a Docker container on the master node in advance. SPM has instructed the PKI component to generate the keypair, has assigned a randomly generated access token to the worker. Keys will be temporarily stored in filesystem of the Zorp SSL Master Docker container, until served to the worker.

2. ***Serve new keypair to worker node***
    a. **Input**
        i. Publicly accessible URL (both address and port) of Zorp SSL Master API, new worker node name and IP address, access
    b. **Output**
        i. Keypair or Error
    c. **Comment** Zorp SSL has been deployed as a Docker container in the master node in advance. The Cloud Orchestrator has initiated key provisioning via SPM and provisioning was successful. Zorp SSL Worker connects to Zorp SSL Master via its public URL, presents its access token, Zorp SSL Master verifies the token and the node's source IP address and serves the new keypair. Keys are removed from the filesystem of the Zorp SSL Master Docker container after successful completion.

4.7.9 Test plan

1. Test items

#	Item to Test	Test Description

1	Security Policy Manager (SPM)	Test whether the component can communicate with CO, PKI and Zorp SSL, and works properly or not
2	Public Key Infrastructure (PKI)	Test whether the component can communicate with SPM, and works properly or not
3	Container Orchestrator (CO)	Test whether the component can communicate with SPM, and works properly or not
4	Prometheus	Test whether the component can communicate with Zorp SSL Master, and works properly or not
5	Zorp SSL Master	Test whether the component can communicate with SPM and Zorp SSL Worker, and works properly or not
6	Zorp SSL Worker	Test whether the component can communicate with Zorp SSL Master, and works properly or not

2. Test features

#	Function to Test	Test Description
1	Provision new keypair	Test whether the function works properly and returns correct response
2	Serve new keypair to worker node	Test whether the function works properly and returns correct response
3	Forward request from Prometheus to worker	Test whether the function works properly and returns correct response

3. Approach

#	Function to Test	Test data description	Metrics to be collected	Pass/Fail criteria
1	Provision new keypair	Data involves two cases: correct URL of Credential Store, incorrect URL of Credential Store	Correct/ Incorrect	Precision = # of incorrect/ # of test runs Pass if precision = 1 Fail if precision<1
2	Serve new keypair to worker node	Data involves cases: incorrect IP address, correct IP address, invalid token, valid token	Correct/ Incorrect	As above
3	Forward request from Prometheus to worker	Data involves cases: non-existent worker, existing incorrectly provisioned worker, existing correctly provisioned worker	Correct/ Incorrect	As above

4.7.10 Re-utilised Technologies/Specifications

Component	Role	Availability
Zorp	Access control and token management	Open Source

5 Updated use case partner security requirements

The COLA security architecture is based on a combination of security best-practices adopted by major cloud platforms, as well as on the experience and first-hand needs of the use case partners. Such needs were initially collected, analyzed and distilled into a set of requirements. The security requirements were produced based on the feedback of five end-user organizations (further referred to as *verticals)*: *Outlandish*, *CloudSME* (combining the use cases of HKN and Rheinschafe GmbH), *Saker*, and *INY-SARGA*. The target organizations represent various service domains and business models, which contributes to describing a rich variety of use cases and viewpoints:

- HKN is a German Managed Hosting Company, focusing on building HA clusters for its customers. HKN's customers are normally small and medium sized, German companies.

- Rheinschafe GmbH from Duisburg, Germany is a Digital Agency founded with the main focus on developing websites with TYPO3 and digital communication.

- Outlandish is a 20-person cooperative digital agency specialising in middleware, usability, search and scalable data applications. Outlandish's main focus is on the interface between computers and users in insight-generation and data management. Outlandish have considerable experience building highly usable and intuitive data management solutions.

- Instrumentacion y Componentes S.A. provides high quality services and solutions with added value in IT and Communications, Energy, Laboratory Equipment, Electronics and Medical Equipment.

- Saker Solutions Limited has a mission to expand the benefits achieved from the use of simulation modelling. Saker a provider of simulation-based tools, training, support and consultancy in the UK.

An initial set of requirements has been collected throughout February – April 2017 and distilled into a set of common security requirements for the COLA project, published in Deliverable *D7.1 COLA Security Requirements*.

The evolution of the threat landscape, as well as the introduction of new legislation – such as the General Data Protection Regulation (GDPR) highlighted the need to collect additional and updated feedback. A final set of use case updates were collected in January 2018. The feedback of the use case partners has been compressed into four categories: access control, computation security, data security and compliance. The updates are presented below.

5.1 Instrumentacion y Componentes S.A. (Inycom) Security Requirements

The requirements update provides the following details:

- Access control:
  - The system has only one role, automated service transparent to the end user.
- Computation security:
  - The semantic processing engine is the core business asset;

    - Semantic processing in not supported by the MiCADO functionality;
    - The semantic processing will be *deployed* using the MiCADO framework.
- Data security:
    - Anonymization is explicitly excluded from the scope of the project;
    - Use of personally identifiable information (PII) in use case scenarios;
    - Storage security all low priority, since work is on public.
- Compliance:
    - According to GDPR - must have a registry of collected information;
    - Data must be processed in EU;
    - Once database persistence issue is solved, data must be sent encrypted.

5.2 SAKER Security Requirements

The main scenario for the Saker use case is a private cloud completely disconnected from the Internet. The following specific use case aspects apply:

- Access control:
    - Access control within the internal, air-gapped systems is ensured using Windows authentication;
    - Currently the models are run in the local desktops or out to SakerGrid, that is planned to be supported by MiCADO;
    - There is currently no access control in SakerGrid apart from Windows authentication;
    - Every analyst can access SakerGrid directly.
- Computation security:
    - The models that are deemed security sensitive can only be run on physically separate (air-gapped) infrastructure.
- Data security:
    - No PII is used in the process of creating and running the models;
    - Use of PII is proactively avoided;
    - Databases containing sensitive data are stored on air-gapped networks and servers;
    - Data used for security-sensitive scenarios cannot be stored in public clouds, even encrypted.
- Compliance:
    - Sensitive scenarios related to the core business (such as nuclear power plant evacuation models) are run on physically separate infrastructure. This excludes them from the scope of the COLA project;
    - Scenario models created and run for Government organizations always run on private infrastructure;
    - There are explicitly No specific security compliance requirements for non-government data; most important assets are model configuration parameters and efficiency results.

5.3 Outlandish Security Requirements

Outlandish employs a wide range of security technologies that should be potentially supportable by the MiCADO framework. The requirements update provides the following details:

- Access control:
  - Multiple user roles with varying degree of access and control
  - Audience AGNECY (AA) access to the console; built-in AWS policies and roles currently in use;
  - Support for Ansible roles to set up software requirements (for example NGINX and Node.js) is explicitly assumed;
  - Two access levels (read-access and full access) must be translated to MiCADO access levels, as follows:
    - Role A: submit TOSCA descriptors;
    - Role B: provide console access;
    - Role C: the continuous integration pipeline and machine accounts are a distinct access level.
- Computation security:
  - Currently on AWS - only OUTLANDISH controlling the servers.
- Storage security:
  - Full disk encryption – typically unlocked at boot – may be necessary for particularly sensitive applications that store either PII or business critical information;
  - The use case partner expects the cloud service provider to provide disk encryption.
- Compliance:
  - Use of PII is currently a grey area – the company does not use *personal data*; however, the collected data could be considered as PII;
  - Future development may include more PII-able data.

6 Summary and Conclusions

The scope of this deliverable was fourfold. First of all, by providing a detailed security analysis of MiCADO's core architecture we gave valuable insights regarding the overall security of the system. This analysis allowed us to identify a set of basic security requirements for the infrastructure. These requirements are summarized below:

- Protecting the communication between a user and the infrastructure;
- Protecting the communication between virtual machines in the infrastructure;
- Protecting the communication between a machine in the infrastructure and any external entity.

Secondly, we identified several threat surfaces based on MiCADO's infrastructure. By analyzing these threat surfaces we presented a concrete list of threat models with specific possible attacks that can be performed. Moreover, for each of the described attack vector, we presented possible counter measures such as:

- Using TLS/SSL to secure communication;
- Using captcha and/or lock-out account mechanism to hinder user impersonation attacks;
- Using emails for reset password functions;
- Using HSTS protocol, i.e. HTTP Strict Transport Security;
- Using email notification to alert about unexpected increase in cloud resource usages;
- Providing sensitive information storage;
- Using firewall.

Thirdly, we described the security requirements collected from the use case partners. Such requirements were mainly extracted from the specific needs of the pilots based on their applications. Such requirements include:

- Protecting personally identifiable information (PII);
- Protecting data in transit;
- Providing full disk encryption.

Finally, based on the security enablers/ components described in D7.2 we provided a list of countermeasures for the infrastructure against several possible attacks. As a result, in this document, we gave detailed specifications for all the identified security enablers:

- Image Integrity Verifier to verify container image;
- CryptoEngine to provide cryptographic functions;

- Security Policy Manager to provide central management for security components;
- Credential Manager to provide authentication and credentials storage that help to hinder user impersonation attacks;
- Credential Store to provide sensitive information storage;
- Zorp to provide firewall and TLS/SSL.

Based on the conducted security analysis as well as on the open specifications for security enablers that was presented in this deliverable, we plan to further describe how these enablers are coordinated with core components of MiCADO in order to deliver security enforcement. This work will take place in deliverable D7.4.

7 References

[1] D6.2 – Prototype and documentation of the monitoring service.

[2] D5.4 – First Set of Templates and Services of Use Cases.

[3] Krawczyk, Hugo, Ran Canetti, and Mihir Bellare. "HMAC: Keyed-hashing for message authentication." (1997).

[4] https://docs.docker.com/engine/swarm/how-swarm-mode-works/pki/#rotating-the-ca-certificate, last accessed on 5 Feb, 2018

[5] Kaliski, Burt. "PKCS# 5: Password-based cryptography specification version 2.0." (2000).

[6] Paul Grassi et al. Digital identity guidelines. NIST Special Publication, 800-63, 2017.

[7] Gruschka, Nils, and Meiko Jensen. "Attack surfaces: A taxonomy for attacks on cloud services." *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. IEEE, 2010.

[8] Gelernter, Nethanel, et al. "The password reset mitm attack." *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017.

[9] Antonis Michalas and Ryan Murray. "Keep Pies Away from Kids: A Raspberry Pi Attacking Tool". Proceedings of the 1st ACM CCS International Workshop on Internet of Things Security and Privacy (IoT S&P'17) in Conjunction with ACM CCS 2017, Dallas, USA, October 30 – November 03, 2017.

[10] Antonis Michalas, Nikos Komninos and Neeli R. Prasad. "Multiplayer Game for DDoS Attacks Resilience in Ad hoc Networks". Proceedings of the 2nd IEEE International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless Vitae 2011), Chennai, India, 2011.

[11] Antonis Michalas, N. Komninos, Neeli R. Prasad and Vladimir A. Oleshchuk. "New Client Puzzle Approach for DoS Resistance in Ad hoc Networks". Proceedings of the IEEE International Conference on Information Theory and Information Security (ICITIS'10), Beijing, China, 2010.

[12] Antonis Michalas, Nikos Komninos and Neeli R. Prasad. "Cryptographic Puzzles and Game Theory against DoS and DDoS attacks in Networks". Encryption: Methods, Software and Security", Nova Science Publishers, 2011.

[13] Antonis Michalas, Vladimir A. Oleshchuk, Nikos Komninos and Neeli R. Prasad. "Privacy preserving Trust Establishment scheme for Mobile Ad Hoc Networks". Proceedings of the 16th IEEE International Conference on Communications (ISCC'11), Corfu, Greece, 2011.

[14] Tassos Dimitriou and Antonis Michalas. "Multi-Party Trust Computation in Decentralized Environments". Proceedings of the 5th IFIP International Conference on New Technologies, Mobility & Security (NTMS'12), Istanbul, Turkey, 2012.

[15] Antonis Michalas and Nikos Komninos. "The Lord of the Sense: A Privacy Preserving Reputation System for Participatory Sensing Applications". Proceedings of the 19th IEEE International Conference on Communications (ISCC'14), Madeira, Portugal, 2014.

[16] Tassos Dimitriou and Antonis Michalas. "Multi-Party Trust Computation in Decentralized Environments in the Presence of Malicious Adversaries". Ad Hoc Networks Journal, a special issue on "Smart Solutions for Mobility Supported Distributed and Embedded Systems", Elsevier,

[17] Kassaye Yitbarek Yigzaw, Antonis Michalas and Johan Gustav Bellika. "Secure and Scalable Deduplication of Horizontally Partitioned Health Data for Privacy-

Preserving Distributed Statistical Computation". Journal of Medical Informatics and Decision Making (BMC), 2017.

[18] Rafael Dowsley, Antonis Michalas, Matthias Nagel and Nicolae Paladi. "A Survey on Design and Implementation of Protected Searchable Data in the Cloud". Journal of Computer Science Review, Elsevier, 2017.

[19] Klein, Daniel V. "Foiling the cracker: A survey of, and improvements to, password security." Proceedings of the 2nd USENIX Security Workshop. 1990.

[20] Ding, Yun, and Patrick Horster. "Undetectable on-line password guessing attacks." *ACM SIGOPS Operating Systems Review* 29.4 (1995): 77-86.

[21] Paul Grassi et al. Digital identity guidelines – Authentication and Lifecycle Management. NIST Special Publication, 800-63B, June 2017. https://pages.nist.gov/800-63-3/sp800-63b.html#sec5

[22] Callegati, Franco, Walter Cerroni, and Marco Ramilli. "Man-in-the-Middle Attack to the HTTPS Protocol." *IEEE Security & Privacy* 7.1 (2009): 78-81.

[23] Duane Peifer. "SSL spoofing. Man-in-the-middle attack on SSL". Owasp presentation. https://www.owasp.org/images/7/7a/SSL_Spoofing.pdf

[24] Hodges, Jeff, Collin Jackson, and Adam Barth. *Http strict transport security (hsts)*. No. RFC 6797. 2012.

[25] Open Web Application Security Project, *Session Management Cheat Sheet*, .

[26] Open Web Application Security Project, REST Security Cheat Sheet, https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

[27] https://www.vaultproject.io, last accessed on May 3, 2018

[28] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, no. 2, pp. 63–81, 2011.

[29] Cockburn, Alistair. "Writing effective use cases." *Addison-Wesley, 2001*. ISBN 9780201702255

[30] Paladi, N., & Gehrmann, C. (2016). TruSDN: Bootstrapping Trust in Cloud Network Infrastructure. 12th EAI International Conference on Security and Privacy in Communication Networks.

[31] Paladi, Nicolae and Karlsson, Linus. 2017. Safeguarding VNF Credentials with Intel SGX. In Proceedings of the SIGCOMM Posters and Demos (SIGCOMM Posters and Demos '17). ACM, New York, NY, USA, 144-146.

[32] Paladi, Nicolae, Linus Karlsson, and Khalid Elbashir. "Trust Anchors in Software Defined Networks." *European Symposium on Research in Computer Security*. Springer, Cham, 2018 (in press).