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

D2.5 mF2C Security/Privacy Requirements and Features (IT-2)

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

Work Package	**WP2 Technology Survey, business models and architecture definition**
Due Date:	*M24*
Submission Date:	*28/12/2018*
Version:	*1.0*
Status	*Final*
Author(s):	*Jens Jensen, Shirley Crompton (UKRI-STFC)* *Cristovao Cordeiro (SixSq)* *Denis Guilhot (WOS)* *Antonio Salis (ENG)* *Matija Cankar (XLAB)*
Reviewer(s)	*Cristovao Cordeiro (SixSq)* *Denis Guilhot (WOS), Olmo Rayón (WOS)*

Keywords
Security, Privacy, Architecture, Requirements

Project co-funded by the European Commission within the H2020 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	*16/11/2018*	*Initial import from earlier draft*	*STFC, SIXSQ, WOS*
0.2	*16/11/2018*	*Add section about container security*	*Cristovao Cordeiro (SixSq)*
0.3	*21/11/2018*	*Add chapter 4 content*	*Antonio Salis (ENG), Matija Cankar (XLAB)*
0.4	*17/12/2018*	*Internal review*	*Cristovao Cordeiro (SixSq), Denis Guilhot (WOS), Olmo Rayón (WOS)*
0.5	*20/12/2018*	*Addressing reviewer comment (C Cordeiro, D Guilhot, O Rayón), additions from A Salis and M Cankar.*	*J Jensen (STFC)*
1.0	*28/12/2018*	*Quality Check. Document ready for submission.*	*Maria Teresa Garcia (ATOS)*

Table of Contents

List of figures

List of tables

Executive Summary

An IoT infrastructure needs to be secured, in order for people to trust it, and to prevent attackers from compromising the security and privacy goals of the applications deployed in the infrastructure. This deliverable extends the requirements defined in D2.4 [1], and makes them more precise, by taking into account the experiences from IT-1, the planned work for IT-2 (numerous targets were considered out of scope for IT-1), and recent industry and academic research.

The security components developed for IT-1 were sufficient to validate the basic design, of using agent-based security supported by public/private keys, where the public keys were signed by an authority that was deployed in the cloud. Additional components developed for IT-1, but in the end not integrated into it, highlighted additional issues which are still largely unresolved, the first of which is how to obtain a consistent implementation of the security model across all components, without re-inventing the wheel. In particular, the need for message-based and channel (socket) based security was raised (features that come out of the box when using SOAP but are much harder with REST), as well as whether we should encrypt stored data. The answer seems to be that mF2C use cases need both message-based and socket-based security, but encrypting stored data is not a good idea because key management is hard and does not seem suited to a fog architecture.

This deliverable describes in some length secure-by-design software development (greatly extending the corresponding section of D2.4). Certainly, not all software in IT-1 was developed under this principle, and following all the recommendations would triple the length of the project, so priorities need to be set. These need to be set after D3.2 and D4.2 have been finished, with the proposed implementations of the security requirements set out here.

Other issues that were postponed from IT-1 include the physical security of the devices, where, based on industry “solutions,” it would make sense to at least investigate a few of the options provided by the industry, as well as current practices for implementing tamper-resistant or tamper-evident devices, so in turn the project can make recommendations to parties that wish to make use of the mF2C results. Physical security of the edge and fog devices is a particular concern in UC2, but also UC1, albeit to a lesser extent.

Another requirement revealed by the analysis of use cases is the need – somewhat contrary to earlier plans – to support application security on the mF2C framework. This, if implemented, would need to be done using the same code as used by the platform (itself re-engineered from the code developed before IT-1), but using different keys, so a compromised application would not compromise the rest of the infrastructure.

Beyond this, there are opportunities to do more research-type work into security applications of blockchains and machine learning. Like the physical security mentioned earlier, this work should be able to happen in parallel with the implementation of the security which will be proposed in D3.2 and D4.2, with only relatively few dependencies between the areas. As initially outlined in D2.4 [1], there is now an opportunity to make good on the promise that deliverable to investigate acting on the monitoring data to automatically handle security incidents, without blocking legitimate activity that will look similar, such as an emergency in UC1.

1. Introduction

This is the follow-up deliverable from D2.4 [1] which described security and privacy features and requirements in general, before it was known what could realistically be delivered for IT-1 (D2.4 was written for PM04, April 2017.) As such, the present deliverable brings together information from D2.4 [1], D2.6 [2]and D2.7 (architecture and revised architecture, respectively), as well as the experiences with security in IT-1.

1.1. Purpose

This document ("Security and Privacy Features and Requirements for IT-2") is not intended to repeat D2.4 [1] which was quite comprehensive and broad in scope; rather, we highlight the most relevant topics and their application to the mF2C IT-2 release.

It should be noted that there are currently three other deliverables in preparation which cover security topics from different perspectives. D3.2 covers the security technologies and implementation of the Agent Controller, and D4.2 the same for the Platform Manager. As mentioned above, D2.7 covers the Architecture for IT-2, and also discusses its security. These deliverables provide in more detail how the requirements listed here would be implemented.

1.2. Scope

A *raison d'être* for mF2C is to provide a framework for developing fog-to-cloud applications. Specifically, mF2C makes it possible to manage heterogeneous and diverse resources that take advantage of the computational and data management capabilities in an ecosystem spanning things at the edge, through smart devices in the fog to large scale systems in the cloud.

Some of the requirements listed here may go beyond a Proof of Concept (PoC). Specifically, if a mF2C platform is to be deployed in a production environment, it will be necessary to implement security measures that may not need to be in the PoC implementation for IT-2. For example, a service run for a PoC may not be deployed with full redundancy and fault tolerance; but it would be desirable to show that the service can be deployed in a fault tolerant way.

1.3. Structure of the document

Section 2 describes prior work in mF2C - security developed for IT-1. The section also briefly expands on the state of the art of the IoT security industry, to the extent that is has changed since D2.4 [1], and is relevant to mF2C. Section 3 provides an overview of the high-level goals for IT-2, with reference to the security developed for IT-1 and the objectives postponed from IT-1 to IT-2. Section 4 describes the security in the architecture in more detail. Section 5 contains the conclusion.

1.4. Glossary of Acronyms

Some of the more common abbreviations like “US” for United States or “IBM” are not included here; some abbreviations that are used only in one location in the deliverable (“PBC”) and/or whose expansion is not relevant to the understanding (“SIM card”, “QR code”) are also not included. When appropriate, the context is explained in parentheses.

Acronym	Definition
AC	Agent Controller
AES	Advanced Encryption Standard (Symmetric key algorithm)
AI	Artificial Intelligence
API	Application Program Interface
CA	Certification Authority
CAU	Control Area Unit (D3.2)
CEP	Complex Event Processing

Acronym	Definition
DAST	Dynamic Application Security Testing
DCT	Docker Content Trust
DDoS	Distributed Denial of Service
DLT	Distributed Ledger Technology
DPO	Data Protection Officer (GDPR)
fTPM	Firmware-based Trusted Platform Module
GDPR	General Data Protection Regulation
gRPC	Google RPC (Remote Procedure Calls)
GSSAPI	Generic Security Services Application Program Interface (IETF)
IETF	Internet Engineering Task Force
IoC	Indicator of Compromise (SOAR)
IoT	Internet of Things
IP	Internet Protocol
IP	Intellectual Property
IT-*x*	(Software release) Iteration *x* where *x* is 1 or 2
JWT	JSON Web Token (RFC 7519)
LSTM	Long/short term memory (machine learning)
MAC	Media Access Control (in “MAC address”)
MAC	Message Authentication Code
MCU	Microcontroller Unit
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
OIDC	OpenID Connect
OS	Operating System
PII	Public Identifiable Information (GDPR)
PKI	Public Key Infrastructure
PM*xx*	Project Month *xx*
PM	Platform Manager
PoC	Proof of Concept
REST	REpresentational State Transfer (web services)
RFC	Request for Comment (IETF)
RPC	Remote Procedure Calls
RSA	Rivest–Shamir–Adleman (Asymmetric key cryptosystem)
SAST	Static Application Security Testing
SECaaS	Security-as-a-Service
SIEM	Security Information and Event Management
SOAP	SOAP web services protocol (originally Simple Object Access Protocol)
SOAR	Security Orchestration, Automation, and Response
SOC	Security Operations Centre
TLP	Traffic Light Protocol (https://www.us-cert.gov/tlp)
TPM	Trusted Platform Module
TRL	Technology Readiness Level (On a scale of 1-9 of increasing readiness)
UC	Use Case

Table 1. Acronyms and abbreviations

2. Prior Work

In this section, we briefly summarise the security code developed for IT-1, as well as the general security trends in the IT and IoT industry.

2.1. Security in IT-1

Table 2 describes the security software developed for IT-1, both components that were demonstrated in the review and some that were not, either for lack of time or because they were not fully integrated in the final release. The table includes an estimated TRL[1] for each component. Some components have low TRL because a decision was made to postpone their development and integration to IT-2.

Security Process/Component	Status (TRL, 1-9)	Needs re-engineering	Demonstrated in IT-1
User registration process	7	NO	YES
Agents bootstrap through CAU	5	YES	YES
Agents have individual X.509 certificates issued by a cloud-based CA, with the CAU as intermediary. On initialisation, agents generate their own key pair and certificate request, and send the request through the CAU to the CA.	6-7	NO	YES
emmy[2] (zero-knowledge proofs library)	4	NO	NO
Agent controller library	5	YES	NO
Socket-based (TLS) security	8	NO	YES
Message based security through JSON signature and encryption	3	YES	NO
Two CAs, one for (semi-permanent) infrastructure and one for agents	4	YES	YES
Security testing framework prototype	3	??	NO
Simulated wireless for security testing	1	??	NO
Incident handling through firewall controls	2	-	NO

Table 2. Security Requirements in IT-1

2.2. Industry IoT Security "Solutions" Architecture

For comparison, we include a brief overview of industry trends in the IoT security and fog-to-cloud space. This is not meant to be comprehensive (D2.4 [1] has already provided a more comprehensive discussion), but rather to illustrate more recent research and development in the IoT security industry, particularly relating to the requirements identified for IT-2 in this deliverable.

2.2.1.Trending Security Features and Approaches

The followings are some of the key trending features and approaches:

- From DevOps to DevSecOps - the approach brings development and operations together through the automation of security tasks by embedding security controls and processes in the DevOps lifecycle. It is about introducing security into every stage of the lifecycle and making

[1] https://en.wikipedia.org/wiki/Technology_readiness_level

[2] https://github.com/xlab-si/emmy

everyone involved responsible for security, not just bolting it on as an afterthought in the deployment stage. The main benefits of this approach include early detection of vulnerabilities and improved security, as security is tailored to the application right from the ground up. Improved security in turn results in fewer incidents and down times during the operation phase.

- Software TPM (fTPM) [3]- messages from the device are signed and claims attested using integrity protected RSA private key stored in the fTPM module.
- Dashboards, SIEM, monitoring, automated incident response to facilitate early detection and prevention of security incidents. Machine Learning algorithms may be used in the background to analyse network traffic or event logs to filter out atypical behaviour which may constitute security breaches.
- Security-as-a-Service (SECaaS) - SECaaS service is a turnkey solution for companies to secure and maintain their IT infrastructure on a cost-effective subscription basis. A SECaaS provider typically provides security services running in the cloud that buyers can integrate into their corporate infrastructure. Such services generally cover:
    - Identity Management and Access, e.g. single sign on, access brokerage, user provisioning;
    - Cryptography and key management;
    - Security policy enforcement;
    - Malware/spyware and intrusion detections;
    - Email security, e.g. protection against “malvertising” (advertising containing malware), targeted attacks, phishing, data breaches, etc.;
    - Additional network security to hardware firewalls that facilitates the early detection of threats through monitoring network traffic and regular security screenings, e.g. penetration testing, log management etc.;
    - Update services to ensure virus definitions, software patches and device firmware are updated independent of user compliance;
    - Web application security through scanning and sealing vulnerabilities on external-facing web applications and access endpoints;
    - IoT security, e.g. device key provisioning, identification, mutual authentication and attestation.
- Blockchain or distributed ledger technology (DLT) - its inherent security features, e.g. tamper-proof fabric with built-in redundancy, transaction encryption, consensus mechanism and access control, etc., are considered by many as an appropriate solution to security and accountability related issues in IoT [4]. There is ongoing research into applying the technology as a security measure to deliver data security [5], manage authorisation [6] and device identification [7], etc.

Open source software security - somewhat curiously from our perspective, the industry seems worried about security risks arising from reusing open source products, both in terms of unmanaged vulnerabilities and “risks to IP” (including licensing) - at least enough to have several companies sell “solutions” to manage risks associated with using open source components. For example, OWASP Dependency-Check[3] scans for known, public security risks in libraries that the code depends on. From

[3] https://www.owasp.org/index.php/OWASP_Dependency_Check

our perspective, the main worry is about whether the upstream code is being maintained, and whether the licences are compatible.

2.2.2.Industry Solution Examples

As mentioned above, the reader is reminded that this section is not meant to be exhaustive, but instead highlight a selection of industry state-of-the-art solutions in IoT which would be worth investigating during the rest of the project. In particular, we summarise a few industry offerings for the physical security of edge and fog devices because this is a concern in UC2 (section 4.1.2) and, to a lesser extent, UC1 (section 4.1.1); see also 4.2.4 for the discussion.

- Azure Sphere[4] - Microsoft described its Azure Sphere product as the "..the most significant push by a large technology vendor to holistically improve IoT security…" Azure Sphere is an integrated product designed with IoT security measures, i.e. secure by design, that cover:
  - hardware - a range of powerful microcontrollers (MCU) with built-in security technology (e.g. hardware root of trust, cryptography) and network connectivity for devices;
  - software - secure OS written for running on the Azure MCU to create a trustworthy OS platform with secured app containers and integrated security monitor;
  - cloud security service - a turnkey security service that guards every Azure Sphere device by renewing security, identifying emerging threats and broker trust among devices, cloud and other endpoints;
  - application development tools - Visual Studio tools and code templates to fast track development of secure IoT applications.

  Except for DLT and open source software security, the security model and features correspond to those highlighted in Section 2.2.1 above.
- Synopsys - Similar to Azure Sphere, Synopsys[5] offers a suite of silicon to software solutions for IoT market which include:
  - DesignWare IP - optimise System-On-Chip design aiming to maximise energy-efficiency of battery operated devices and sensor functions;
  - Wire and Wireless Interface IP - include Bluetooth LE, 802.15.4 and WiFi 802.11ah, LoRA and other low-power cellular technologies to facilitate interoperable connectivity;
  - Security IP - provide a range of defence against evolving threats and to secure data processing; from on board public key and security protocol accelerators, true random number Generators, secure hardware root of trust and secure boot;
  - IP Accelerated - offer a range of prototyping and software development kits as well as integrated IP subsystems to accelerate software development efforts and shorten time to market.
- Secure microcontrollers. Some hardware vendors see an opportunity in securing edge devices and provide specialised devices which are tamper-resistant or provide similar security features.

[4] https://azure.microsoft.com/en-gb/services/azure-sphere/

[5] https://www.synopsys.com/solutions/internet-of-things.html

- Microchip have both secured microcontrollers as well as dedicated security chips [8] which can handle certificates and keys. With the former, a secure microcontroller could be deployed at the edge to provide additional protection. The latter could be used to secure an existing device which is itself physically secure but the connection over which it sends data is not; if the device is not capable of handling certificates by itself, the security chip can be used to handle the protection;
- Xilinx [9] have secured devices for IoT which ought to be capable of implementing a significant compute ability at the edge;
- Security as a service [10].

In addition to the security of the device, there is a related question of the reliability of the device. For example, a basic Raspberry PI (see UC3 below) was never designed to be secure (anyone with physical access to it can remove the memory card), nor was it designed to be run 24/7[6]. However, there are industrial versions of, essentially, the Raspberry PI, and many other related solutions that are engineered to a higher standard. In truly defensive implementations, both code and physical hardware are designed to operate defensively, selectively powering and monitoring subsystems, reacting correctly to power failures, attempts at tampering, etc. If we do not implement this for IT-2, we should at least research, and try to project and estimate what an industry-scale implementation would look like.

[6] This is not to say that it cannot be run 24/7; merely that it was designed as a hobbyist device, not a ruggedised industry device.

3. High Level Objectives and Goals

Software does not become "secure" by writing about security if developers then ignore it. Software could potentially be secured at a later stage by someone refactoring the code, or possibly by sandboxing insecure code in a virtualised environment. However, it is generally considered best to implement security in the development and DevOps processes (what some in the IT security industry call "DevSecOps", see Section 2.2.1), from the very beginning of the code being written. For example, running regular static code analysis would help to identify vulnerabilities early on; leveraging community/industry best practices by coding to standard interfaces and adopting approved security protocols etc. Securing software that was not written securely is likely to be even more time consuming than writing software securely from scratch, although the process of "refactoring and hardening" software is commonplace in IT projects, to take prototype and development software to a higher TRL, and this process could improve security as well.

Implementing security by design (section 3.1, below) takes more time and effort than just implementing the basic application functionality. As an example, consider deploying a customised server in a container: if it needs to run in a read-only container as a non-privileged user, logging to a separate logging service, and serving data only from a mounted external volume, the process of building the image and making sure the service works properly in the container is more time consuming.

Worse, projects are often tempted to prioritise basic functionality over security, not because they don't want to secure it, but spending time and effort implementing and improving functionality confer tangible benefits, as functionality can be easily demonstrated. The DevSecOps model addresses this security-later mentality by promoting security into first class requirements rather than as an afterthought.

In particular, we note that for IT-1, security was not prioritised, although some components did have security features built in.

3.1. Secure by Design

Although "Secure by Design" was covered in D2.4 [1], we can now expand the list based on the experiences from IT-1 and the additional plans for IT-2. It is also updated to take recent developments in the IT security industry into account.

- Security appropriate to the development cycle
    - Design, architecture - identify security risks and controls; and incorporate mechanisms, e.g. security gateway, standard protocols or design patterns etc. to deal with them;
    - Sprints - security questions raised: identifying necessary security tests in addition to functional tests;
    - Build Environment - standardise and automate the build and release processes, e.g. using source code repository and continuous integration tools;
    - Ensure insecure code is not left around, e.g. running static application security tests (SAST) during development to identify security vulnerabilities and remedy them without delay;
    - Appropriate tools for security development
        - E.g. automated analysis of source code;
        - DAST (Dynamic Application Security Testing);
    - All developers have a responsibility for securing the code (and the skills).

- Security appropriate to deployment/testing (DevSecOps)
    - Secure software development, management, build, deployment
        - Development and deployment tools used in secure mode if possible (Docker, supervisord, etc.);
        - Or if not, the exception is understood and assessed;
    - Ensure tests can be done safely (without triggering alarms), e.g. incorporate security tests like DAST, in the continuous build process;
    - Automating security testing;
    - Rapid reaction to software vulnerabilities (prioritised for instance using TLP[7] - Red/Amber/Green/White).
- Design, whenever possible, uses:
    - Widely used and well-maintained security libraries (e.g. OpenSSL);
    - Standard interfaces (e.g. GSSAPI);
    - Standard protocols/formats (i.e. standards from a Standard Defining Organisation, e.g. HTTPS, OAuth2, OIDC, JWT).
- Building application security on the common framework
    - Securing inter-fog, fog-to-cloud, and fog-to-fog communications via
        - encrypting messages using standard cryptographic algorithms (e.g. AES, RSA);
        - fTPM and TPM to sign and attest claims and messages;
        - well-designed APIs, e.g. CIMI[8] in mF2C;
    - Provide global identity management using standard framework (e.g. PKI);
    - Expose standard APIs to extend the security solutions implemented in the core layers up to the user application level (in other words, applications can build on mF2C security or implement their own[9].).
- Isolation. Whether we support application security directly on the mF2C security framework or not, application breaches should not affect the underlying platform, and should not affect other applications.

Physical security of devices – would need to be considered in any production deployment of edge and fog infrastructure. In particular, see section 4.2.4 for a discussion of the requirements of physical security in the mF2C use cases.

3.2. Privacy by Design

Like Secure by Design, Privacy by Design requires that software architects and developers take privacy in general and GDPR in particular into their design and code to ensure that sensitive data are handled appropriately and in compliance with GDPR directives. D2.4 [1] section 2.6.2 discussed the benefits of this approach in minimising privacy risks and building trust. D3.1 [11] and D4.1 [12] gave analyses of the security and privacy requirements of the Agent Controller (by architecture layers) and Platform Manager (by use cases) respectively.

In IT-1, we defined a security policy with three data protection categories:

- Public - for data not requiring protection

[7] https://www.us-cert.gov/tlp

[8] http://www.dmtf.org/sites/default/files/standards/documents/DSP0263_2.0.0.pdf

[9] Application security can never be wholly independent of mF2C security if applications use the same transport layer as mF2C (namely, the network that at the time of writing is being offered by the leader agent.)

- Protected - for data which needs to be integrity protected but is not confidential
- Private - for data which needs both integrity and confidentiality protection.

Note that both protected and private data would require sender origin authentication. However, the decision regarding the classification of each message needs to be taken by the sender, or, in practice, by the person implementing the code that calls the communications library.

Although unsophisticated, this policy proved sufficient for IT-1 – a more sophisticated policy might make clearer distinctions between the platform itself and its user data, and between applications and platform. Nevertheless, this design was deemed sufficient also for IT-2, although for control data communications, it would only be relevant between agents or between agents and third parties (also see Section 3.3.4 below). The agent implementation in IT-1 shown that intra-agent communications are sufficiently secured using the default private Docker[10] network: the recipient listens on a port configured in its container, and the sender sends data to a similarly configured port. As long as these are deployed together (i.e. on the same host), the Docker network handles the communication between the two containers internally. In IT-2, the focus is on protecting the confidentiality and integrity of communications external to an agent through encryption and secure transport protocols as well as to facilitate accountability through secure logging.

With respect to user application data, the security policy is also useful for ensuing GDPR compliance (see D3.1 Section 4.3 [11]). Though to enforce consent, additional metadata must be stored alongside the data as well as pro-actively enforced by the user application. The mF2C platform facilitates this requirement by providing relevant security functionalities like message token generation, cryptography and access control via the Security block (see D2.7, section 3.6).

The isolation principle (section 3.1) applies to privacy as well: the compromise of, say, an application, may lead to a data breach of the application data, but should not lead to a breach of data from other applications.

3.3. Deployment and DevSecOps

In D2.4 [1], very little attention was paid to containerisation technologies, the security features they provide and the security considerations they introduce. In fact, the deliverable only covers "virtualisation" in general terms and in terms of virtual machines, and not container technologies specifically, for the simple reason that the deliverable was written before the technology was sufficiently explored. We therefore must cover those in some detail here.

3.3.1. Configuration

The software components are frequently created in a multipurpose way or for different target environments. Adapting services to the target environment is solved by manual, automated, or distributed configuration, or possibly a combination of these. However, if a device or service were misconfigured, or the configuration was compromised by an attacker, the infrastructure would likely be non-functional or vulnerable to data breaches. This impact surface can be minimised with the following steps:

- Using pre-deployed devices with a trusted hardware component.
- Using pre-deployed devices with a trusted software component (for example, the app could contain the relevant trust anchors).
- Using certificates/checksums to assure that the software is authentic, possibly validated by hardware.

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

As discussed in D2.4 [1], the mitigations should consider the risks and costs of mitigation: attempting to protect against the more resourceful attacker would get increasingly expensive.

The deployment of the applications on top of the mF2C platform needs. The edge devices connecting to the platform could be deployed with software pre-configured by the company business that provides the services (as in all use-cases). Alternatively, security could be configured by the user who buys a device, e.g. by scanning a code with their phone. The first scenario with pre-deployed mF2C is better from the security aspect, as the mF2C updates can be installed on a pre-identified device. In the scenario where the user installs mF2C on the device, we must rely on the app store security.

Configuration of the mF2C platform is currently a fairly complex procedure which could potentially influence the security of the system. The platform services are built on containers which in turn are built from generic images, so once a containerised service is deployed, it needs to be configured correctly. Like the app, each individual containerised service must get its configuration from a trusted source. Also, the container deployment needs to be configured correctly. For example, some containers must deploy on the same host, relying on the internal Docker network for security. If an attacker could make the containers deploy on different hosts, or insert another container between them (like a classic man-in-the-middle attack), they could compromise the security of the infrastructure. Thus, there must be a means of ensuring that a system is correctly configured, in addition to verifying the integrity of the deployed software. In the near future, for some types of edge devices, this process may become hardware assisted, similarly to how the TPM on a computer motherboard can assist with checking the integrity of the computer hardware.

The same point, rather obviously, applies to the PKI. In particular, CA certificates need to be pre-deployed with Agents, in order to establish trust in the infrastructure, otherwise they would have no trustworthy means of establishing whether they are talking to their peers. This was already established in D2.4 [1], but the implementation was rather rigid, assuming everyone was using the same PKI. IT-1 had two PKIs, the fixed infrastructure and the slightly unfortunately named "untrust" PKIs, meaning the fog agents. In general, each deployment must have (at least) two (independent[11]) PKIs, so these must be easier to set up correctly.

3.3.2. Container Security

It is often harder to build secure Docker images than to build the default, where everything runs with the most liberal permissions. For example, deploying a service that runs as root in the container is much easier than one that runs as a non-privileged user, because the non-privileged account needs to be created in the Dockerfile and the container's filesystem needs to be configured to give suitable permissions (read, execute, write) to this account. The unprivileged account may be the more secure choice for containers exposing ports to the outside world, even if the container's root account is unprivileged on the host, because a compromise of the service would only affect the areas that are writeable by the service as opposed to the areas writeable by root. A similar or alternative container security measure is to make the container read-only, so no malicious intervention can modify the code in the container, but this obviously requires that the container's state be managed elsewhere, e.g. in mounted volumes or over network connections to other containers.

Another consideration is that containers share kernel with their hosts, so any vulnerabilities in the host kernel is inherited by the running container. Moreover, as the container is typically used to run services that listen on ports, it makes sense to make them as secure as possible, particularly if they are exposed via the host to the Internet/Fog.

[11] Technically they need not be wholly independent, but the very basic PKI such as the ones we use here – one root certificate issuing end entity certificates for each PKI – is sufficient for our purposes.

For mF2C, the container technology being used is Docker[12], which describes its security as being a result of the security level in the following four areas (quoted from the official Docker Security page):

- the intrinsic security of the kernel and its support for namespaces and control groups (cgroups);
- the attack surface of the Docker daemon itself;
- loopholes in the container configuration profile, either by default, or when customized by users;
- the "hardening" security features of the kernel and how they interact with containers.

When it comes to the intrinsic security of the kernel, given that in mF2C, agents are installed in user devices where the Docker client already exists (as an application requirement), the kernel security will be managed by the user itself and the associated device. This means that if the user's device kernel is somehow compromised or misconfigured, the resulting vulnerabilities will be propagated to all Docker containers, including the mF2C agent. This also applies for the Docker daemon itself which needs to be running on the user's device in order to install the mF2C agent. Given that the Docker daemon execution requires root privileges, any untrusted access to the device's Docker daemon (either through an unprotected API or directly through escalation of privileges inside the device itself) could unwillingly grant management and tampering access to all containers running in that device, including the mF2C agent.

A recommendation for further improving the mF2C agent security in the future would be to examine the hosting kernel prior to the installation of the agent. Nevertheless, in mF2C, it is assumed that the Docker daemon is properly configured, and namespaces are supported, providing a complete isolation between running containers, and allowing the containers to be given its own network stack. In this case, since the mF2C agent is deployed in the form of Docker Compose application, all containers will belong to the same network. This makes the mF2C agent completely isolated from other processes running within the device, while still sitting on a bridge interface which, from the outside, makes the agent look like a cluster of physical machines running on a private sub-network.

Since the execution of container-based services is supported in mF2C, one of the security goals of the project is to make sure that users cannot exploit the attack surface of the Docker daemon on mF2C workers (agents which are executing a service). One example of such security protections is to make sure the mF2C's Lifecycle Manager does not flood the worker's disk space by blindly pulling Docker Images every time a container-based service is scheduled. Another example comes from the need for the mF2C agent to make use of certain Linux[13] kernel capabilities in order to have access to root-only areas, like CAP_NET_ADMIN. In this case, the mF2C software needs to ensure a proper use of these kernel capabilities, otherwise the network of the hosting device might get compromised.

Control groups (cgroups) are also taken as an added-value in mF2C as they provide a security barrier against attacks that aim at causing service disruption and possibly system unavailability. Even though cgroups are taken for granted as an intrinsic feature of Docker Containers, mF2C is also implementing a Sharing Model feature, with add extra resource constraints.

Finally, the container configurations and user customizations are related with the Docker Image building procedures adopted by mF2C, and the way these images are then used when deploying the mF2C agent. All mF2C Docker Images are publicly hosted in Docker Hub[14] and thus accessible by everyone. A recommendation for future security improvements in mF2C is to enable Docker Content

[12] https://www.docker.com/
[13] https://www.kernel.org/

[14] https://hub.docker.com/

Trust (DCT)[15] for the deployment of the mF2C agent, which means Docker Images would need to be signed. Apart from that, in order to ensure that the mF2C Docker Images are not prone to security vulnerabilities, the following image building conventions are taken:

- the amount of Docker layers constituting an image should be kept to a minimum;
- use trusted parent images;
- only expose ports which are necessary and for which application security measure have been implemented.

Docker Images are always rebuilt whenever there are updates to the mF2C components, no matter how small. Security updates are always treated as a major update.

3.3.3. Container Deployment and Availability

For the fog-to-cloud infrastructure to be secure, it is necessary to ensure that services and agents can be deployed correctly and will implement the security framework once deployed. Availability of Docker-based services is usually based on ensuring that containers are either stateless or at least do not retain their state on the host, so can be freely moved to other systems or replicated if the need arises. In mF2C IT-2, we do not impose this requirement but merely recommend that the person building the image takes this into consideration.

3.3.4. Implementing Data Security

As mentioned in Section 3.2, in the mF2C architecture, different types of data are transferred hierarchically between layers. We classify the different types of data into three categories in line with the mF2C Data Security Policy (see Section 3.2):

- Public – Data are accessible to all mf2c users. No security needs to be applied.
- Protected – Data must be integrity protected. In this case, integrity is provided by checksum or signature and guarantee that data is not altered. Protected data are accessible by all mf2c users.
- Private – Data must be both confidentiality and integrity protected. Access controls, authentication, checksum or signature and encryption need to be enforced for Private data.

Control data (i.e. data pertaining to the operations of the mF2C platform), in particular, must be given the appropriate protection, see D4.2 for further discussion.

There are two remaining questions. First, for private data, encryption must be targeted to the recipient. This means that:

- The public key of the recipient must be known to the sender, and
- Messages will need to be encrypted N times if broadcast to N recipients.

A possible solution is to employ a trusted intermediate who can decrypt the message and re-encrypt it, e.g. the Leader (see D2.7, section 3.1.1). Even then, there must be metadata to enable the Leader to decide who the recipient is. Also, we assume that the Leader has the public keys of all the recipients. By extension, the trusted intermediate model would work also in a hierarchical Leader model with the message being communicated up and down as appropriate, provided that a Leader is capable of deciding who is the sub-Leader (if there is a Leader in each layer) that can act as a trusted intermediate for a given recipient (Agent) in that specific layer. If the applications have access to the same sets of keys, then they could potentially compromise the control traffic, so it would make sense to introduce a different layer of protection by, having a parallel PKI for applications, or some similar type of

[15] https://docs.docker.com/engine/security/trust/content_trust/

protection. In particular, applications need not be constrained to the architectural layout of the platform.

Another seemingly simpler solution would be to use symmetric encryption. The message would be encrypted once using the symmetric key and the key then encrypted with the target recipient's public key and send or broadcast to the recipients. However, key management remains a serious issue and it would be a serious challenge to make such a solution work, and scale.

The other open question is how user applications make use of the mF2C platform communications, if they want to. As mentioned above, it is expected that the applications use the mF2C transport layer, so they will need a certain minimum of interfacing with mF2C security; and the full security should certainly be available to the application developer who wishes to make use of it. However, some separation of duties between the applications security and mF2C platform security is necessary, as mentioned above, which could be implemented using mF2C code but changing the keys: in other words, agents could provide features for applications to communicate (but see also section 4.4), but with different keys. Alternatively, of course, applications could implement their own security.

3.4. GDPR

The EU General Data Protection Regulation (GDPR) [13] came into force on 25 May 2018, replacing the Data Protection Directive 95/46/EC [14]. One of the principles of mF2C is to address security and privacy considerations while dealing with personal data; thus, GDPR considerations are taken into account from the design phase.

First of all, it is important to define the roles defined by the new regulation that are covered by the project. Two of the main roles are the Controller, who decides how and why data is processed, and the Processor, who executes data management. The controller will be represented by the consortium members; the processor will be the providers of the mF2C framework as processing is fully automated within the architecture.

Taking this into account, the consortium will be in charge of ensuring that the key pillars of the GDPR are accomplished, mainly:

- Lawfulness, fairness and transparency, ensuring the users' rights about consent and object;
- No repurposing of data in the context of the project, collecting only that data required for the processing purpose;
- Ensure accuracy, up-to-dateness and users' rights to correct data and to be forgotten;
- Grant confidentially, integrity and security of all processed data.

In order to be able to implement these principles, goals and requirements were taken into account in IT-1 for the design of the security framework, if not for its implementation, but should be implemented in IT-2 as a core functionality of mF2C. For this reason, all Personal Identifiable Information (PII), mainly coming from the use cases, will be categorized depending on their level of sensitiveness, e.g. IPs, location, personal profile information or any other kind of identifiers. This work will be performed by each use case owner, and properly documented in the corresponding deliverables, with the assessment of members of the consortium in charge of security tasks. If needed, legal advice will be requested for dealing with a specific topic.

As a preliminary step, a set of initial requirements has been identified as a minimum to be covered within project developments. This list will be updated or extended when needed, and (as of this writing) needs to be made available to the end users through the mF2C portal. The list is as follows:

- Identification of the potential risks of all technical activities related to data protection (see D2.4 [1], section 2.1);

- Ensure data protection applying different techniques, such as encryption, tokenization and/or pseudonymisation, in order not to make it identifiable by a third party (D2.4 [1], section 2.7;
- Secure storage (in house/cloud) of data, processed data and historic data (D2.4 [1], section 4.1);
- Grant minimum user rights: access, rectification, modification and removal. This must include the right to be forgotten that affects not only data but also metadata;
- Apply restrictions to data portability and processing based on the limitations agreed with the user;
- Establish a specific deadline for data removal when it is no longer necessary and provide automated mechanisms for doing so;
- Develop and provide a clear informed consent to be signed by the end user who will be accessing to any of the use cases' applications. A Data Governance Plan will be developed in the context of the project and documented in the context of management tasks
- Data access must be monitored to prevent breaches and noticing violations. Specific automated mechanisms must be implemented for solving it and notifying the incident to the affected users within 72 hours;
- Standardised data format and high quality of all stored data;

Maintain auditable record on the types of data processed and how they are processed.

4. Security in the Architecture

This section is about security requirements as they relate to the different architectural layers and applications. The reader will note that D2.7 describes the architecture itself, and the location and (expected) functionality of the security components within the architecture.

4.1. Supporting Use Cases Security

The three Use Cases present a network architecture with four layers: 1 cloud layer, 2 fog layers and the edge/IoT. In all Cloud and Fog layers, devices run the mF2C agent, so they use the built-in security/privacy capabilities of mF2C to guarantee the security and privacy of processed data. See also section 4.4.

Notably, the transport layer (e.g. Wi-Fi) is shared across the fog cluster, so applications would likely use this as well and will need to integrate with the security framework at least for this purpose.

4.1.1.UC1 - Smart Infrastructure

In UC1 (Smart Infrastructure), there are five device groups that run in the edge/IoT communicating:

- LoadSensing with the Tiltmeter - it logs the inclination of the building and forwards the information;
- Gateway - it treats this information and reports an alert that triggers intervention actions in case a critical threshold is reached;
- Location devices - they identify which worker is located closest to the alert reported;
- Jammer Detector - this is launched when a communication error is reported, in order to confirm the origin of this error;
- Mobile devices - these report the alert to the workers and actors located nearby;
- Emergency vehicles and traffic lights - all are involved in the optimised global intervention.

Besides the above, there is the cloud service that runs the Grafana[16] monitoring software, plus worker nodes in the fog.

The above devices communicate over different network media with different levels of security. Communication over Ethernet can be secure – if it's a private network – whereas those over Wi-Fi, LoRa, Bluetooth, and cellular networks might need to have security implemented and integrated with mF2C security on top of those already built-in to these communication protocols. In case of jamming attacks, the most reasonable contingency action would be to communicate across different networks; using those that were not affected by the interference situation.

In terms of Security Policy, the control data sent from, for example, the Jammer Detector to the Gateway, needs to be at least Protected. In particular, it makes sense to implement message origin authentication (in practical terms via a signed checksum). It also makes sense to ensure that only legitimate alarms can be raised, so the location agent can be used to detect which actor is the closest to where an alarm originates, and report on its validity.

By default, no personal data is handled, so protection levels Public and Protected are sufficient. The information about the location of the closest actor should be presented in such a way so that no personal data are revealed. If this were not possible, the data should become Private.

In terms of the Platform Manager (PM), we expect that no further protection requirements arise due to the absence of Private data. However, as the UC specifically includes an availability attack – by an attacker with a jammer – one has to be careful not to give the attacker information that they could use against the infrastructure, so the use of PROTECTED or PRIVATE data within the PM should not be fully excluded.

4.1.2.UC2 - Smart Boat

In UC2, of the Smart Boat, there are six different devices that run in the edge/IoT or fog communicating:

- Laptop or Intel NUC[17] - these were used only for IT-1. Both devices work in the fog layer; they run the mF2C agent and the Smart Boat software that currently cannot be executed on Raspberry PI;
- Raspberry PI - this is the core Fog device that aggregates data from sensors via the Boat Monitor. It processes and stores the derived data. It also runs a backend for Android/web access;
- Network Component in the fog layer - this supports communication between the on-boat components, between other boats nearby and the cloud;
- Boat monitor in the IoT layer - this collects data from the sensors and forwards them to the Smart Boat application. It has some processing capabilities;
- Two additional sensor hubs in the edge/IoT - these are physically connected to the sensors and read their values on a continuous basis;
- Android smartphones in the edge/IoT - these are used to access reporting and the latest forecast information.

Besides the above, there is the cloud service that is connected to the Network component using available communication media. Its role is to offer maritime services to all boats within communication reach.

[16] https://grafana.com/
[17] https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html

The challenge here is on the different connection scenarios that could happen between boats. Devices installed in boats can communicate over different wireless protocols, namely Wi-Fi, LoRa and cellular 3G/4G network, depending on the distance from the coast and between other boats.

All these media require security setup integrated with mF2C built-in capabilities.

In terms of the managed data, all data included in the present use case are personal and sensitive data as they expose current position of the persons on the boat. For the current case, we have a special agreement with each tester of the device, who consented to the collection of personal data when the tester is traveling with the device. The consent declaration is GDPR-compliant and is created by XLAB's internal Data Protection Officer (DPO) with the collaboration of Smart Boat development team.

4.1.3. UC3 - Smart Fog Hub in Airport

In UC3, the Smart Fog Hub in Airport, there are four different communicating devices that run in the edge/IoT or fog:

- NuvlaBox[18], a dedicated appliance that works in the fog layer - it runs an mF2C agent and provides real-time computing and storage resources to the edge elements. It communicates with the Airport system to get and dispatch all flight related events to interested travellers. Airport administrators can access the dashboard to manage all relevant information, and monitor in real-time position of travellers and other reports;
- Laptop node that works in the fog layer - it runs an mF2C agent providing additional processing capabilities, offering load balance support in case of increasing processing requests from the edge. It was used only for IT-1, as currently the mF2C agent cannot be executed on a Raspberry PI;
- Raspberry PI that works in the edge/IoT - it acts as access nodes, providing session management, hotspot capabilities for location processing and fast response to the edge devices;
- Smartphones, used by travellers, work in the edge/IoT - these are connected to the access nodes with Wi-Fi, and interact with the system using an Android app.

Besides the above, there is the cloud layer based on an OpenStack instance, wired connected with the fog layers. This provides scalable computing power for machine learning algorithms used for the recommendation system.

In terms of communication media, the Cloud and Fog nodes are wire-connected and different security mechanisms can be setup, and mF2C security features can be used. In the fog area, instead of the smartphones and access nodes communicating through Wi-Fi, it is possible to use other media if necessary to have security setup integrated with mF2C features.

Speaking about managed data, the use case uses only a traveller's position paths, and his/her device's MAC address as the user reference. Since there is no need to have it in clear text, we hash the MAC address to anonymous it. For this reason, in the use case only requires Public and Protected protection levels.

4.2. Other Operational Security

Given the very different nature of IoT compared to traditional network and data centre technologies, any adoption of IoT platforms requires to think long and carefully about some key security aspects.

[18] https://sixsq.com/products-and-services/nuvlabox/overview

First of all, IoT systems are dependent on the operations of a large number of inexpensive devices and software instances; this heterogeneity increases the overall complexity of the system and the network, so the need for more frequent updates or changes or replacements that could impact the system security. These changes/replacements could also bring misconfigurations in the network.

Also, management of certificates is pretty relevant as certificates represent trustworthiness and are the pillars of the IoT security. Only devices with a valid certificate will get access to the system, thus certificate expiration dates should be tracked and properly managed to avoid the risk of having some critical devices offline, the operational issue being a trade-off between the lifetime of the certificate and the need to frequently rekey the agent. A client can request a rekey of its expiring-but-not-yet expired certificate, and the authority is supposed to check prior to the issuance of the rekeyed certificate that the client is still entitled to have a certificate and that the data asserted therein is still valid, but rekeying is potentially a substantial overhead on the client.

Security monitoring of a multi-layered architecture with strict correlation between different components is another critical point, security events monitoring needs to analyse and even predict exploitation attempts to enable rapid respond to minimize the impact; thereby preventing the incident propagating to other nodes.

4.2.1.Detecting Incidents

In deliverable D2.2 [15], we explained the emerging role of Machine Learning (ML) and Artificial Intelligence (AI) approaches in the field of cybersecurity, offering alternative solutions to many interesting research challenges.

In the case of incident detection, ML has proved to be particularly interesting in solutions dedicated to preventing possible jamming attacks. In the case of mF2C system, as in other fog-to-cloud systems, attackers can affect the availability of the different mF2C devices and make these nodes inaccessible to the users by jamming or flooding requests.

Fog-to-cloud oriented architectures, such as mF2C provide an opportunity in offloading security functions to different mF2C devices, and the new technologies that enable it should be examined. The data collected locally in mF2C nodes, from smart device or sensors, can be used to detect anomalies and to model system behaviour patterns over time as backgrounds to detect incidents. While the mF2C system is not application data oriented, the use case owners can take advantages of its distributed infrastructure and leverage different ML algorithms on local data captured at each mF2C node to detect attacks.

The mF2C system can benefit from the results of the research done in examining the effectiveness of various ML techniques including artificial neural networks, support vector machine, logistic regression, K-nearest neighbours, decision tree and Naive Bayesian, etc. for detecting different anomalies such as jamming or DDoS attacks etc.

More recently, the focus is in using different deep learning methods for cybersecurity purposes. Researchers in [8] proposed a distributed deep learning scheme for cyber-attack detection in an F2C architecture for IoT applications. In [16], a “long/short term memory network” (LSTM) with self-learning capabilities is used to extract the baseline patterns and to detect anomalies which may represent attacks in similar distributed F2C architecture.

The mF2C project should investigate the above techniques to decide whether ML is a viable option for enhancing mF2C security. However, using ML to detect unusual behaviour is still an open area of research with different opinions on the advantages of this approach. Going further and using ML to *react* appropriately to the incident is an even more ambitious goal, although a simple precautionary reaction of closing networks and ports would be relatively straightforward. We discuss this goal further in the following section.

4.2.2. Automated incident handling

As the threat landscape evolves and expands, handling the ever-increasing number of security incidents promptly and automatically has been a major challenge for traditional IT systems. In dealing with huge IoT networks with a large attack surface, it is increasingly critical to adopt automated incident response processes.

Security Orchestration, Automation and Response (SOAR) is a new category of security tools defined by Gartner [17] as a tool that:

- Collects security threat data and alerts from different sources;
- Enables incident analysis, triage and prioritization, both automatically and manually with machine assistance;
- Defines and enforces a standard workflow for incident response activities;
- Encodes incident analysis and response procedures in a digital workflow format, enabling automation of some or all incident responses.

SOAR tools work closely with SIEM and leverage the integration to:

- Receive alerts and additional security data to identify security incidents;
- Draw in data required for analysts to further investigate an incident;
- Assist analysts in proactive incident response and threat hunting, which relies on querying and exploring cross-organization data.

According to Gartner, the next-generation SIEM solution will include a native SOAR component to fully automate case management and threat defence.

The starting points to automate incident handling are:

- Continuous event collection and monitoring of IoT and Fog nodes;
- Threat intelligence, to detect malicious actions as early as possible and adapt defences accordingly.

All nodes need to generate events to be monitored for anomalies. These events are collected by fog nodes and transferred to the cloud for long retention time. In the mF2C Project, the threat intelligence with the use of ML algorithms can be distributed on all fog nodes (subject to capability) for quicker response time and better protection [18]. Each node would act both with a proper intelligence or using "collective" intelligence coming from the cloud.

The decision-making process would orchestrate and automate responsive actions (reactive or proactive) based on environment states and assessment risk. At the same time, all events are stored on the cloud, enabling sharing of logs and threats, and analysis, which is essential for detecting and managing simultaneous attacks across systems or across fogs. All results, including deep analysis, correlation and the Indicators of Compromise (IoCs), can then be copied to all fog nodes for real-time usage and execution of best counter-measures.

This approach could enable:

- Reduced dwell time, from compromise to detection, currently the average is about 24 hours [19];
- Fast Containment and Incident Response, e.g. through selectively firewalling or segregating networks;
- Tracking and Auditing capabilities for forensic requirements.

The ML assisted automated orchestration [20] of machine-driven actions on IT systems, as part of a response to an incident, could reduce the overall response time and ensure a better protection to the distributed system. This could be even more valuable since we are approaching systems with a huge (increasing) number of nodes, especially at the edge, thus the capability to manage the incident response would make a big difference. Traditional Security Operations Centre (SOC) approaches with human analysis and decision making would be too-costly and slow, compared to this proposal.

Great care should be taken here to weigh the risks between allowing an attack to filter through the system against the risk of inadvertently closing a legitimate but spiky service use – e.g. a UC1 emergency.

4.2.3. Complex Event Processing

The SOAR category, as defined by Gartner, is based on the concept of Complex Event Processing (CEP). This is event processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances. The goal is to identify meaningful events, such as threats, and respond to them as quickly as possible[19].

These events can be generated in various layers of the IoT ecosystem, or in different layers of a complex system. The CEP helps by analysing and correlating these events, inferring complex events such as coordinated attacks, and treating them accordingly.

CEP relies on a number of techniques, including:

- Event-pattern detection
- Event abstraction
- Event filtering
- Event aggregation and transformation
- Modelling event hierarchies
- Detecting relationships (such as causality, membership or timing) between events
- Abstracting event-driven processes

Most successful applications of CEP in industry make extensive use of Machine Learning and Artificial Intelligence algorithms, and include the detection of credit-card frauds, business activity monitoring and security monitoring.

In terms of open source software, Apache Flink[20] is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams, specifically designed to run in all common cluster environments, with good performance and scaling capabilities.

Flink can seamlessly manage any kind of stream of events, from credit card transactions, to sensors measurements, system logs or any user interaction with a website of mobile application.

Data can be processed as unbounded or bounded streams.

1. **Unbounded streams** have a start but no defined end. They do not terminate and provide data as it is generated. Unbounded streams must be continuously processed, i.e., events must be promptly handled after they have been ingested. It is not possible to wait for all input data to arrive because the input is unbounded and will not be complete at any point in time. Processing unbounded data often requires that events are ingested in a specific order, such as the order in which events occurred, to be able to reason about result completeness.

[19] https://en.wikipedia.org/wiki/Complex_event_processing
[20] https://flink.apache.org/flink-architecture.html

2. **Bounded streams** have a defined start and end. Bounded streams can be processed by ingesting all data before performing any computations. Processing bounded streams does not necessarily require ordered data because a bounded data set can always be sorted.

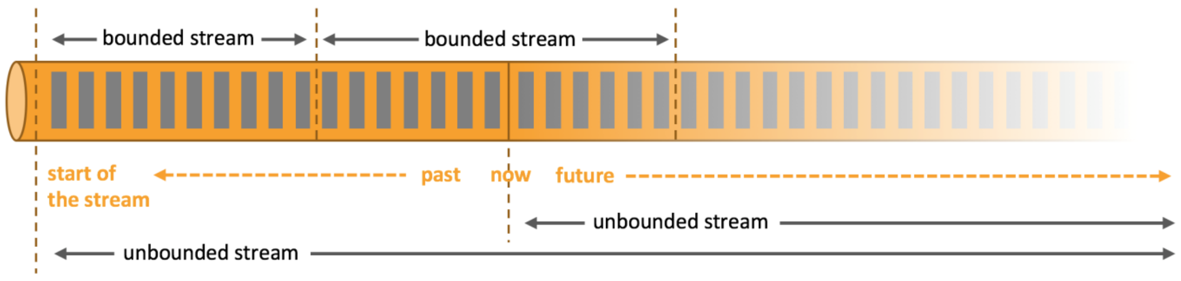


Figure 1: example of bounded and unbounded streams – from Flink

4.2.4.Physical Security of Devices

The mF2C project is a software focused project, it is not our objective to develop tamper-proof or tamper-evident hardware. Nevertheless, we should at least have a perspective on the physical security of devices, as it is a major concern in many practical deployments. From the perspective of the use cases (sections 4.1.1-3), UC3 could rely fully on the security of the environment – the airport. For UC1, there is a concern that people may tamper with the edge devices, but this is mitigated at least in part by not relying on data from any single device. The security of the UC2 (Smart Boat) will need additional research: a solution may be out of scope of the mF2C project (tamper proof devices), but it would make sense to have some suggestions and maybe test a few simpler solutions (see section 2.2.2 for additional background).

4.3. Security and Privacy Requirements

In this section, we briefly discuss requirements on agent security.

4.3.1.Agent Security

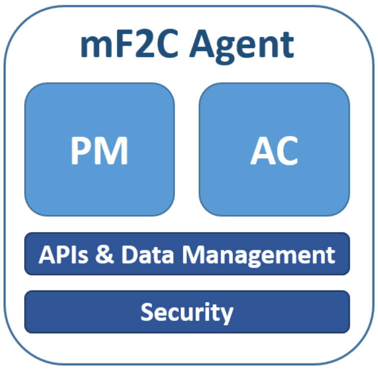


Figure 2: Agent security

Figure 2 shows the updated architecture (at a very high level; see D2.7, section 3, for the details of the agent architecture) of a full mF2C Agent which is composed of two main blocks: the Platform Manager (PM) and the Agent Controller (AC). In addition, we have a communication and data management layer comprising CIMI and DataClay which provides cross-cutting functionalities. Last but not least, the Security layer and details of the proposed security implementation for AC and PM are provided in D3.2 and D4.2 respectively.

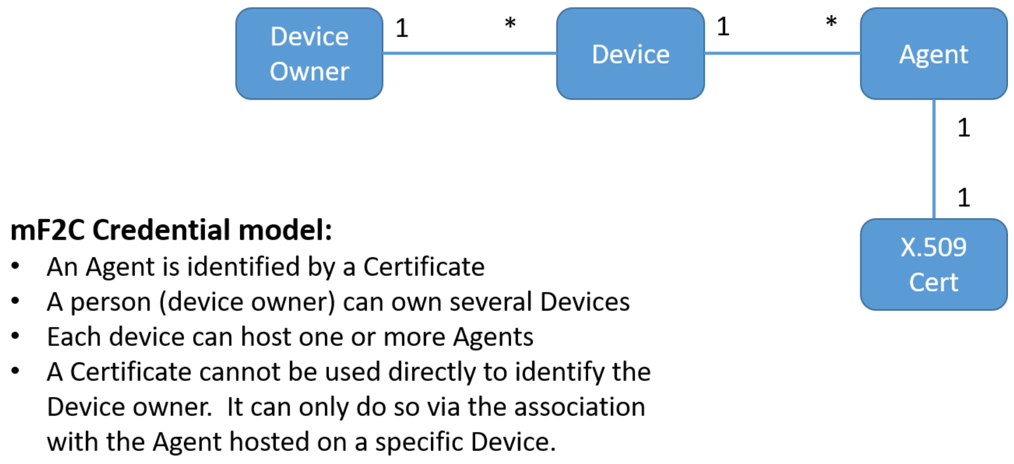
4.3.2. Leader Agent

The leader agent in the mf2c architecture is the fog area manager as a parent to its children such as agents and IoT devices. The leader security requirements may be such as below:

- A leader agent must be selected through a secure election process such that an imposter cannot usurp the leader;
- The leader must discover all the agents in its area in secure way;
- A proper mutual authentication between leader and agents must be carried out;
- All the information such as control data, IPs, resource information, etc. must be processed and transferred in a secure way between a leader to its agents and backup leader. The data must be encrypted to avoid attacks such as eavesdropping, man-in-the-middle and spoofing, etc.;

A leader can act as access controller to its agents to avoid unauthorized user getting access to the mf2c system.

4.3.3. Identity



mF2C Credential model:

- An Agent is identified by a Certificate
- A person (device owner) can own several Devices
- Each device can host one or more Agents
- A Certificate cannot be used directly to identify the Device owner. It can only do so via the association with the Agent hosted on a specific Device.

Figure 3: mF2C credential model

Identity is pivotal to a PKI; in mF2C, we need to be able to uniquely identify an Agent through its X.509 certificate to support access control decision according to the privilege associated with this identity. Figure 3 shows the relationships between a certificate and the different roles in mF2C. The main requirement on identity in mF2C is unchanged from IT-1: namely, that agents be identifiable with individual cryptographic credentials. Certificates are required for all of the following purposes:

- To authenticate the client in a secure connection ("secure" is explained in section 4.1);
- To authenticate the agent in incoming secure connections (server mode); here it is important that the agent is named in the certificate so the client can check the server's identity (cf. RFC 2818, section 3.1);
- To authenticate the sender of a message (sender origin authentication), which also provides message integrity;
- To decipher decryption keys for messages classified as PRIVATE and encrypted to the agent (with a symmetric key).

As it is generally good practice not to sign anything with a certificate used for authentication, it follows that each agent may need two certificates. (In addition, agents may need application-specific credentials as well, as described in section 3.3.4.)

4.3.4.Security of Communications

Authenticating a client and server to each other, as required in the previous section, can be done with socket-based security, using TLS, or transparently wrap the packets being sent, e.g. with GSSAPI. In either case, "secure" means client and server are authenticated to each other; an attacker cannot interfere with the messages without a high likelihood of detection, and, optionally, the communications channel can be encrypted.

In contrast, the last two bullet points in section 4.3.3 require message-based security. Messages can be passed from one agent to another (e.g. through a Leader to a service in the cloud).

Communications are essential not just for applications but also for components in the mF2C platform, since the platform is distributed. However, it does not necessarily follow that "secure everything" is the best approach, as we discussed in D3.1 [11]. In this section, we look at different communications patterns for IT-2 below.

4.3.4.1. Inter-Agent Communications

In the updated Agent architecture, all communications flow through the new APIs layer (see D2.7 Section 2). At the core of this layer are CIMI and Data clay. CIMI is an open standard for managing resources within an infrastructure and DataClay is a data management component.

In mF2C, CIMI provides a single-entry point to incoming communications and the mF2C management API, both for mF2C users and the internal mF2C components. CIMI supports the REST architecture style using HTTP for communication and this design influences the security requirements for inter-agent communication. These requirements are:

- A secure REST service must only provide HTTPs endpoints to ensure point-to-point security to protect authentication credentials in transit;
- Mutual authentication between the Agents (client and server mode);
- Adopt standard protocols for authentication, e.g. OAuth2, OIDC, etc.;
- Validate input and Content-Type to deter code injection/execution attacks;
- Restrict HTTP methods by applying a whitelist or access control list to make sure that the caller is authorised to use the incoming HTTP method on the target resource;
- Protect CIMI endpoints by firewall rules or access control lists;
- Use standard format of security tokens, e.g. JWT, to facilitate access control decision and protect their integrity by either a signature of a message authentication code (MAC);
- Support message-level security:
    - Encrypt private data to protect confidentiality if the endpoint is not the target recipient;
    - Sign sensitive data to attest integrity and protect against malicious tampering;
- Logs security incidents like token validation errors to help detect attacks, e.g. through log analysis. But avoid logging sensitive information like passwords, security tokens etc. in clear text.

Apart from the CIMI managed inter-agent communications, DataClay performs direct data replication between the Leader and Backup Leader Agents which by-passes the Reverse Proxy (see D2.7, Section

3.6.3). DataClay uses gRPC[21] framework for data communication over HTTP/2[22]. To protect this communication, mF2C should aim to:

- enforce point-to-point channel security using the SSL/TLS protocols;
- authenticate and authorise callers using mF2C issued X.509 certificates or access tokens.

The first requirement is supported by gRPC and the second could by implemented using the Credentials Plugin API built-in to Protobuf[23], the gRPC tool[24].

4.3.4.2. Fog-to-Cloud Communications

In general, it would be unwise to give a fog device direct access to the Internet. If it needs to access services in the cloud, it should be possible to limit its access to devices that have authenticated and limit access to the specific communications required. In IT-1, the CAU implemented a gateway that enabled a client in the fog (i.e. an agent) to get certificates from the CA in the cloud (see D3.2)

One obvious open question, that was insufficiently covered in IT-1, is how applications communicate with services in the cloud. Applications would be subject to the same constraints as agents, if they communicate over the same networks, because there would obviously not be a means of blocking unauthorised agent traffic while letting agent traffic through unhindered, and if there were, it would be a security risk because an application might have security vulnerabilities.

Thus, at least some of the mF2C communications security MUST be applied to applications. In particular, controlling this information becomes important if we implement the ability to react to a security incident to block the traffic of a compromised fog node. Architecturally, D2.7 - or D3.2, perhaps, must decide where this capability sits and how it is implemented - are messages communicated through the leader? Or can other gateways be set up?

4.3.4.3. Edge-to-Fog Communications

D3.1 foresaw the communications of edge devices with fog devices through two distinct means. Either the edge device had the ability to communicate securely with the "nearest" fog node (i.e. with the data at PROTECTED level at a minimum), or it had a private communications channel. As foreseen in section 2.2 of this deliverable, it would make sense to investigate the range of devices capable of implementing these security measures, if the device itself could be trusted, but the communications channel into the fog could not.

4.3.4.4. Fog-to-Fog Communications

Although this case does not currently arise in the mF2C platform itself, it could arise in the use case. If a node in a fog cluster needs to communicate with a node in a different fog cluster, then they cannot communicate directly with each other: there is no transport layer link between them. It follows that either a gateway must be set up - similar to the fog to cloud gateway implemented in IT-1 by the CAU - to connect the two clusters - or messages must be routed through the cloud. For the former, a means of routing messages through the Internet at least must be implemented; for the latter, services must be deployed in the cloud to enable this communication. In particular, two points need consideration. The first is how the message is secured appropriately if it is classified as PRIVATE, as we must assume

[21] https://grpc.io/

[22] https://http2.github.io/

[23] https://opensource.google.com/projects/protobuf

[24] Security for DataClay will need to fit with the underlying communications which in this case is gRPC; contrast this with COMPSs which uses network sockets where security is implemented through GSSAPI.

the sender is able to encrypt the message with the recipient's public key, and socket-level security is not sufficient (i.e. the message itself must be secured.) The second point is how to route messages to the recipient in the remote fog cluster. There is no global addressing scheme, and the potential cloud service fulfilling this task would not necessarily know which fog cluster contained a specific agent. One means of solving this problem is to have a central message queue in the cloud, but this, too will need some thought.

4.4. Architecture Layers

Security and privacy mechanisms used in mF2C depend also on the specific layers, namely cloud, fog and IoT. The different measures and requirements defined for each layer were already included in D2.4 [1], and the security components for each layer are discussed in D2.7.

4.4.1. Architecture Layers – Cloud

The Cloud layer security comprises of securing cloud resources and providing tools to monitor and manage the security events. Securing of cloud resources is achieved through security by design approach while for the alerting and event management SIEM tools like Alien Vault[25], OSSIM[26] and OSSEC[27] could be used.

If we choose to implement SIEM, it will likely have one instance in cloud, where the main database and dashboard backend will be deployed (Figure 4). There, all the data aggregated from the specific sensors integrated in the mF2C modules will be stored and processed. The data collected from sensors deployed in fog agent will be gathered in SIEM through the secure MQTT channels. Further details and options for a SIEM implementation should be included in D4.2.

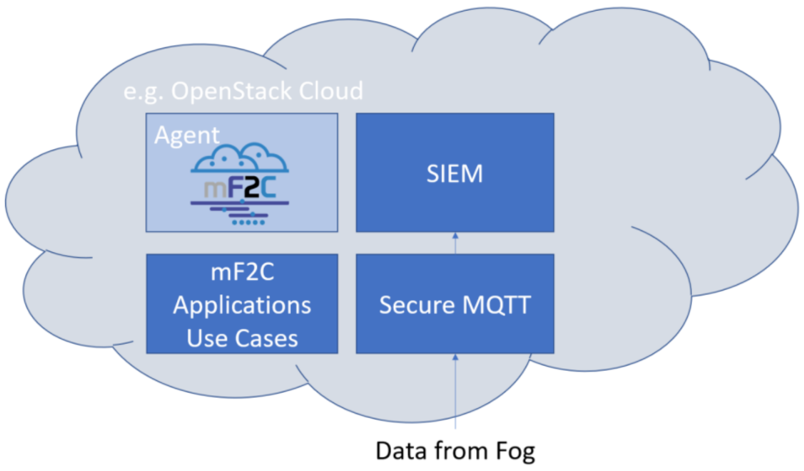


Figure 4: SIEM high level architecture

The SIEM dashboard will include insights of reports, assets, alerts, and log monitoring. It can also provide intelligence from the machine learning part of the detecting incidents.

[25] https://www.alienvault.com/products/ossim/download

[26] https://github.com/alienfault/ossim

[27] https://github.com/ossec/ossec-hids

4.4.2. Architecture Layers – Fog, Edge

The fog is a hierarchical organisation of agents. The leader agents are like parent nodes in a tree and presents the link towards the upper layer closer to the cloud for each ancestor agent. The security messages and monitoring will follow the same path from the bottom agents to the cloud, where all data will be processed, decomposed and presented with machine learning tools and dashboards.

To efficiently monitor the state of the edge device and the agent components itself, each component will need to provide a sensor attached to its logs and events (Figure 5). The MQTT communication should use message=based security – see D3.2.

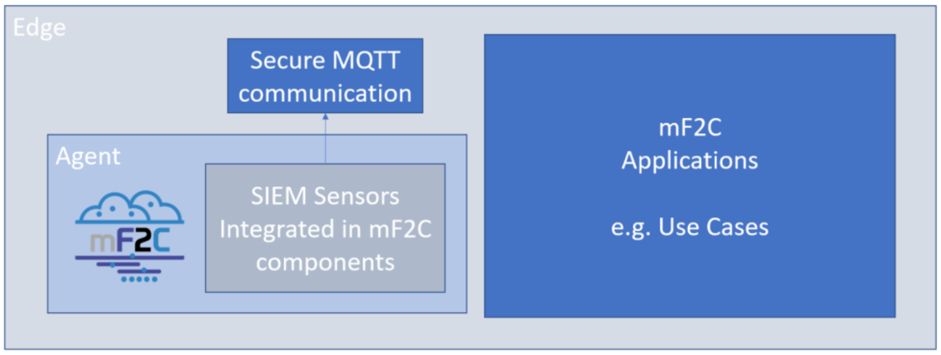

Edge
Secure MQTT communication
Agent
mF2C
SIEM Sensors Integrated in mF2C components
mF2C Applications
e.g. Use Cases

Figure 5: Gathering SIEM data at the edge

4.4.3. Layers – Sensors

The sensors and IoT devices do not necessarily have the ability to self-monitor and provide extra services to communicate this data to upper layers. Instead, the agent responsible for each sensor will monitor the current status of the sensor and include this information into the security monitoring and SIEM workflow.

One of the emerging security aspects is to retain records of the source and integrity of sensor data communicated over untrusted networks or relayed through untrusted participants. mF2C has an opportunity to apply the mF2C security model (summarised in section 3.2) to sensor data by exploring the options for adding security at the very edge, at the sensor level. The other aspect of sensor data security is the physical security of the sensor itself, as outlined in section 2.2.

5. Conclusions

This document has presented the security requirements for mF2C IT-2. They follow up from the security requirements documented in IT-1 (D2.4) [1], considering the lessons learned with the security software developed for IT-1, including the parts that were not integrated into the official release. This deliverable does not replace D2.4; it extends it (e.g. secure by design), or makes the discussion from D2.4 more specific (architecture).

It is clear that we cannot meet all the requirements, as there will quite simply not be enough time. In addition, we would have had to retrain most of the software developers in the project in secure by design methodologies, we would need to take existing code and refactor and harden it – or rewrite it – and then repackage it. What took us most of the past year, leading up to the IT-1 release (and associated but not integrated components) would instead take another three years.

Instead, we need to choose the components to fix, and prioritise the requirements. This is not done here, because this process depends on the deliverables D3.2 and D4.2 which describe the proposed implementation of the security of, respectively, the agent controller and the platform manager. Only after knowing the implementation plans can we assess the work required to secure – or refactor and harden – a component, or whether it is good enough as it is and can be sufficiently isolated in container.

Having said that, there are still a few points that stand out from this analysis of the requirements:

- The analysis of IT-1 (notably sections 2.1, 4.3) showed that the core principle – using an agent-based security model underpinned by a PKI – was a sound choice. However, some software needs to be re-engineered (section 2.1).
- The feedback from the use cases (section 4.1) was that there needs to be a part of the mF2C platform security which can support applications. This need not be implemented from scratch, it should use exactly the same code, but needs to use different keys, because otherwise a compromised application can compromise the platform security.
- Secure by design is quite comprehensive and all developers should be familiar with it. Some components must have a reverse burden of proof, meaning they must explain why they are not secure (if they are not).
- Although not needed in any project deliverable, the question of physical security has popped up again. Having rightly been considered out of scope of IT-1, there is an opportunity for us to investigate physical security, largely independently of software development for IT-2 (with a couple of exceptions, if specific languages, coding environment, or deployment restrictions are needed.) Physical security of edge devices is an issue for UC2 primarily, and to a lesser extent for UC-1.
- Blockchains and machine learning were previously identified as important technologies, without specific application areas. We now have more specific possibilities, using both to implement, innovate, and improve existing operational security characteristics.
- The GDPR has come into force since writing D2.4 [1], and, obviously, cannot be ignored. Implementing encrypted storage is a possibility but is very hard to do correctly and robustly, due to the need for key management – similar to the weaknesses of supporting a fog infrastructure with symmetric keys – both cases would require an architecturally centralised keystore. This does not seem acceptable in a fog where, although the leader may hold keys

(section 4.3.2), it can disappear and be replaced; with symmetric keys a lost keystore breaks the whole infrastructure. Something else needs to implement GDPR.

As mentioned above, the details of the proposed security implementation must now be fleshed out in D3.2 and D4.2, and then the process must start of evaluating what is feasible to achieve within the rest of the project.

References

[1] mF2C Consortium, "D2.4 mF2C Security/Privacy Requirements and Features (IT-1)," April 2017. [Online]. Available: http://www.mf2c-project.eu/wp-content/uploads/2017/05/mF2C-D2.4-Security-Privacy-Requirements-and-Features-IT1.pdf.

[2] mF2C Consortium, "D2.6 mF2C Architecture (IT-1)," June 2017. [Online]. Available: http://www.mf2c-project.eu/wp-content/uploads/2017/06/mF2C-D2.6-mF2C-Architecture-IT-1.pdf.

[3] H. Raj et al., "fTPM: A Software-only Implementation of a TPM Chip".

[4] P. Freemantle, B. Aziz, T. Kirkham, "Enhancing IoT Security and Privacy with Distributed Ledgers - A Position Paper," in *2nd International Conference of Internet of Things, Big Data and Security*, Porto, Portugal, April, 2017.

[5] IBM Security, "Five considerations for blockchain applied to data privacy and GDPR," [Online]. Available: https://www.ibm.com/blogs/blockchain/2018/05/five-considerations-for-blockchain-applied-to-data-privacy-and-gdpr/.

[6] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang and J. Wan, "Smart Contract-Based Access Control for the Internet of Thing".

[7] G. Pinto, J.P. Dias, H.S. Ferreira,, " Blockchain-based PKI for Crowdsourced IoT Sensor Information.".

[8] "Microchip IoT security devices and microcontrollers," [Online]. Available: https://www.microchip.com/design-centers/embedded-security.

[9] "Xilinx Secure IoT," [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp493-iiot-edge-platforms.pdf.

[10] "Security-as-a-Service," [Online]. Available: https://www.rambus.com/security/trusted-services/iot-security-service/.

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

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

[13] "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016," [Online]. Available: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en.

[14] “Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995,” [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A31995L0046.

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

[16] A. Abeshu and N. Chilamkurti, “Deep Learning: The Frontier for Distributed Attack Detection in Fog-to-Things Computing,” vol. 56, no. 2, Feb 2018.

[17] A. Chuvakin and A. Barros, “Preparing Your Security Operations for Orchestration and Automation Tools,” Gartner, 2018.

[18] K. Dempsey et al., “Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations,” *NIST Special Publication,* pp. 800-137, September 2011.

[19] M. Bromiley, “The Show Must Go On! 2017 SANS Incident Response Survey,” SANS Institute, 2017.

[20] T. Julian, “Implementing O&A: Why Intelligence Is the Key to Strategic Orchestration,” 12 March, 2018.