Towards an Open, Secure, Decentralized and Coordinated Fog-to-Cloud Management Ecosystem

D4.2 Security and Privacy aspects for the mF2C Platform Manager block (IT-2)

Project Number	**730929**
Start Date	**01/01/2017**
Duration	**36 months**
Topic	**ICT-06-2016 - Cloud Computing**

Work Package	**WP4, mF2C Platform Manager block design and implementation**
Due Date:	*M25*
Submission Date:	*01/02/2019*
Version:	*1.0*
Status	*Final*
Author(s):	*Jens Jensen (STFC)*
Reviewer(s)	*Román Sosa (ATOS)* *Xavi Masip (UPC)*

Keywords
Web services security, IoT security

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

This document is issues within the frame and for the purpose of the mF2C project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 730929. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	*18/10/2018*	*Initial outline uploaded*	*Jens Jensen*
0.2	*11/01/2019*	*DataClay security section finished*	*Alex Barceló (BSC)*
0.3	*15/01/2019*	*Overall contribution*	*Shirley Crompton (STFC-UKRI)*
0.4	*18/01/2019*	*Machine Learning for Security content*	*Jasenka Dizdarevic (TUBS)*
0.5	*24/01/2019*	*SLA Security content*	*Román Sosa (ATOS)*
0.6	*25/01/2019*	*Telemetry&Monitoring, Landscaper and Analytics content*	*Giuliana Carullo (INTEL)*
0.7	*25/01/2019*	*Monitoring content*	*Alexander J. Leckey (INTEL)*
0.8	*26/01/2019*	*Ready for internal review*	*Jens Jensen (STFC)*
0.9	*30/01/2019*	*Addressed ATOS and UPC reviews*	*Jens Jensen (STFC)*
1.0	*01/02/2019*	*Quality Check. Deliverable ready for submission*	*María Teresa García (ATOS)*

Table of Contents

Version History 3

Table of Contents 4

List of figures 5

List of tables 5

Executive Summary 6

1. Introduction 7

1.1. Purpose 7

1.2. Structure of the document 7

1.3. Glossary of Acronyms 7

2. PM Security in IT-1 9

2.1. Review of PM Security Implementation in IT-1 9

2.2. Gaps identified in IT-1 and tasks postponed to IT-2 9

3. Security Technologies for mF2C Platform Manager Components – IT-2 10

3.1. REST Web Services 11

3.1.1. Network – IP layer 11

3.1.2. Transport layer - Socket 11

3.1.3. Transport layer - HTTP 11

3.1.4. Message layer Security 12

3.1.5. API key 12

3.2. Deployment 12

3.2.1. Container-based deployment 12

3.2.2. Non-container deployment 13

3.2.3. App deployment 13

4. Platform Manager Security 13

4.1. CIMI Security 14

4.1.1. Securing CIMI 14

4.1.2. Authentication and Authorisation 14

4.1.3. Securing Communications between Agents 15

4.2. DataClay Security 17

4.2.1. Data Security 17

4.3. Networking, Protocols and Firewalls 17

4.3.1. Initial Fog to Cloud communication - CAU 18

4.3.2. Fog to Cloud Communication of Registered Agents 18

4.3.3. Fog to Cloud Communication for Applications 18

4.3.4. Firewalls 19
4.4. Service Orchestration 19
4.4.1. SLA Security 19
4.5. Distributed Execution Runtime 20
4.5.1. COMPSs 20
4.6. Telemetry and Monitoring 20
4.6.1. Landscaper and Analytics Engine 20
4.6.2. Monitoring 21
4.6.3. Machine Learning for Security 21
5. Conclusions 23
References 25
Annex 1: At-Rest Encryption 27
Background 27
Key Management 27
Escrow 27
Key Strength 27
Implementation 28

List of figures

Figure 1 Updated architecture of a full mF2C Agent (see D2.7) 10
Figure 2 Inter-Agent communication use cases 15
Figure 3 Authentication and authorisation process flow for inter-agent communication 16

List of tables

Table 1. Acronyms 8
Table 2. JWT standard claims 15
Table 3. Summary of action items 23

Executive Summary

The Platform Manager (PM) is the seat of the functionality of the mF2C agent: it is here that services are categorised, orchestrated, and managed. It is here that tasks are scheduled, executed, monitored, and the metadata in turn used for optimisation and Quality of Service management. However, there are security concerns around these functionalities: for example, if payments are made or resources are traded as in the Smart Boats Use Case, it is necessary that the end user can trust the data in the system. The foundations of mF2C fog-to-cloud security are described in the associated deliverable for the Agent Controller, D3.2. PM security builds upon this, but also applies other standards on top of it, for example standards for authorisation. The core task of PM security is to implement security for REST web services: unlike SOAP, these do not come with any security other than those offered by the HTTP protocol used to implement REST, and while HTTP can provide the basic security features of SOAP - authentication, basic confidentiality through an encrypted socket - for any more sophisticated feature, we will have to implement those ourselves - the options for the implementation are described in this document.

IT-2 is the last software release of the project, so it is this release that must prove the worth of the entire project. This release must have learnt from the experiences with IT-1 and improve upon them. At the same time, we cannot do everything: implementing all the recommendations would add years to the project. The purpose of this deliverable is to clarify the implementation options for further prioritisation. The essentials that must be addressed are the deployment, web services security, the COMPSs task execution framework, DataClay, and CIMI, all of which have their implementation options and plans described in this deliverable (other essentials, such as user privacy and the GDPR are described in D3.2). Out of the need to implement these essentials come the main tasks for the project over the coming six months.

1. Introduction

1.1. Purpose

This deliverable provides a description of the technology and plans for the implementation of the Platform Manager security in IT-2, building on requirements defined or updated in D2.5 [D2.5]. The deliverable D3.2 provides the same description for the Agent Controller. Other relevant deliverables are D2.5 ([D2.5], Security and Privacy Requirements for IT-2) and D2.7 (Architecture for IT-2). In order to avoid overlapping with D2.5 [D2.5] and D2.7, D3.2 and this deliverable, D4.2, focus on the proposed implementation. Action items focus on things that should be investigated and resolved in IT-2 (as opposed to open research questions) and are identified throughout as “PMx” where x is a number, and which are summarised in the conclusion.

To be clear, the earlier deliverables D3.1 [D3.1] and D4.1 [D4.1] remain valid, so we do not repeat information here that was covered in those two deliverables.

1.2. Structure of the document

The document is structured into 5 sections:

- **Section 1** (this section) is the introduction.
- **Section 2** provides a brief overview of the status of PM security in IT-1.
- **Section 3** describes our background technologies, such as web services security, that underpin the proposed implementation of the PM security.
- **Section 4** describes the proposed PM security implementation.
- **Section 5** provides the conclusion, which also contains a summary of the action points arising from this deliverable.

1.3. Glossary of Acronyms

Acronym	Definition
AC	Agent Controller
API	Application Programming Interface
CAU	Control Area Unit
DDoS	Distributed Denial of Service
DNS	Domain Name System
ECP	Enhanced Client Protocol (SAML)
GPG	GNU Privacy Guard
gRPC	Google Remote Procedure Call
HTTP	Hypertext Transfer Protocol
JWT	JSON Web Token (RFC 7519)
IETF	Internet Engineering Task Force
IdP	Identity Provider
IoT	Internet of Things
IP	Internet Protocol (IETF), the lowest layer protocol in Internet communications.
IT	ITeration (in IT-x where x is 1 or 2)
LCM	Lifecycle Manager
MITM	Man in the Middle (attack)
ML	Machine Learning
M2M	Machine-to-Machine (communication)
NIO	Non-blocking Input/Output (Java)
OIDC	OpenID Connect
PKI	Public Key Infrastructure
PM	Platform Manager
PSK	Pre-Shared Key (Wi-Fi)

REST / ReST	Representational State Transfer (web services)
RFC	Request for Comments (IETF)
SAML	Security Assertion Markup Language
SLA	Service Level Agreement
SOAP	(originally) Simple Object Access Protocol (web services)
SSID	Service Set Identifier (Wi-Fi)
SSO	Single Sign-On
TCP	Transmission Control Protocol, the protocol layer above IP (q.v.)
TLS	Transport Layer Security
TSDB	Timeseries database

Table 1. Acronyms

2. PM Security in IT-1

As described in section 2.1 of D2.5, a number of security components were developed for IT-1, and some, but not all, were integrated into the actual IT-1 release. This section briefly recapitulates the status of those components that fall under the scope of the PM, and briefly discusses the limitations identified in IT-1.

2.1. Review of PM Security Implementation in IT-1

It seems that most of the security components developed for IT-1 (whether integrated or not) fall under the category of the AC and are described in D3.2, section 2.1. The main development to highlight is that the agent-to-agent communications (using the PKI to authenticate each other) can now be subsumed into CIMI, a development which will be described in this deliverable.

2.2. Gaps identified in IT-1 and tasks postponed to IT-2

1. COMPSs security - COMPSs was originally developed for a HPC environment so had no native security; however, a GSSAPI-based security implementation allowing remote access and remote communication was developed prior to the start of mF2C. We discuss COMPSs in section 4.5.1.
2. Web services security (e.g. for CIMI) were considered out of scope for IT-1 but can of course no longer be considered out of scope - sections 3.1 and 4.1.1.
3. Nothing was done for authorisation of authenticated clients. We are proposing to improve on that for IT-2; see section 4.1.2.

3. Security Technologies for mF2C Platform Manager Components – IT-2

This section describes some of the technologies available for implementing PM security, including access control, in IT-2. Other related technologies are described in D3.2, namely, those proposed primarily for the AC. In general, the aim is to use open standards with interoperable implementations, as opposed to proprietary or ad-hoc solutions and de-facto "standards."

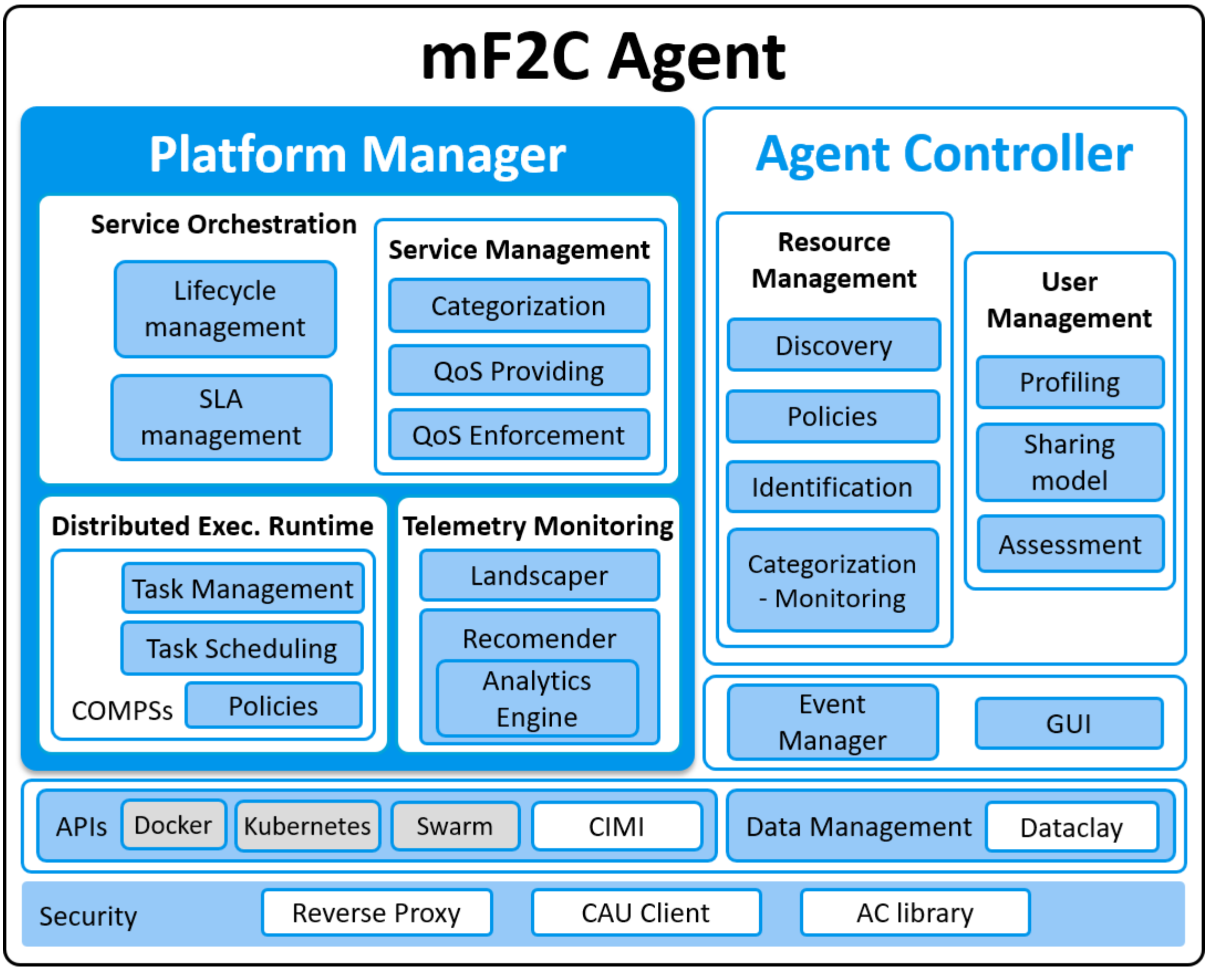


Figure 1 Updated architecture of a full mF2C Agent (see D2.7)

Figure 1 shows the updated architecture of a full agent. In IT-1 an Agent was deployed as a collection of Docker containers that communicate over a private Docker network using CIMI as a single interface brokering web service requests in REST to different blocks within an Agent. In IT-2, Security is refactored as a concrete module offering cross cutting services to both the PM and AC. Given CIMI's role as the single-entry point to an mF2C agent, securing CIMI is crucial to securing mF2C. Intra-agent communications that flow over the private Docker network (as in IT-1, on the same host) are considered sufficiently secure, so our focus is to secure communication between Agents or between an Agent and third-party components, e.g. an external payment service. Apart from CIMI-brokered communications, we also cover those technologies relevant to securing individual PM blocks as well as DataClay which offers data management functionalities to both PM and AC.

3.1. REST Web Services

We start with security best practices relevant to REST, the web services design principle used to design and implement web services for CIMI. Unlike SOAP, the lightweight REST web services do not have a built-in method for doing security (nor for error handling), the conventional approach to REST security is to implement security at the Physical, Network, Socket, HTTP, and/or Message layer (or a combination of these). The Physical layer was suggested as a security mechanism for edge devices that were not capable of implementing the security controls themselves, cf. D3.1, and need not be repeated here.

Another reason for covering security principles for REST in some depth is that they apply also to gRPC. gRPC is needed for both DataClay (section 4.2) and Monitoring (section 4.6.2.).

3.1.1.Network – IP layer

At the network layer, a very basic security mechanism is to filter by IP address, e.g. configure the server hosts to blacklist clients with specific IP addresses, or whitelist addresses of trusted clients. More sophisticated security features are available in IPsec, DNSsec, and IPv6.

3.1.2.Transport layer - Socket

At the transport layer, security mechanisms are available through Generic Security Services API (GSSAPI) and transport layer security (TLS). At the simplest level, one could just do client authentication (i.e. the server MUST always authenticate, and the client MUST verify the server's identity, cf. RFC 2818, section 3.1), and look up the client name in an access control list. More sophisticated approaches would have the protocol carry authorisation attributes (usually in addition to authentication) and make access control decisions based on those attributes. See for example RFCs 3281 and 3820 (for X.509) and RFC 6680 (for GSSAPI).

3.1.3.Transport layer - HTTP

At higher layers, there are essentially two options for security: one is implemented through the security protocol (SAML, OIDC), and the other token-based (OAuth2, JWT). Moreover, particularly if the initial security evaluation is expensive (in networking or computational terms), it makes sense to temporarily cache the security state in a session cookie. We cover these three options in turn.

3.1.3.1. Security Protocol

The aim of the protocol approach is similar to the earlier approach (section 3.1.3): make attributes available and process those for authorisation decisions. Note that for SAML there must be an IdP or proxy issuing the assertion, and the assertion needs to be conveyed to the resource by some means, defined in a SAML profile (such as WebSSO or ECP [SAMLP]), or Moonshot [RFC 7833]. For OIDC [OIDC], the userinfo API is used; again there must be a server to call out to (to which the user has authenticated), and there must be a session reference to refer to the user.

3.1.3.2. Session - Token-based

There are several ways of doing token-based security. The two most important standards are OAuth2 (RFC 6749) bearer token (6750) and JWT (7519). Tokens are issued by an authority and are sent to the server in the HTTP header (RFC 7235). In the case of OAuth, the resource MUST call out to the authority to validate the token, whereas with JSON Web Token (JWT), the token can carry its own validation (claims, scope, digital signatures).

The advantage of the OAuth is that it has several flows to suit different use cases, but most of these flows need the user to be present to authorise the delegation and this usage is not appropriate for M2M interactions. Clients (the entities to whom the rights are delegated) are identified using symmetric keys, i.e. shared secrets. The bearer token form (RFC 6750) requires no client key to validate the token, but the server needs to call out to an authorisation server to check that the token is valid for the proposed action.

The disadvantages of OAuth2 - user presence in most flows, the need for validation in an authorisation server, and the need for shared client secrets, when mF2C agents already have asymmetric keys, mean OAuth2 is not a good match. Instead, we propose to use JWT for authorisations when token-based security is required. Section 4.1.2 outlines the proposed use of JWT for authorising access to CIMI resources.

3.1.3.3. Session Cookies

At the HTTP layer, a session cookie is often used to maintain state between web requests (as well as traditional, even though this violates the primary principle of statelessness in REST. A stateless self-contained access token with the required client information such as JWT (see Section 3.1.5) is preferred over session cookies for the initial authorisation; session cookies should be used for continuity or session restarts once the initial authorisation is granted.

3.1.4.Message layer Security

The final option for implementing REST security is to secure the payload (e.g. in a POST). While it would not in general be wise to ignore security in the layers below - an attacker could inspect or modify the HTTP header - message layer security could be implemented to, for example, hide parts of the message from the recipient, thus providing some of the more sophisticated security features provided by SOAP (the recipient might relay this part of the message onward to a final recipient).

3.1.5.API key

On a few occasions we mention the "api-key" method of authenticating REST web services, mainly because CIMI supports it. This is a method used by some commercial cloud service providers, where the client shares a secret - i.e. a symmetric key - with the service and uses the secret to sign the HTTP header. It is, however, not a standard, so is considered out of scope for mF2C; also, the problem with distribution and maintenance of symmetric keys would impose an extra burden on the project.

3.2. Deployment

This section describes the technology options for deployment of the mF2C platform. The security risks associated with deployment were identified in D2.5, section 3.3. In particular we need to highlight any changes from practices in IT-1. These items are not necessarily specific to the PM, but are included here partly for historical reasons, and partly because some PM components raise issues regarding deployment.

3.2.1.Container-based deployment

During IT-2 we will support not only Docker environments, but also Docker swarm and Kubernetes. This means that we will be able to deploy and run mF2C service instances in this kind of clusters. This approach should help mitigate the risks identified in D2.5 [D2.5], section 3.3.3.

Section 3.3.2 of D2.5 [D2.5] describes additional security options for containers and the images upon which they are built. These are not usually considered essential - unsigned images are the norm rather than the exception - and in practice there is no great security risks as long as we deploy our own images. However, as described in section 4.6.2, Monitoring has identified a requirement for signed images. Signed images would also make sense for the trust anchor deployment: we recommend that trust anchors be deployed in a separate image, which can be used to build mF2C containers, but can easily be replaced by another trust anchors if someone else deploys mF2C.

Shared secrets cannot, however, be put into images. It will be up to post-deployment configuration to ensure that the few remaining cases where we could not avoid shared secrets be configured correctly (such as the password for the metrics database, section 4.6.2). It would also make sense for the project to eliminate the need for shared secrets altogether.

PM1: Implement container signatures.

PM2: Eliminate the use of shared secrets.

PM3: Use container images for trust anchor deployment.

3.2.2. Non-container deployment

As most use cases still have a need for the Raspberry PI in IT-2, it is necessary to have a deployment model that supports this. One of the experiences with IT-1 was that a standard Raspberry PI3 did not have enough memory (1 gigabyte) to support deployment of the mF2C containers. However, Raspberry PI is expected to be indirectly supported through the adoption of NuvlaBox Nano as the micro-Agent. The Microagents (NuvlaBox Nano) will deploy the mF2C app through Nuvla.

PM4: support deployment on Raspberry PI.

3.2.3. App deployment

Although not part of the PM as such, IoT/edge apps designed for mobile phones need to be built and made available to the users (in Use Case 2 and 3) through app stores. The principal concern raised here is the need to minimise the post-deployment configuration needed for end users of mF2C, such as in the Smart Boats or Smart Hub use cases, where users deploy apps on their mobile phones. The more the users need to configure security parameters after they have installed the app, the more likely it is that they misconfigure something.

The app store ensures the integrity of the app, but not necessarily the confidentiality of the configuration information.

The app needs to be deployed with:

- The trust anchors of the mF2C PKIs and any app-specific PKI
- The IP addresses of any (local or, if applicable, cloud) Control Area Unit (CAU(s))
- The SSID and any other configuration requirement for Wi-Fi connection (e.g. PSK), see section

PM5: deploy apps with as much information as is needed to securely connect to mF2C, bearing in mind PM2 (avoiding shared secrets).

4. Platform Manager Security

Referring to Figure 1 again, we shall now go through the PM components, identifying the relevant security technologies. As mentioned in D2.5 [D2.5], section 3.2, some components can be secured by being deployed together on the same host, relying on the Docker private network for security, so we need not cover those here. Moreover, as mentioned in D2.7, section 2.5.3, much of the communications have been subsumed into CIMI, so we cover CIMI security in some detail (section 4.1.1 and 4.1.2), and the communications via CIMI in 4.1.3. (Both CIMI and DataClay have traditionally been documented in WP4, so we continue this tradition).

4.1. CIMI Security

As outlined in section 2, securing CIMI is crucial to securing mF2C. CIMI is a REST service application running on a host server that secures communication using the HTTPS protocol.

4.1.1.Securing CIMI

The proposal is to secure CIMI through remote access, with limited privileges to the host server, and using a reverse proxy (Traefik, see D3.2, section 4.7.1) to filter network traffic and HTTP requests (see also D2.7 Section 3.6.3).

4.1.2.Authentication and Authorisation

Access to CIMI resources is authenticated at the client level; in the mF2C case, a client would be an individual Agent. mF2C uses PKI, and each Agent is identified by an X.509 certificate issued when the agent joins a fog cluster. In D2.5 Section 4.3.3. we summarised the mF2C credential model, based on X.509 certificates; the certification process itself is described in D5.1 [D5.1] Section 3.2 (Discovery, Authentication (Security) and Categorization Workflow). Note the certificate is used for authentication but not for authorisation.

CIMI currently offers several options for authentication and authorisation, which include:

- internal username/password
- api-key/secret
- Github OAuth protocol
- OIDC protocol

It should be noted that CIMI mandates that all mF2C users register for a conventional username and password credential even though they may not use this internal identity directly for authentication and authorisation. CIMI maps a user's other credentials, e.g. api-key, to this conventional credential and maintains the mapping in an internal database.

As CIMI uses REST over HTTPS protocol, mF2C opted to adopt the standard WS security protocol of JWT, which CIMI already supports. Table 2 shows the standard claims (fields) in a JWT [RFC7519]. As we do not intend to use Proof-of-Procession, the Subject (the identity bearer) must be defined and this could be the DN attribute in the Agent's X.509 certificate.

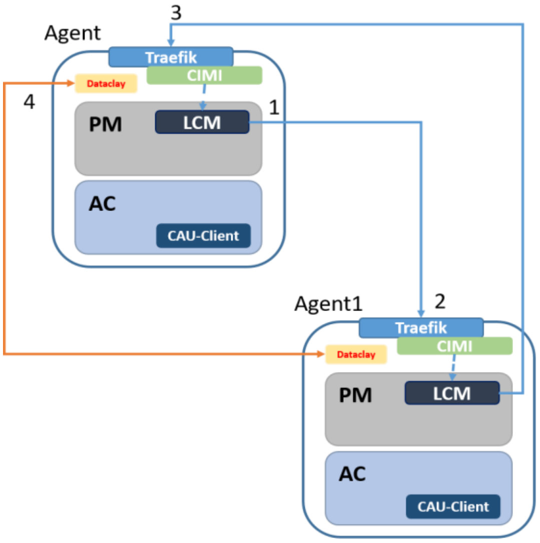
Code	Name	Description
iss	Issuer	Identifies principal that issued the JWT.
sub	Subject	Identifies the subject of the JWT.
aud	Audience	Identifies the recipients that the JWT is intended for. Each principal intended to process the JWT must identify itself with a value in the audience claim. If the principal processing the claim does not identify itself with a value in the aud claim when this claim is present, then the JWT must be rejected
exp	Expiration time	Identifies the expiration time on or after which the JWT must not be accepted for processing. The value should be in NumericDate format.
nbf	Not before	Identifies the time on which the JWT will start to be accepted for processing.
iat	Issued at	Identifies the time at which the JWT was issued.
jti	JWT ID	Case sensitive unique identifier of the token even among different issuers.

Table 2. JWT standard claims

PM6: Implement JWT authorisation for CIMI. To be precise, CIMI already supports JWT; the issue is to implement a set of authorities that can issue tokens, and to define a scheme as in the table above which describes capabilities etc.. and ensure that it is enforced.

4.1.3.Securing Communications between Agents

This section sketches out how an Agent can securely access a CIMI-managed resources on another Agent. The scenario used for modelling the interactions is based on the use case that an Agent (called "Agent" in Figure 2, below) needs to request resource from its parent ("Agent1"). (For now, we assume that the Agents can only communicate vertically as in IT-1, not horizontally). We provide a discussion of the process interactions, roles and responsibilities of the components involved in more detail in section 4.1.3.1, below.



Inter-agent communication use cases:

Case 1: A.LCM -> A1.LCM:

1. A.LCM makes a ReST call to A1.LCM api endpoint to request resource
2. Call goes via A1 traefik (reverse proxy) which forwards the call to the LCM
3. A1.LCM replies via similar arrangement

Case 2: A.Dataclay replicates leader data to A1 (backup leader)

4. A.Dataclay stream data via gRPC directly to A1.Dataclay
5. (Child) A1.Dataclay replicates resource data to (Leader) A.Dataclay

CIMI currently offers 4 modes of authentication; each associated with a "Session Template" resource:

1. username/password
2. apikey/secret
3. Github OAuth protocol
4. OIDC protocol

Figure 2 Inter-Agent communication use cases

The two cases - CIMI (accessing LCM in Fig. 2) and DataClay - use different network protocols: REST web services, and gRPC respectively. The underlying communication layer though is the same for both - HTTP - in the sense that they both have the same transport layer and application protocol, so the options discussed in section 3.1 apply to gRPC as well as REST. Section 4.1.3.1 provides the remaining details for CIMI, whereas DataClay is covered in section 4.2.

4.1.3.1. Token-based Secure Inter-agent Communication via CIMI

Figure 3 illustrates the main process flow based loosely on OAuth2 Implicit Flow (RFC 6749) and the derived Disconnected Flow [Sandoval] for constrained IoT devices. In mF2C, we are concerned with M2M interactions with minimal involvement of the User Principal (or "Resource Owner" in OAuth2 terminology, RFC 6749). Hence, conventional OAuth2 processes (e.g. Authorisation Code flow) which require the User Principal/Agent to interactively provide credential to a third party IdP (e.g. Google, LinkedIn) is not really applicable for our purpose. This flow, however, should be considered for other deployments that, to improve usability, choose to make use of user's existing credentials rather than having them create a new registration in the portal; it would be entirely appropriate for a flow in which the agent's connection to the fog is authorised through the user (for the Smart Boats and Smart Hub use cases.) That being said, there is definitely a need for a client-only (i.e. agent without the User

Principal) authorisation, since agents will need to act on behalf of the user and a user may control several agents at the same time.

The above description covers most parts of the proposed flow, except:

- Traefik processing, i.e. the reverse proxy that forward requests to the relevant CIMI controlled resources. Traefik is described in D3.2, section 4.7.1.
- Fine-grained CIMI and CAU internal processing. We are investigating using Hyperledger Fabric to provide a permissioned blockchain for near-real time sharing of security information (e.g. Agent public keys) in the CAU network. This is also described in D3.2, in section 3.3.

And we assume that:

- The CAU client is an internal block of an Agent trusted by other blocks in the same Agent and they communicate via secure private Docker network.
- The two Lifecycle Managers (LCM) communicate using REST over HTTPS in line with CIMI specification.
- An internal user for Agent0 exists in the Agent1 CIMI instance (see Section 4.1.2) either through:
    - Auto-registration, or
    - Exclusive registration during a prior process, e.g. discovery
- Only vertical communication permitted between Agents.

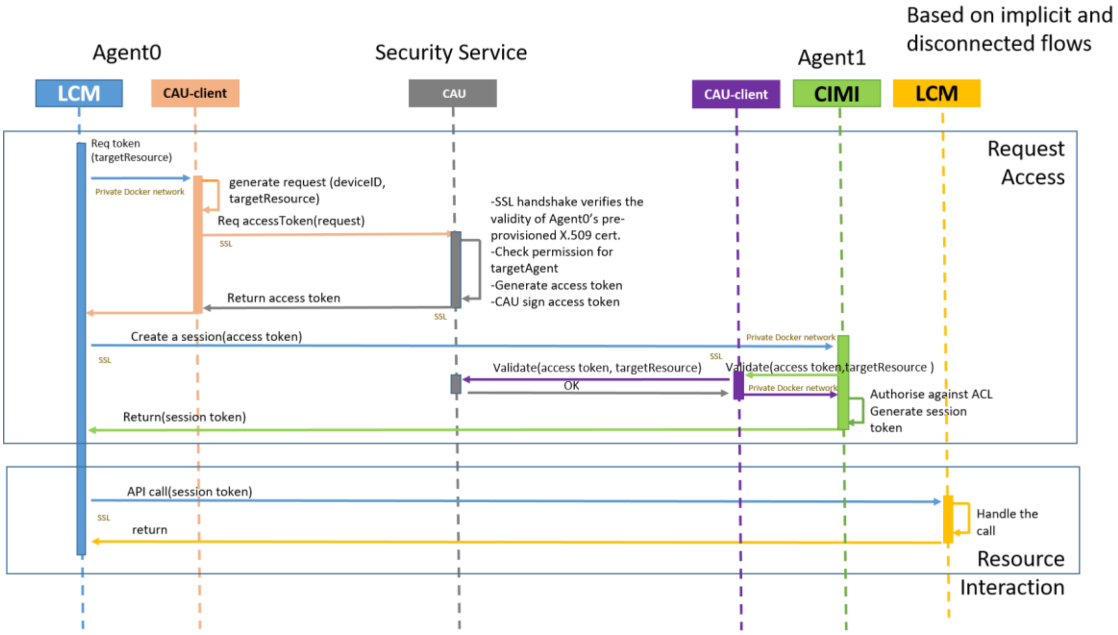


Figure 3 Authentication and authorisation process flow for inter-agent communication.

To help understand the responsibilities of the key components featured in the flow, we attempt to approximate our components to OAuth2 convention as follow:

- Agent0.LCM : Client Application
- CAU : Authentication and Token Introspection Services
- CIMI : Resource Service and Authorization Server (CIMI uses user-defined ACLs to control access to resources)

In mF2C, the CAU-client works as a proxy to the CAU. The CAU is a loosely-coupled security service that offers security functionality locally to nearby agents (see D2.7 Section 2.5.3). The CAU Client interacts with the CAU, which authenticates the requesting Agent against its database of X.509

certificates and returns a self-contained access token attesting the Agent's claims. The Agent then exchanges the access token for a CIMI session token granting it time-limited access to the requested CIMI resource (e.g. an Agent block).

PM7: Implement OAuth2 support. Specifically, the requirement is to identify the location of the authorisation server, and the types of flows that we need (e.g. with user, or without.) Unlike JWT, OAuth2 generally assumes a single authority.

4.2. DataClay Security

As discussed previously in Section 4.1.3, DataClay performs direct connections between agents through gRPC. This system, developed at Google, uses HTTP/2 (HTTP version 2) for transport, so all the security technologies described in section 3.1 apply. Given that there is in mF2C a pre-existing PKI for "Agent Credential" X.509 client certificate, these suffice to secure the channel between agents for IT-2.

The choice of this socket-based schema is driven by the fact that DataClay security must be performed at system level (and not user level, such as CIMI). By enabling the HTTPS mode in the gRPC client and server the communication is encrypted in-flight; the server certificate impedes any Man-in-the-Middle (MITM) attack on DataClay communications and the use of client certificates protects the DataClay infrastructure against any rogue unrecognized device.

The plan for IT-2 is to secure the communications between agents as explained above, taking advantage of the security features offered by gRPC. For this, we will need to make some adaptations in the way DataClay uses gRPC, and integrate the existing mF2C client certificates in the process.

PM8: implement gRPC security.

4.2.1.Data Security

One of the options for supporting the security goals of the GDPR is to encrypt data at rest (i.e. while it is stored, with support for encryption/decryption either within DataClay or in the agent.)

We consider encryption at rest beyond the scope of mF2C because of the complexity of implementing the necessary key management in a voluntary and dynamic F2C ecosystem. In terms of data in transit, TLS already ensures point to point security as data are encrypted between the client and the server; in transit, data is protected with an ephemeral key which can be discarded once the transfer has finished, so the issue of key management does not arise.

In D4.1 we analysed the data flows in the three mF2C use cases and concluded that while TLS is sufficient for most application situations, Use Case 2's (Smart Boat) data plan sharing scenario means that private data may be relayed through third parties to the intended recipients. For this particular scenario, encrypting data at the message layer using a shared secret with the recipient is recommended. D3.1 similarly discussed security policy requirements for control data. With the exception of the scenarios described for inter-agent communication (see Section 4.1.3), control data flow between blocks through private Docker network and do not require additional security measures.

4.3. Networking, Protocols and Firewalls

One of the fundamental security principles is that agents do not, as a rule, have access to the Internet through mF2C until they have been authenticated (and authorised) . This is to protect the reputation of mF2C. This section reviews the options for implementing fog-to-cloud security for unauthenticated agents, authenticated agents, and, by extension, for applications deployed on the mF2C framework (if they use the same transport layer).

In D3.2 section 3.1, we described the minimal requirements for implementing an mF2C fog - basically, a transport layer with a CAU, connected to a CA in the cloud. However, there is of course no requirement that fogs be the leanest possible, and the requirements of applications may also impose extra demands on the fog. Although architecturally out of scope for the mF2C framework, we discuss options briefly in section 4.3.3; see also D3.2, section 4.8.

4.3.1.Initial Fog to Cloud communication - CAU

An agent will need to obtain a certificate (D3.2, section 3.2) with which it can authenticate itself and implement the security mechanisms described in D3.2 section 2.1 and in section 2.1 of this deliverable. In IT-1, we introduced a CAU (D3.2, section 4.3) as a gateway that connects an as yet unauthenticated agent to a service in the cloud that issues the certificate (see also D2.7, sections 4 and 4.6.1). The CAU is generally provided locally, in each fog cluster (cf. D3.2 sections 3.1 and 4.4.1) - in fact, in IT-1, the requirement was that every fog cluster must have a CAU.

In IT-2, we are introducing an additional CAU service in the cloud, the "Cloud CAU" (D2.7, section 5.1.) One purpose of this Cloud CAU is to enable agents to get certificates if they are attempting to connect to a fog that does not have a (fog) CAU, or, for agents that have an independent direct connection to the Internet.

As a consequence of this architecture, a fog that does not have a local CAU, and does not provide unauthenticated access to the Internet, cannot bootstrap agents that do not have independent access to the Internet. Agents that have been bootstrapped elsewhere, e.g. in another fog, or a previous Internet connection, could still connect to this fog, as they would not need to be bootstrapped again.

4.3.2.Fog to Cloud Communication of Registered Agents

Once an agent is able to authenticate, the next question is whether an agent needs to send data to services in the cloud, and, if it does, how it does it (assuming the device hosting the agent does not have its own independent network connection, of course). Does it get permission to route packets out of the fog through the transport layer provided/used by mF2C? Or does it route messages via its leader(s)? The latter could be implemented similarly to the horizontal agent-to-agent communication in IT-1, where messages were required to be relayed by the (single) leader. The former could be implemented through a separate access-controlled network which can route packets to the Internet, or through a router which requires X.509 client authentication.

A compromise between the two options would be to implement additional gateways, like the CAU, that are able to relay messages from agents in the fog to services in the cloud.

It may be necessary also to ask the opposite question: how can a cloud service send a message to an authenticated agent? In this case, querying the Cloud CAU for the location of the agent in question - which fog is it in - should be able to facilitate the cloud-to-fog communications.

PM9: Implement fog-to-cloud communications for authenticated agents.

4.3.3.Fog to Cloud Communication for Applications

With the discussion in 4.3.2 in mind, we now need to look at applications deployed on mF2C: after all, if they were unable to communicate with the cloud, it would not be much of a fog to cloud infrastructure. Assuming the hosting device does not have independent means of reaching the Internet, how can we enable the application to communicate with services in the cloud? A simple option is to use DataClay (section 4.2) to replicate data between the fog and the cloud. This option, however, requires that we can route DataClay gRPC calls between the fog and cloud - if using the socket mechanism proposed in section 4.2, it can probably only happen through a gateway, although this would require further investigation (the problem being that security is negotiated only after the socket is opened, so the security challenge is not available during the process that establishes the TCP socket). Also, COMPSs provides mechanisms (section 4.5.1) for sufficiently secure communications and it may be worth investigating also how to selectively route COMPSs-specific messages to COMPSs services in the cloud - this might be easier than for DataClay, because COMPSs security is built using GSSAPI, so individual messages are "wrapped" - see section 4.5.1.

Beyond DataClay and COMPSs, it may be necessary for applications to make other types of communications with cloud services. Again assuming no alternative means of reaching the Internet, applications need to authorised to send messages over the mF2C fog network. There are three options for this, which are not necessarily mutually exclusive:

- We may require to be PKI-based authentication (cf. D3.2, section 4.8), optionally reusing existing mF2C code such as the CAU client (D2.7, section 4.6.1) for their own bootstrap mechanism. An authenticated client could be authorised to access the cloud as described for agents in 4.3.2, by authenticating to a router.
- We require users to authorise the access, e.g. through a portal where they accept the acceptable use policy, or authenticate through emmy (D3.2, section 4.6.1), or provide credit card details, or some similar mechanism.
- We authorise access through a token. Particularly for web services, or where having a certificate alone may not be sufficient, token-based authorisation methods (cf. section 3.1.5) can be useful.

4.3.4. Firewalls

Some of the security code prototyped for IT-1 (cf. D3.2, section 2.1) investigated remotely controlling firewalls in order to control botnets, etc. Resurrecting this early work and making use of it by connecting it with the ML facility (section 4.6.3), in order ensure that an automated reaction to an incident can be implemented, would be interesting, but is probably not achievable in the remaining year of the project, given our other priorities.

4.4. Service Orchestration

Service orchestration (Figure 1) covers SLA Management, Lifecycle Management, and Service Management. In this section we discuss only SLA Management separately; it is thought that securing CIMI (section 4.1), plus the security provided by the local Docker network (section will be sufficient to provide security for the other components. See also Figure 2.

4.4.1. SLA Security

The concepts and models used in the SLA Management component are inspired by the WS-Agreement specification. Therefore, we have followed the approach of WS-Agreement with regards to security. Its "Security Considerations" section says [GFD.192]: "[...] agreement participants SHOULD be authenticated to insure their identity of the initiator during creation and management of an agreement. Further, one MAY wish to provide a method for signing or otherwise authenticating a document based on the WS-Agreement schema if that document is to be consumed outside the interactions defined by the WS-Agreement interactions and port-types" (this uses the additional terminology of [RFC 2119]).

The SLA Management can only be accessed from the outside through the CIMI component. Since requests to CIMI resources needs to be authenticated at the user level following the mechanism explained in section 4.1.2, the identity of the agreement initiator is ensured.

Although the agreement document is not used outside the SLA Management once it has been created and stored - it is just used internally for the SLA assessment - there is a field in the agreement document used in mF2C intended to contain the X.509 signature of the details field of the agreement. The procedure that could be followed is:

1. The agreement details are generated based on a template and sent to the client, who can then inspect its contents - or a textual representation of it.
2. If the client agrees, the digital signature is obtained using the client's certificate, and stored in the signature field.
3. The agreement document, containing the details and the signature is sent to the SLA Management and stored.

4.5. Distributed Execution Runtime

As regards the Distributed Execution Runtime, we need to cover only the COMPSs software.

4.5.1.COMPSs

To protect applications from eavesdroppers, the runtime has a security mechanism that provides communications with confidentiality, integrity and authentication. For its implementation, the runtime leverages the GSSAPI, an IETF standard API to access security services, so developers create secure applications while avoiding security-vendor lock-in.

Besides defining a common interface, GSSAPI also provides an operating model where both ends negotiate a secure context – authenticate themselves and agree on the mechanisms for data ciphering and integrity – before transferring any information. Upon the establishment of the security context, GSSAPI processes (“wraps,” in GSSAPI terminology) the messages and opaques their content, returning a token that can be securely shipped to the other end. Although GSSAPI defines the format of the exchanged tokens and its content – actually, the security framework does – it does not establish nor provide any transmission mechanism. Therefore, applications invoke GSSAPI to wrap a value and obtain a token to ship to the other end. Upon the reception of a token, the receiver invokes GSSAPI to unwrap the token (i.e. decrypt, and, optionally, validate signatures) and recover the original content of the message. In our case, COMPSs uses the Java NIO library to transfer tokens over TCP sockets.

4.6. Telemetry and Monitoring

This Section walks through several possible risks and relative possible solutions as regards telemetry and monitoring. Specifically:

4.6.1 - addresses main security and privacy concerns about Landscaper and Analytics component, as well as highlighting general security measures that should be extended across the entire solution.

4.6.2 - focuses on monitoring and telemetry by providing possible approaches to secure it.

4.6.3 - concludes the Section by highlighting future possibilities on how to use the above components and ML for security purposes.

4.6.1.Landscaper and Analytics Engine

The highly dynamic environment in which mF2C operates, require additional security measures to be enforced across the system. Indeed, traditional measures operated by data-centers (e.g., perimeter security) are not enough anymore. Furthermore, the high dynamicity of both Cloud and devices in the fog increases the likelihood of security risks such as:

- component/device theft;
- component/device manipulation;
- no possible assumption about infrastructure availability and relative security.

Hence, we recognize the importance of securing each component (multi-layer security, or security in depth). We analyzed what could be done in order to improve overall security and privacy. Both Landscaper and Analytics Engine could deal with potentially sensitive data that must be protected by preventing non-mF2C components to get access to it. Specifically, a federated approach could be used in order to enforce authentication and authorization across the entire system.

As regards protecting inner working of those two components, robust encryption mechanisms should be used to securely store data. Furthermore, by enforcing encryption across the entire system's communications (not only for protecting data) it would be possible to mitigate attacks such as the MITM attack. Indeed, let's consider the following scenario: two components (either application level or hardware level) need to exchange information between each other. The communications will be at risk for both passive MITM (i.e., eavesdropping) and active MITM (i.e., actively capturing information exchange and potentially manipulating it). Furthermore, good practices would be to secure any communication, not only those carrying sensitive data. The proposed CIMI security framework should be sufficient to address both all of these concerns.

4.6.2.Monitoring

Monitoring will be carried out by the telemetry framework which uses a collection of customized modules (plug-ins) to instrument the devices and their executing services. Examples of these plugins include monitoring of the CPU, disk I/O usage, Docker Container metrics, etc. By default, all telemetry plugins to be installed need to be signed. Private/public keys and keyrings are generated by the GPG (GNU Privacy Guard) suite of cryptographic software . The plugin is signed with the private key and the public key needs to be added to the deployer's keyring (cf. section 3.2.) The signing is an armored detached signature in the form of an .asc file.

Like DataClay, the framework communicates with these plugins over the gRPC protocol. In order to avoid transferring the metrics in plaintext, we need to secure the communications through any of the available methods (sections 3.1 and 4.1.3.) These would need to be set at the installation phase, as described in section 3.2.1, or, alternatively, use the agent certificate. Monitoring on the same physical host can of course continue to use the host's private Docker network.

All metrics captured are published to a Timeseries database (TSDB); the current implementation is InfluxDB. The database credentials (username/password) are required to be stored in the monitoring framework's config file - in addition to other config items. So ensuring this is restricted to prevent unauthorised access to the raw metrics needs to be understood - it would not make sense to bake this into an image, as in the distribution of the trust anchors (section 3.2.1.) Currently only the Analytics Engine module requires access to this data as it analyzes device and service performance.

PM2: Eliminate shared secrets.

PM8: Implement gRPC security.

4.6.3.Machine Learning for Security

In deliverable [D2.2] and [D2.5], we discussed the emerging role of Machine Learning (ML) and other Artificial Intelligence (AI) approaches in the field of cybersecurity, offering alternative solutions to many interesting research challenges. While the focus is usually on how to use different machine learning techniques in the cloud and large servers, for the mF2C platform, it is more important to focus on bringing the analysis closer to the fog nodes. The goal is to achieve offloading of some security functions to different mF2C devices in the fog layers.

In the current mF2C architecture it makes more sense to implement ML in Platform Manager components, specifically the ones that are in charge of analytics and telemetry, as well as Lifecycle Management. As a starting point on how to implement ML for security in systems like the one mF2C offers, we consider the research done in [Moh], which offers an overview in ML algorithms (anomaly detection models, optimization methods, reinforcement learning) used in different fog computing use cases. The mF2C ML security approach can benefit from the results of this research, which investigates the effectiveness of various ML techniques in different architecture layers. With reference to the mF2C architecture specifically, the research suggests that algorithms such as K-nearest neighbours, Random Forest, Naive Bayesian, Logistic Regression, Decision Trees and Artificial Neural Networks would make sense to be applied in the fog layers of our architecture. These algorithms can be used by Telemetry and Lifecycle component for detecting different anomalies such as jamming or DDoS attack. The telemetry component, which is responsible for tracking mF2C agents joining and leaving the system, can use this knowledge for analysing the behaviour and take preventative actions against possible attacks. Lifecycle Management, in a similar fashion, can perform analysis while receiving service instances in order to determine if unexpected high number of service requests is due to a potential attack.

Another possible application of ML for security, mentioned in [D2.5], is based on the application data. That is, the data collected locally in mF2C nodes in different fog layers of the mF2C architecture can be used to detect anomalies and to model system behaviour patterns. However, for this approach it is necessary to mention that the mF2C system is not application data oriented, so it is incumbent on the use case owners to on their own take advantages of mF2C infrastructure and leverage different ML algorithms in nodes that have enough processing power.

5. Conclusions

Unlike the Agent Controller, the Platform Manager (PM) had very little security prototyped for it in IT-1, relying instead only on demonstrating core functionalities. Thus, except for prior security code for COMPSs, which as yet remains to be integrated with mF2C, we have little experience with security code for the PM in IT-1, so this deliverable necessarily starts from the basic requirements. Nevertheless, what stands out is the commonalities - once we support REST web services (for CIMI), gRPC (DataClay and Telemetry), and GSSAPI (COMPSs), we can in principle implement security for all of the PM components. However, the deliverable describes how this security foundation alone is insufficient to meet the needs of the PM security, as a number of action items have been identified throughout the deliverable. These are, in order introduced, listed in the table below.

#	Short Description	Explanation	Section
PM1	Signed images	Monitoring and trust anchor deployment suggest signed images	3.2.1
PM2	Eliminate shared secrets	Shared secrets cannot be deployed in images	3.2.1, 4.6.2
PM3	Images for trust anchors	Using images for trust anchors would make it easier to build trusted mF2C containers, and to customise them by replacing the trust anchors	3.2.1
PM4	Support Raspberry PI	In addition to NuvlaBox Nano, Use Case owners still need Raspberry PI in IT-2	3.2.2
PM5	Pre-configure apps with required information	For user-facing applications, aim to minimise the amount of post-install configuration users have to do.	3.2.3
PM6	Implement JWT authorisation	JWT provides a token-based authorisation with the possibility to distribute authorities	4.1.2
PM7	Implement OAuth2 support	OAuth2 has been proposed for IoT, and is suitable for flows involving users (such as Smart Boats and Smart Hub)	4.1.3.1
PM8	Implement gRPC security	gRPC - used by monitoring and DataClay - needs security integrated into the mF2C security framework	4.2.4, 4.6.2
PM9	Fog-to-cloud communications	How can agents be allowed to communicate with cloud services, but restricting untrusted agents (or attackers) from the same privilege?	4.3.2
PM1	Signed images	Monitoring and trust anchor deployment suggest signed images	3.2.1

Table 3. Summary of action items

Solutions for all of these should be demonstrated in IT-2. As with D3.2, we cannot achieve everything described in this deliverable - it would add years to the project - but it seems pertinent to require that the above points be addressed.

As a part of the prioritisation process, it is important to define also what is out of scope, like we did with IT-1. For example, we have decided that encrypting data at rest (where it is stored) is out of scope also for IT-2, due to the difficulty in implementing a suitable key management. As long as the decisions to not implement a security feature is taken in the open, we can make sure we understand the consequences and can assess the impact on the overall security functionality of the project.

Beyond the challenges described above, this deliverable also identifies research challenges. These are, again in no particular order,

- Using machine learning to detect security incidents, and, in particular, to place as much of the analytics as possible in the fog, according to the core principles of mF2C.
- Using distributed ledger technologies to improve the security features of the CAU, the fog-to-cloud gateway demonstrated in IT-1.

References

[D2.2] mF2C Consortium, “D2.2 Tracking Scientific, Technology and Business Trends (Version 2),” September 2018. [Online] http://www.mf2c-project.eu/wp-content/uploads/2018/10/mF2C-D2.2-Tracking-Scientific-Technology-and-Business-Trends-Version-2.pdf

[D2.5] mF2C Consortium, “D2.5 mF2C Security/Privacy Requirements and Features (IT-2),” January 2019. [Online]. http://www.mf2c-project.eu/d2-5-mf2c-security-privacy-requirements-and-features-it-2/

[D3.1] mF2C Consortium, “D3.1 Security and privacy aspects for the mF2C Controller Block (IT-1)”, June 2017 [Online] https://www.mf2c-project.eu/wp-content/uploads/2017/06/mF2C-D3.1-Security-and-privacy-aspects-for-the-mF2C-Controller-Block-IT-1.pdf

[D4.1] mF2C Consortium, "D4.1 Security and privacy aspects for the mF2C Gearbox block (IT-1)" June 2017 [Online] https://www.mf2c-project.eu/wp-content/uploads/2017/06/mF2C-D4.1-Security-and-privacy-aspects-for-the-mF2C-Gearbox-block-IT-1.pdf

[D5.1] mF2C Consortium, "D5.1 mF2C reference architecture (integration IT-1)" May 2017 [Online] https://www.mf2c-project.eu/wp-content/uploads/2018/06/D5.1-mF2C-reference-architecture-integration-IT-1.pdf

[GFD.192] Andrieux, A. and Czajkowski, K. and Dan, A. and Keahey, K. and Ludwig, H. and Nakata, T. and Pruyne, J. and Rofrano, J. and Tuecke, S. and Xu, M. WS-Agreement specification, GFD.192. September 21, 2011

[MR] M. Moh and R. Raju, *Machine Learning Techniques for Security of Internet of Things (IoT) and Fog Computing Systems*, 2018 Int. Conf. on High Performance Computing & Simulation (HPCS), Orleans (2018), pp. 709-715.

[OIDC] *OpenID Connect*; https://openid.net/connect/

[SAMLP] J Hughes, S Cantor, J Hodges, F Hirsch, P Mishra, R Philpott, E Maler (eds) *et al.*: *Profiles for the OASIS Security Markup Language* (2005) http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf

[RFC 2119] S Bradner: *Key words for use in RFCs to Indicate Requirement Levels*, (1997); http://www.rfc-editor.org/rfc2119.txt

[RFC 2818] E Rescorla: *HTTP over TLS* (2000); http://www.rfc-editor.org/rfc2818.txt

[RFC 3281] S Farrell, R Housley: *An Internet Attribute Certificate Profile for Authorization* (2002); http://www.rfc-editor.org/rfc3281.txt

[RFC 6680] N Williams, L Johansson, S Hartman, S Josefsson: *GSSAPI Naming Extensions* (2012); http://www.rfc-editor.org/rfc6680.txt

[RFC 6749] D Hardt *The OAuth2 Authorization Framework* (2012); http://www.rfc-editor.org/rfc6749.txt

[RFC 6750] M Jones, D Hardt: *The OAuth 2.0 Authorization Framework: Bearer Token Usage* (2012); http://www.rfc-editor.org/rfc6750.txt

[RFC 7235] R Fielding, J Reschke (eds): *Hypertext Transfer Protocol (HTTP/1.1): Authentication* (2014); http://www.rfc-editor.org/rfc7235.txt

[RFC 7519] M Jones, J Bradley, N Sakimura: *JSON Web Token (JWT)* (2015); http://www.rfc-editor.org/rfc.txt

[RFC 7833] J Howlett, S Hartman, A Perez-Mendez (ed): *A RADIUS Attribute, Binding, Profiles, Name Identifier Format, and Confirmation Methods for the Security Assertion Markup Language (SAML)* (2016); http://www.rfc-editor.org/rfc.txt

[Sandoval] K Sandoval: *OAuth2: why it's vital to IoT security*; https://nordicapis.com/why-oauth-2-0-is-vital-to-iot-security/ (accessed Jan.2019)

Annex 1: At-Rest Encryption

Background

Encryption at rest means data is encrypted while stored. The storage could be a working copy of the data, or a backup, or an archive.

While at rest encryption mainly targets an unauthorised person accessing the data but cannot read it because it is encrypted, D2.4 mentions at rest encryption only in connection with the GDPR: partly to protect the data, and partly to implement destroying the data: the conventional approach to deleting a person’s data is to throw away the encryption key – that way, one does not have to worry about deleting cached copies of the data, backups, archives, etc.

Normally data has to be decrypted before it can be processed, which means that the processor must have access to the key, or something else must decrypt the data before it reaches the processor. Obviously the key must be access controlled, or an attacker might get access to the key.

Key Management

One should be careful not to confuse the keys. Entities (such as agents) have asymmetric key pairs, a public key typically tied to the entity's identity in a certificate, and a private key which can decrypt stuff encrypted with the public key, and vice versa.

Note that data encryption is never done with the asymmetric keys, whether at rest or in flight. This is because:

- Asymmetric key cryptography is slow, much slower than symmetric key
- Asymmetric keys can only encrypt small messages, and there are some other more exotic restrictions in RSA for example.
- The key management would be a nightmare in general, particularly for keys that “expire.”

For both in-flight (resp. at-rest) encryption, a symmetric key is generated randomly, the data is enciphered with the symmetric key, and the symmetric key itself is encrypted with the public key of the recipient (respectively, owner) of the data. This is the way to have more than one owner, by encrypting the same key more than once. Obviously if the symmetric key is compromised by one entity, the whole thing falls apart - the data will need to be decrypted and re-encrypted.

Escrow

Sometimes symmetric keys have to be escrowed. This means that they are stored with a “trusted third party” so the ciphertexts can be deciphered in case the normal access path becomes inaccessible. It would be perfectly sensible for a company, for example, to be able to decrypt employee’s encrypted laptops. It may also arise from compliance: if law enforcement needs access to the data.

Key Strength

For encryption in flight, it normally suffices to have keys strong enough to protect the data in flight, although there remains a risk that an attacker can record the data and attack it years later, when computers are more powerful and/or vulnerabilities have been discovered in the algorithm. For most of data that is encrypted in flight (typically HTTPS web browsing and web e-commerce), worrying about an attacker deciphering the data years later is simply not relevant: the data will be useless by then.

For data at rest, particularly archived data, encryption needs to be strong enough from the start that the data is considered safe for potentially decades. There are numerous other issues that would need thought, such as maintaining access control lists. Loss or leakage of the symmetric key may also become more likely with time. It may be feasible to “wake up” the data to do format migrations and re-encrypt it at the same time.

Implementation

Implementing at-rest encryption, even without an archive facility, is non-trivial. In mF2C, it would have to be implemented on top of the DataClay security, with a separate key management database, and, due to the way DataClay interfaces with the components, would most likely need deeper alterations to DataClay itself.