

SCRATCh

SECURE AND AGILE CONNECTED THINGS

Deliverable D1.1b

SCRATCh



Deliverable D1.1b: Requirements Elicitation

Work package: WP1

Affected milestone: MS3

Partners involved: HI Iberia
AnyWi
BEIA
DFKI
Diebold Nixdorf
Irdeto
Nimbeo
Quobis
NXP
Sirris

Date: 27/11/2020

Deliverable version: v1.0

Editor: Raúl Santos de la Cámara, HI Iberia

Author(s): Raúl Santos de la Cámara, HI Iberia

Responsible Contact: Raúl Santos de la Cámara
HI Iberia Ingeniería y Proyectos
c/ Juan Hurtado de Mendoza, 14. 28036 Madrid (Spain)
rsantos@hi-iberia.es
(+34) 699 830 005

Version history

| Date | Version | Author | Comment |
|------------|---------|--------------------------|---|
| 16/04/2020 | 0.1 | Raúl Santos (HIB) | Initial ToC and objectives sent for comments |
| 29/04/2020 | 0.15 | Raúl Santos (HIB) | Refined ToC. |
| 14/05/2020 | 0.2 | Raúl Santos (HIB) | Final ToC, added deliverable production schedule. |
| 23/06/2020 | 0.3 | Franklin Selgert (AnyWi) | Integration of contents for section 2 with abstraction of the SecDevOps phases and links to tools. |
| 02/10/2020 | 0.4 | Raúl Santos (HIB) | Reformulation of section 3, launch of second requirement collection phase. |
| 17/11/2020 | 0.5 | Raúl Santos (HIB) | Requirements M25 inserted in chapter 3, content for section 2.3 on process, added introduction conclusions. |
| 23/11/2020 | 0.9 | Raúl Santos (HIB) | Polishing up of section 2, reformatting of the document. Release candidate proposed for internal review. |
| 27/11/2020 | 1.0 | Raúl Santos (HIB) | Clean up and production of release candidate document. |

Table of Contents

| | |
|---|----|
| 1. Introduction | 5 |
| 2. SCRATCH approach to requirements | 7 |
| 2.1. Evolution of the process within the course of the project..... | 7 |
| 2.2. SCRATCH requirements process as of M25..... | 12 |
| 2.3. Good-practices / Design constraints..... | 14 |
| 2.4. Process guidance document | 20 |
| 3. Requirements M27 | 30 |
| 4. Conclusions and next steps | 51 |
| 4.1. Conclusions..... | 51 |
| 4.2. Next steps | 51 |
| 5. References..... | 52 |

1. Introduction

D1.1b is the culmination of the task T1.1a and T1.1b providing requirements elicitation, analysis and study of its broader management in the scope of a SecDevOps environment. In the previous deliverable T1.1a a first approach was presented, including a first overview of the capture and representation methodologies, some of the possible tools for use in the project and a first list of SCRATCH requirements at that point in the project. This included general requirements as well as some requirements expressing the initial needs of more concrete results of the project such as tools and demonstrators for our project use cases (Smart Retail, Police and Smart Grid).

Since the wrapping up of that document over a year has passed in which the project has moved from an initial conceptualization stage to a development stage in which the major building blocks have been outlined. These include first descriptions of the demonstrator, which outline the problems that we're trying to solve, and the generic demonstrator and its related documentation such as the architecture and QMS document, which present a first approach to the solutions that SCRATCH proposes.

On the highest level, the overall goal of this document is twofold:

1. To present a requirement collection and management strategy that fits within the general methodology proposed by SCRATCH. This includes appropriate links with the toolset and the generic demonstrator. Since the tools are presented as a toolset rather than a monolithic toolchain, it is important that the requirements approach caters to this flexibility, offering options to deal with requirements at different stages of the architecture and different process levels.
2. To present, using that methodology, the requirements for SCRATCH, some of the requirements already defined at the delivery of this document (M27). Since developments in the project are iterative and continue for more than a year and a half after this deliverable is presented, requirements will continue to evolve. So the snapshot presented here is to be understood as a vision in a particular moment of time. Requirements will continue to evolve using the methodology presented in 1) until the end of SCRATCH.

To fulfill these objectives, the deliverable is organized as follows:

- In this section 1, a summary of the document (goals and general approach) is provided.
- In section 2 we present the SCRATCH approach to requirements elicitation, management, storage and connections with the toolset and different parts of the DevOps cycle, addressing the objective 1) above. This is the main methodological part of this document and the ultimate contribution of WP1 on requirements to the SCRATCH workflow.

- In section 3 we present the current (as of M27) requirements for SCRATCH: the methodology itself, the toolkit as a whole and each of the application use cases: Smart Retail, Police and Smart Grid.
- Finally, in section 4 the work performed is summarized and conclusions are presented. This includes links with subsequent activities even after the end of WP1 activities.

2. SCRATCH approach to requirements

2.1. Evolution of the process within the course of the project

In the course of WP1 and the whole of SCRATCH we have iterating several differing views and levels of discussion over how requirements should be handled. These have, in a first approximation, divided basically on two different levels of discussion:

- How the SCRATCH project requirements should be expressed and shared.
- How the requirements of the applications built in accordance to the

The first of these topics (the requirements for SCRATCH) is the main goal of tasks T1.1a and T1.1b in which this deliverable is produced. The collected requirements are presented in the next chapter of this document. But the process of general requirements collection and management for a DevSecOps IoT perspective, which has been part of the discussion for the toolkit in WP2 has informed many of our decisions during the period. We will provide here a short summary of the different strategies followed which mirror well the experience that an SME trying to work out a product in IoT may follow.

The process started with an initial stage in which the most basic methods were put forward: usage of shared table for collecting the requirements and the use of simple methods to format requirements in a straightforward manner.

SCRATCH requirements elicitation

Authors and contact: HI Iberia Ingeniería y Proyectos
 Raúl Santos de la Cámara (rsantos@hi-iberia.es)
 Elena Muelas Cano (emuelas@hi-iberia.es)

Please insert in the table starting in page 2 requirements for the SCRATCH results.

Format: use regular english in the form:

[Req_seqNo_PART-ACR] asset [MUST; SHOULD; COULD; WON'T] requirement
 i.e.: [Req_001_HIB] The Rule Engine MUST comply with the JSR94 standard

Use [MoSCoW prioritization verbs](#) and the following informal guidelines for your requirements!

- a) **Correct**: not containing false statements
- b) **Unambiguous**: not subject to multiple interpretations
- c) **Complete**: as self contained as possible
- d) **Consistent**: among all of the requirements in the system
- e) **Ranked for importance**: distinguish essential from accessory
- f) **Verifiable**: can be tested in the system
- g) **Modifiable**: can evolve over time
- h) **Traceable**: evolution is stored and can be analysed

This is just a summary of the [IEEE 830-1998](#) standard, feel free to refer to it for more details on what the quality attributes mean in detail.

For any issue or doubt please refer to task leaders Raúl or Elena whose email can be found above.

Requirements list

| Req ID | Requirement Text |
|---------------|---|
| [Req_001_HIB] | The Rule Engine MUST comply with the JSR94 standard (example) |
| | |
| | |

FIGURE 1 REQUIREMENTS COLLECTION IN SCRATCH: TEXT-BASED APPROACH

As readable in the Figure, it was proposed that requirements would be expressed in plain language and according to the MoSCoW prioritization terminology and IEEE 830-1998 [1] directions. This approach was discussed internally in the project prior to the kickoff (Spring 2019) and was deemed easy to understand by partners and useful to a certain degree. It represents very much the way in which requirements are elicited and managed for an SME that deals with simple products, most often incorporating little to no security. As such, we could define it as the baseline approach to requirements management.

After the kickoff and upon discussions with the experts in the project that had more knowledge of the different available approaches, it was decided to try and experiment with requirement collection using a dedicated markup language, concretely sphinx-needs [2]. This approach, which was discussed in some depth in the predecessor to this document, D1.1a, offers significant functional benefits:

- More precise syntax, including elements for signaling dependencies, levels of requirements description and others.

- Better versioning: since requirements are expressed in a mark-up language close to rST or Markdown, they can be readily versioned in a code versioning system.
- Better links with code: since they can be expressed at the same level as the code (text files in a versioning system) it is easier to link requirements with code assets. This makes it also much easier to integrate them in tools such as unit testing systems as tests can readily validate outputs with the expected value of the requirement.

These advantages were deemed useful by the consortium and a sphinx-needs versioning and automatic processing system was put in place. The consortium was then asked to elicit the initial requirements and around 40 requirements were collected and reported in D1.1a. The tools provided good representation of the requirements as depicted in Figure 2.



FIGURE 2 SAMPLE REQUIREMENT IN SPHINX-NEEDS

After the deliverable D1.1a was compiled, the approach was reevaluated. It was found out that, while potentially useful and offering benefits to deal with the technical aspects of the project, this approach presented some issues:

- The learning curve of the syntax was very steep. Although a guide was produced so partners could learn the basics of the language and some helping assets such as templates were generated, partners struggled to codify requirements. Sphinx-needs is a markup oriented to developers while producers of requirements are typically more business-oriented users that maybe weren't very efficient in aspects such as keeping text indentation or removing trailing spaces.
- The approach wasn't equally suitable for all kinds of requirements. While for some technical requirements the process could yield significant benefits (e.g., a technical specification of a IoT device's performance such as throughput) it offered fringe benefits to the majority of requirements. For example, Use Case requirements are very much user-story oriented and so they couldn't be

automatically tested. Then the benefits achieved do not justify the extra effort put into generating the sphinx-needs markup.

- The available tools are very basic. While setting up an environment that manages the sphinx-needs markup in a versioning system and generates some output on its compilation was quite straightforward, in the end all editing of the requirements was done in text editors that provide little help to the end-user. This was exacerbated by the fact that checking of syntax was only possible upon committing changes to the repository and checking the output (if syntax was at least correct for compilation) or error traces in a Drone output. This had also the inconvenience that if one user produced bad input, all of the process could be disrupted.

With this experience at hand, it was decided that using sphinx-needs as the mainline tool for requirements management was more suited for very focused development operations and not for the main project.

At the same time, and evolving also from a different part of the work presented in D1.1a, a tool for compiling and managing the general industry requirements (e.g., from ENISA, OWASP and others) was starting to emerge in the project. It was the Knowledge Base kept by AnyWi in the www.trusttab.com website:

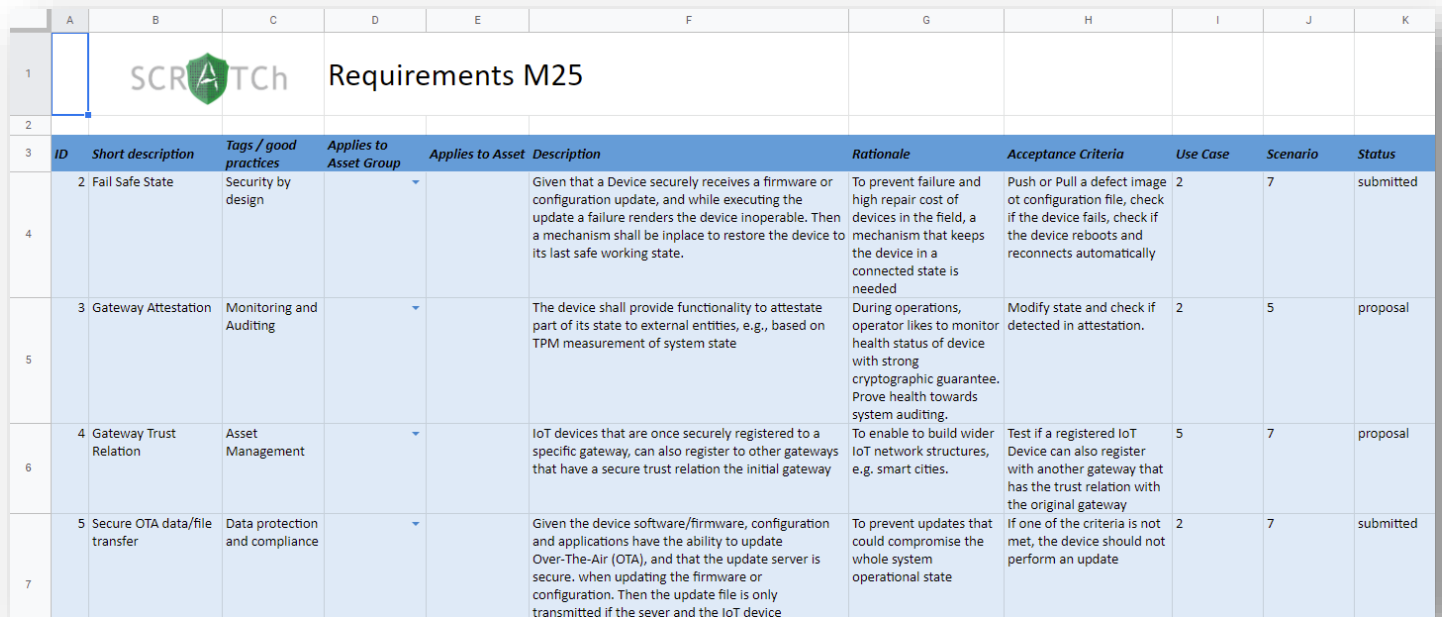
| # | THREAT GROUP | THREAT | Description | ASSETS AFFECTED | Remarks |
|---|------------------|-------------------------------|--|-----------------------------|----------------------|
| 1 | OUTAGES | Network Outage | Interruption or failure in the network supply, either intentional or accidental. Depending on the network segment affected, and on the time required to recover, the importance of this threat ranges from high to critical. | COMMUNICATIONS | VIEW |
| 2 | PHYSICAL ATTACKS | Device destruction (sabotage) | Incidents such devices theft, bomb attacks, vandalism or sabotage could damage devices | INFRASTRUCTURE | VIEW |
| 3 | PHYSICAL ATTACKS | Device destruction (sabotage) | Incidents such devices theft, bomb attacks, vandalism or sabotage could damage devices | PLATFORM & BACKEND | VIEW |
| 4 | PHYSICAL ATTACKS | Device destruction (sabotage) | Incidents such devices theft, bomb attacks, vandalism or sabotage could damage devices | OTHER IOT ECOSYSTEM DEVICES | VIEW |
| 5 | PHYSICAL | Device destruction | Incidents such devices theft, bomb attacks, vandalism or sabotage | IOT DEVICES | VIEW |

FIGURE 3 TRUSTTAB WEBSITE SHOWING ENISA THREATS IN THE KNOWLEDGE BASE

This tool appeared as an excellent repository for these generic high-level assets and SotA and soon it was apparent that it could also host requirements in a more human-

readable form than sphinx-needs. This was the origin of the method for collecting requirements that we have used for this document, the final result of which is discussed in the following Chapter 3.

But usage of the TrustTab website could also be quite technical for all of the intended end-users of a requirements tool (which includes developers but also business users), so a transitional approach was used.



| ID | Short description | Tags / good practices | Applies to Asset Group | Applies to Asset | Description | Rationale | Acceptance Criteria | Use Case | Scenario | Status |
|----|-------------------------------|--------------------------------|------------------------|------------------|---|---|--|----------|----------|-----------|
| 2 | Fall Safe State | Security by design | | | Given that a Device securely receives a firmware or configuration update, and while executing the update a failure renders the device inoperable. Then a mechanism shall be in place to restore the device to its last safe working state. | To prevent failure and high repair cost of devices in the field, a mechanism that keeps the device in a connected state is needed | Push or Pull a defect image ot configuration file, check if the device fails, check if the device reboots and reconnects automatically | 2 | 7 | submitted |
| 3 | Gateway Attestation | Monitoring and Auditing | | | The device shall provide functionality to attestate part of its state to external entities, e.g., based on TPM measurement of system state | During operations, operator likes to monitor health status of device with strong cryptographic guarantee. Prove health towards system auditing. | Modify state and check if detected in attestation. | 2 | 5 | proposal |
| 4 | Gateway Trust Relation | Asset Management | | | IoT devices that are once securely registered to a specific gateway, can also register to other gateways that have a secure trust relation the initial gateway | To enable to build wider IoT network structures, e.g. smart cities. | Test if a registered IoT Device can also register with another gateway that has the trust relation with the original gateway | 5 | 7 | proposal |
| 5 | Secure OTA data/file transfer | Data protection and compliance | | | Given the device software/firmware, configuration and applications have the ability to update Over-The-Air (OTA), and that the update server is secure, when updating the firmware or configuration. Then the update file is only transmitted if the sever and the IoT device | To prevent updates that could compromise the whole system operational state | If one of the criteria is not met, the device should not perform an update | 2 | 7 | submitted |

FIGURE 4 REQUIREMENTS ELICITATION FOR M25 USING SPREADSHEET

The general taxonomy of elements needed for a requirement was used as the basis for a simple requirement collection spreadsheet which was collectively filled in by partners in the project. This was done using a collaborative spreadsheet so all partners could operate in parallel.

After the spreadsheet was complete, a simple import into the database of TrustTab was done so that the requirements could be accessed online.

In the end this approach yielded excellent results both in terms of number of requirements and in general speed of collection by partners. In comparison with the cumbersome process used in D1.1a, for this final iteration we have doubled the number of requirements in a fashion which is easy to understand but can still lead to high-quality, reusable requirements: the elements in TrustTab could be accessed via REST calls by testing programs in order to validate the requirements same as was proposed with sphinx-needs.

It may seem like the approach has come full-circle from the first offering of collaborative documents for requirements, but this final approach is much more integrated with project assets. TrustTab and other tools are currently being proposed as part of the

toolset in WP2 as the core result in SCRATCH. Other open source tools such as CAIRIS¹ and Ephememis², which offer sophisticated requirement management as well as connections with other areas of interest such as risk management are being considered for inclusion in the next iterations of the SCRATCH toolkit.

2.2. SCRATCH requirements process as of M25

In this section, and having taken into consideration the evolution presented in the previous section 2.1, we present the current process of requirements engineering followed in SCRATCH for the production of requirements.

This links with the work undertaken in task T1.2 and work package WP2 regarding the production of a toolkit for developers to produce SCRATCH-based applications. In that production process requirements engineering is often the first activity and so it has to rely on a methodological framework that is presented here in the form of the different processes to follow.

According to Sommerville (Sommerville, 2016) the requirements process contains three phases: elicitation and analysis, specification and validation. A logical fourth would be requirement management. The all-day practice is more complicated and fuzzier, Agile practices, rapid prototyping, novel products, new markets, all contribute to a requirement process not so structured. The experience in the development of the SCRATCH demonstrators did underpin this thought. Analyzing the relation between security and requirements was a next step after initial requirements where gathered. Some thoughts on this analysis is reflected in the whitepaper (franklin2020). A conclusion of this whitepaper: inject (security)design constraints at the beginning of a development cycle, e.g. use an abstract of the ENISA good practices.

How does this fit in the requirements process is presented in Figure 5

¹ <https://cairis.org/> - CAIRIS requirements management tool.

² <https://github.com/shuart/ephememis> - Ephememis requirements management tool.

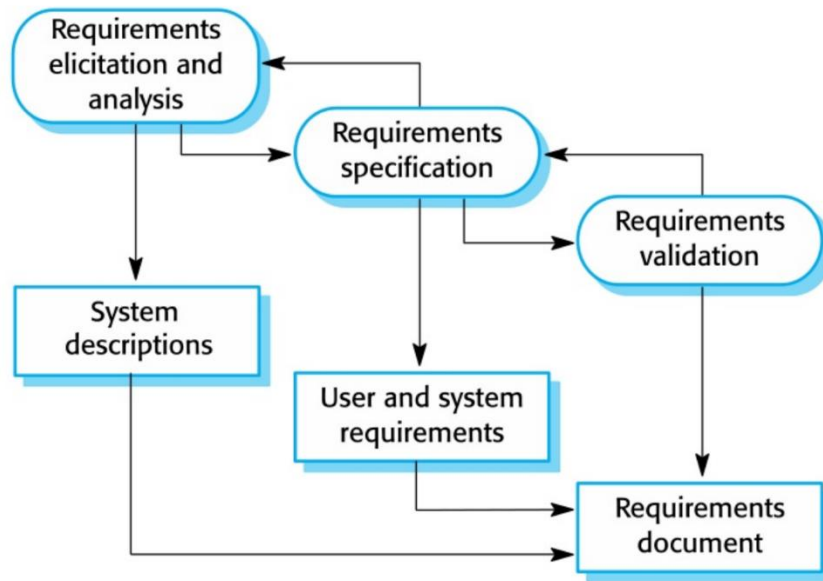


FIGURE 5 REQUIREMENTS ENGINEERING (SOMMERVILLE 2016)

Injecting at start mostly impacts system requirements, analyzing security features it is safe to assume that they mainly impact system description and architecture and are corrected or changed based on user or stakeholder needs. Typically, an agile process feedback from early prototypes may require adaptation of security measurements from a usability perspective or technical limitations. It is these iterations that lead to a viable product, that is then as a last step validated against requirements and sector specific regulations. If due process is followed the system should pass the sector specific regulations as they were input in the early stage development, and deviation should be documented and mitigated.

The SCRATCH approach to the requirements process based on analysis and demonstrator experience is:

1. **Create a minimum generic good practice set to start with for any development.**
The used tools here would be the Knowledge base category system requirements (as depicted in section 2.1 in the Figure 3). This corresponds to the *generic requirements* as presented in D1.1a.
2. **Add a minimum set of process related good practices** to guide the DevOps process.
3. **Combine both sets** in a process guidance document describing the DevOps cycle, with Tooling examples including the SCRATCH specific tools.
4. **For each demonstrator an evaluation document** describing the use of requirements, best practices & guidelines in their development. (separate documents)

We continue to examine the different steps proposed in this process.

2.3. Good-practices / Design constraints

1: IT Architecture technical requirements (source ENISA)

| Title | Description | Threats |
|---------------|--|---|
| Authorization | Device firmware should be designed to isolate privileged code, processes and data from portions of the firmware that do not need access to them, and device hardware should provide isolation concepts to prevent unprivileged from accessing security sensitive code. in order to minimise the potential for compromised code to access those code and/or data. | Failures / Malfunctions Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |
| Cryptography | Ensure a proper and effective use of cryptography to protect the confidentiality, authenticity and/or integrity of data and information (including control messages), in transit and in rest. Ensure the proper selection of standard and strong encryption algorithms and strong keys, and disable insecure protocols. Verify the robustness of the implementation. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |
| Cryptography | Cryptographic keys must be securely managed. Encryption is only as robust as the ability for any encryption-based system to keep the encryption key hidden. Cryptographic key management includes key generation, distribution, storage, and maintenance. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |
| Cryptography | Support scalable key management schemes. It has to be considered that tiny sensor nodes cannot provide all security features because they have lots of system limitations. Thus, the sensed data carried over infrastructure networks may not have strong encryption or security protection. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking Failures / Malfunctions |
| Cryptography | Build devices to be compatible with lightweight encryption and security techniques (including entities secure identification, secure configuration, etc.) that can, on the one hand, be usable on resource-constrained devices, and, on the other hand, be scalable so to minimise the management effort and maximise their usability. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |

| Title | Description | Threats |
|-----------------------------------|---|---|
| Secure and trusted communications | Disable specific ports and/or network connections for selective connectivity. If necessary, provide users with guidelines to perform this process in the final implementation. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |
| Secure and trusted communications | Ensure that communication security is provided using state-of-the-art, standardised security protocols, such as TLS for encryption. | Eavesdropping / Interception / Hijacking Damage / Loss (IT Assets) |
| Secure and trusted communications | Guarantee the different security aspects - confidentiality (privacy), integrity, availability and authenticity- of the information in transit on the networks or stored in the IoT application or in the Cloud, using data encryption methods to minimise network threats such as replay, interception, packet sniffing, wiretapping, or eavesdropping. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking Failures / Malfunctions |
| Secure and trusted communications | Guarantee data authenticity to enable trustable exchanges (from data emission to data reception - both ways). Data is often stored, cached, and processed by several nodes; not just sent from point A to point B. For these reasons, data should always be signed whenever and wherever the data is captured and stored. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |
| Secure and trusted communications | Rate limiting – controlling the traffic sent or received by a network to reduce the risk of automated attacks. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |
| Secure and trusted communications | Make intentional connections. Prevent unauthorised connections to it or other devices the product is connected to, at all levels of the protocols. IoT devices must provide notice and/or request a user confirmation when initially pairing, onboarding, and/or connecting with other devices, platforms or services. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |

| Title | Description | Threats |
|--|---|--|
| Secure Interfaces and network services | Implement a DDoS-resistant and Load-Balancing infrastructure to protect the services against DDoS attacks which can affect the device itself or other devices and/or users on the local network or other networks. | Nefarious Activity / Abuse |
| Secure Interfaces and network services | Ensure web interfaces fully encrypt the user session, from the device to the backend services, and that they are not susceptible to XSS, CSRF, SQL injection, etc. | Nefarious Activity / Abuse |
| Secure Interfaces and network services | Ensure only necessary ports are exposed and available. | Eavesdropping / Interception / Hijacking Failures / Malfunctions |
| Secure Software / Firmware updates | Offer an automatic firmware update mechanism. Devices should be configured to check for the existence of firmware updates at frequent intervals. Automatic firmware updates should be enabled by default. A device may offer an option to disable automatic firmware updates and require authentication for it. | Outages Failures / Malfunctions |
| Secure Software / Firmware updates | Backward compatibility of firmware updates. Automatic firmware updates should not change network protocol interfaces in any way that is incompatible with previous versions. Updates and patches should not modify user-configured preferences, security, and/or privacy settings without user notification. Users should have the ability to approve, authorise or reject updates. | Outages Failures / Malfunctions |
| Trust and Integrity Management | Sign code cryptographically to ensure it has not been tampered with after being signed as safe for the device, and implement run-time protection and secure execution monitoring to be sure malicious attacks do not overwrite code after it is loaded. Only run signed code and never unsigned code. | Nefarious Activity / Abuse Eavesdropping / Interception / Hijacking |

| Title | Description | Threats |
|---------------------------------------|--|---|
| | <p>Measuring the boot-process enables the detection of manipulation of the host OS and software, so that malicious changes in the behaviour of the devices can be detected. It enables boot-time detection of rootkits, viruses and worms.</p> | |
| <p>Trust and Integrity Management</p> | <p>The boot process initialises the main hardware components, and starts the operating system. Trust must be established in the boot environment before any trust in any other software or executable program can be claimed, so the booted environment must be verified and determined to be in an uncompromised state.</p> | <p>Failures / Malfunctions Nefarious Activity / Abuse Outages</p> |

2: Process related requirements

| Title | Description |
|---|--|
| Asset Management | Establish and maintain asset management procedures and configuration controls for key network and information systems, to identify and authenticate of the assets involved in the IoT Service (i.e. Gateways, Endpoint devices, home network, roaming networks, service platforms, etc.). |
| Privacy by design | Privacy must be a guiding principle when designing and developing systems, in order to make privacy an integral part of the system. |
| Privacy by design | <p>Perform privacy impact assessments before any new applications are launched, using a top-down decomposition method that requires first answering three fundamental questions:</p> <ul style="list-style-type: none"> - Where is the targeted application deployed (Legal constraints and cultural significance) - For what purpose (Scope) - For which scenarios (Business requirements) |
| Risks and Threats Identification and Assessment | Identify the intended use and environment of a given IoT device. This will help developers and manufacturers determine the most suitable technical features for the IoT device's operation, and the security measures required. This will also help to effectively handle bugs or enhancement requests |

| | |
|--|--|
| <p>Risks and Threats Identification and Assessment</p> | <p>Identify significant risks using a defence-in-depth approach. Conduct end-to-end risk assessments that account for both internal and third-party vendor risks, where possible. Developers and manufacturers should include vendors and suppliers in the risk assessment process, which will create transparency and enable them to gain awareness of potential third-party vulnerabilities and promote trust and transparency. Security should be readdressed on an ongoing basis as the component in the supply chain is replaced, removed or upgraded.</p> <p>Risk Assessment procedure should be initiated using a top-down decomposition method that requires first answering three fundamental questions:</p> <ul style="list-style-type: none"> - Where is the targeted application deployed (Legal constraints and cultural significance) - For what purpose (Scope) |
| <p>Security by design</p> | <p>Design architecture by compartments to encapsulate elements in case of attacks</p> |
| <p>Security by design</p> | <p>Ensure the ability to integrate different security policies and techniques, so as to ensure a consistent security control over the variety of devices and user networks in IoT</p> |
| <p>Security by design</p> | <p>Security must consider the risk to human safety</p> |
| <p>Security by design</p> | <p>For IoT hardware manufacturers and IoT software developers it is necessary to implement test plans to verify whether the product performs as it is expected. Penetration tests help to identify malformed input handling, authentication bypass attempts and overall security posture.</p> |
| <p>Security by design</p> | <p>Design for power conservation should not compromise security</p> |
| <p>Security by design</p> | <p>Consider the security of the whole IoT system in a consistent and holistic approach along its whole lifecycle across all levels of device/application design and development, integrating security throughout the development, manufacturing, and deployment</p> |
| <p>Security by design</p> | <p>For IoT software developers it is important to conduct code review during implementation as it helps to reduce bugs in a final version of a product.</p> |

2.4. Process guidance document

In this section we describe how the guiding principles above match with the different tools that we propose in the SCRATCH toolset as it can be identified in D1.2. This is also connected to the processes investigated and put forward in the work of work package WP2, in this document we only want to present a short summary of how the requirements management process in SCRATCH applications is informed by the work done in T1.1 and the rest of WP1.

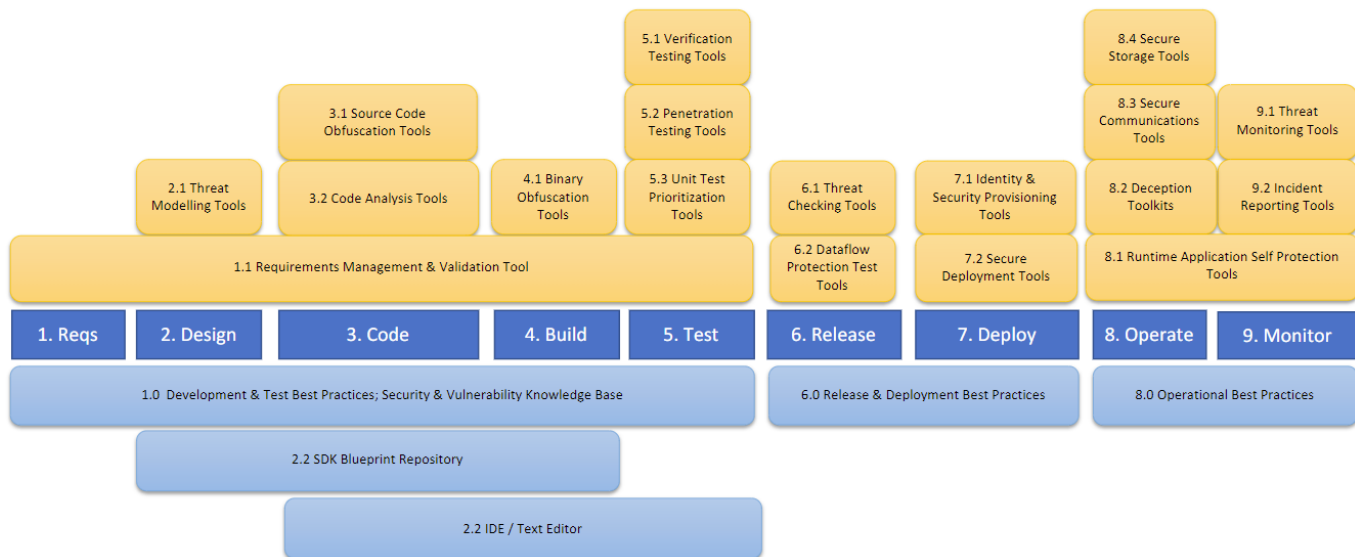


FIGURE 6 SCRATCH COMPLETE TOOLSET (M27)

In the Figure 6 we see the different tools as identified in the task T1.2. We now will abstract the lifecycle of a SecDevOps application and identify for each one of the subprocesses that is contained in that lifecycle the tools from the Figure that are applicable.

Process Description

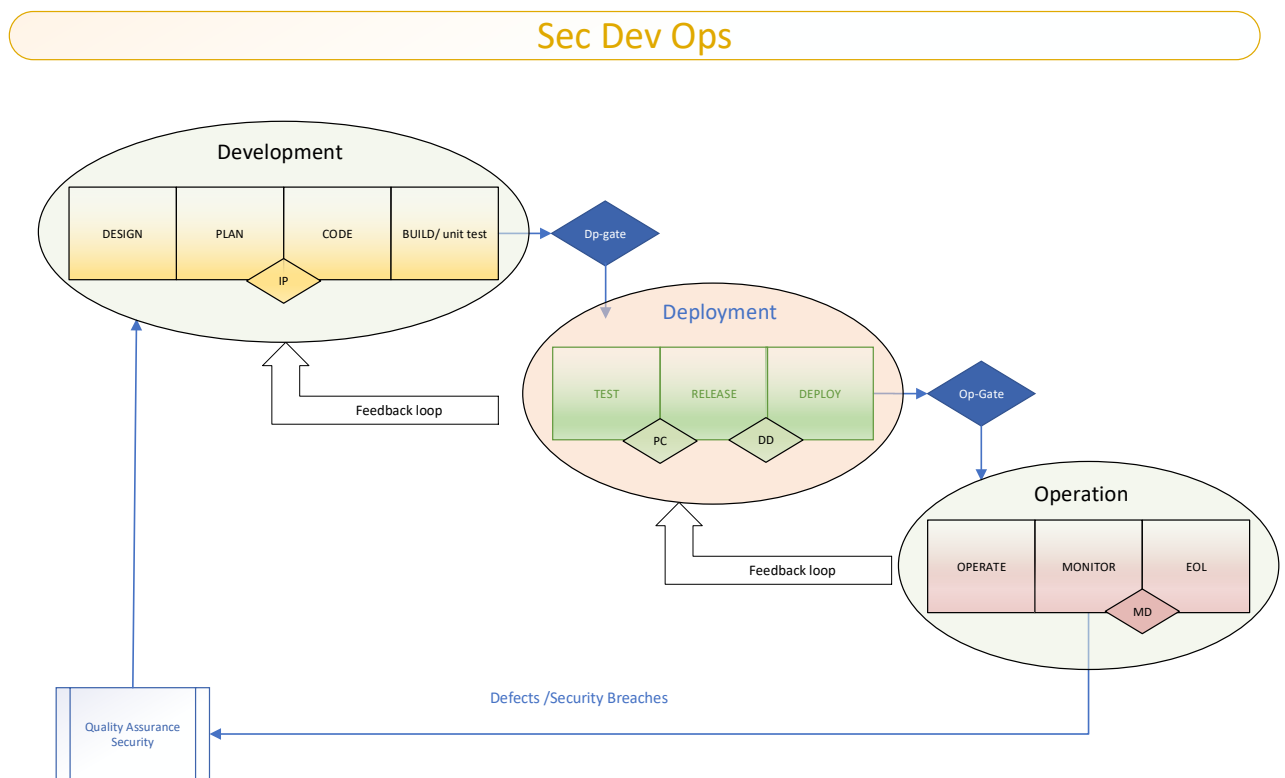


FIGURE 7: SEC DEV OPS PROCESS

In Figure 7 we can see our proposed abstraction of the complete lifecycle of the SecDevOps application as a process which spans the entire lifecycle of a product and is divided into 3 sub-processes:

1. Development
2. Deployment
3. Operation

Between the different sub-processes described in Figure 7 there are multiple decision moments (gates) that interconnect each of the sub-processes. By providing automation or decision support tools in these gates the process can evolve in a continuous integration, delivery, deployment or DevOps process hereby implementing SCRATCH vision of integrated security.

We will now delve into a summary of each one of the three sub-processes (development, deployment and operation) and analyse the underlying tasks that comprise them and match them with existing SCRATCH tools in the toolset presented in D1.2 or rather identify gaps that we will need to fulfil in future work. Each of the sub processes is itself

divided into different phases that will be also summarized and explained in the following subsections.

Development

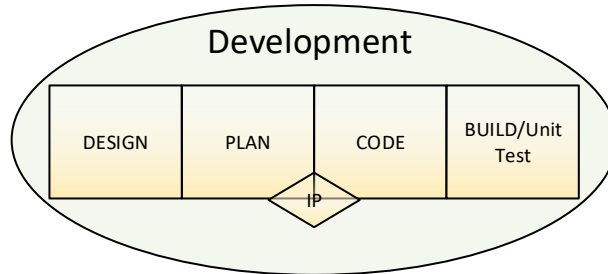


FIGURE 8: SCRATCH METHODOLOGY DEVELOPMENT PHASE

The ultimate goal of the **development process** is to compile a list of needs/requirements by an external agent and translate a set of needs/ requirements into a prototype. For this, phases of planning and coding are also necessary.

Design Phase

Purpose:

1. Capture user and business requirements
2. Define system requirements
3. Draft or choose a starting system architecture
4. Evaluate possible solutions

Main activities (details in procedure requirements gathering)

Document the business requirements, this include targeted security level, regulatory environment, intended distribution area. The guideline for business requirements capture is set in the proposed template [BR-TEMP.DOC](#) (see WP2 documentation for complete reference of these documents). Sector requirements are based on the intended use and regulatory environment and are recorded in [IU-TEMP.DOC](#).

Document user requirements, this includes security measurements/ requirements. The guideline for user requirements capture is set in template [T-UR-001.DOC](#). The expected outcome should be detailed enough to abstract features for agile development.

Based on the business requirements system requirements defining performance assumptions and hardware boundaries are drafted and captured according template [T-SR-001.DOC](#)

A system architecture is chosen, adapted or defined. The to be developed product or software is expected to function in a certain environment/architecture, for the purpose of simplicity only essential parts of the architecture are described or chosen. Essential parts are e.g. Interface, communication, security. Template to be used [T-SA-001.DOC](#).

After a basic layout of the system is known a first security scan can be performed using the STRIDE Model ([T-STRIDE-001.DOC](#)) early awareness of security issues will increase

overall security of the end product at minimal additional cost (The 2019 State of DevOps Report, presented by Puppet, CircleCI and Splunk).

Based on the available information possible solutions are discussed, and a first selection is made, this including e.g. tools, components, development environment. These decisions are captured in a design output document using template [T-DO-001.DOC](#).

Documentation criteria are set in [DOC-CRIT.XYZ](#) and can depend on the specifics of the product under development and the applicable certification demands for the targeted industry sector.

In the following tables we define proposals (as of the closing of this document) of tools and templates for documents to follow the process in a SCRATCH fashion, including the numbering references (e.g., 1.1 for requirements management and validation tool in the next table) to tools in the SCRATCH toolkit as defined in D1.2. This will be repeated for each phase in every process in subsequent sections. However, two caveats apply: (1) this is incomplete work valid only by the writing of this D1.1b and (2) this is WP2 work and is only stated here for informative purposes.

TABLE 1 SCRATCH TOOLSET SUGGESTIONS FOR THE DESIGN PHASE

| name | Description | Ref knowledge base |
|---|--|------------------------|
| Knowledge base | Source of information on available standards and certification requirements and best practices | www.trusttab.com |
| Requirements management & validation tool | To manage, trace and track and validate requirements. Commercial and open source tools available. | 1.1 |
| Threat Modeling | Software to perform threat modeling, proposed to keep a simple approach in design Phase and use the STRIDE template. | STRIDE template NXP |

TABLE 2 DOCUMENTS USED IN DESIGN PHASE

| Document/ template | description | Mandatory |
|-------------------------------------|--|-----------|
| T-BR-XYZ.DOC | Business requirements | Yes |
| IU-XYZ.DOC | Intended Use | Yes |
| T-UR-XYZ.DOC | User requirements | Yes |
| T-SR-XYZ.DOC | System requirements | Yes |
| T-SA-XYZ.DOC | System architecture | No |
| T-DO-XYZ.DOC | Design outputs | Yes |
| T-DEV_QMS_TOOLS.DOC | Tools used listed | No |
| QMS-DOC-CRIT.XYZ | Criteria for documentation | No |
| CMDL | Configuration Management Document List | Yes |
| GRCL | Gate Review Check List | No |

Plan Phase (based on Agile development)

Goal: create and agree upon a feasible planning for the creation of the product/software.

Purpose:

1. Agree upon a planning and the deliverables for a first increment

Main activities

The Design Phase should deliver a general understanding of to be developed product including some rough high-level requirements, some functional component descriptions, security issues and development direction (used tools and software environment). A product owner and development teams should now have enough information to define and describe stories and features for the first increment. Any missing info or requirements can be further detailed in a sprint. There are no specific tools for this phase, but several tools are available that could simplify the work such as Trello.

Code Phase

Goal: agile development of software modules to be used or integrated with chosen hardware

Purpose:

1. Create software modules that fulfill the set stories and features

Main activities

Converts stories into code, captures tests and requirements in code or in separate documents. Code is to be made compliant to the coding standards set in the development environment described in document [CODE-HBK-XYZ.DOC](#). This activity is guided by the procedure Software development [SW-XYZ.DOC](#).

Multiple sprints construct the prototype (software + documentation). Software is continuously compiled in the Build phase and compiler errors are corrected before a next build is scheduled.

Finished features are documented in the [FINISHED FEATURE LIST FFL-XYZ.DOC](#), in this document:

Realized system requirements, solved defects and security breaches are documented.

A trace back to user needs is documented

Requirements and associated testcases are documented conform the company standard described in [CODE-HBK-XYZ.DOC AND SW-XYZ.DOC](#) AND deviations are documented in [DO-XYZ.DOC](#).

TABLE 3 SCRATCH TOOLSET SUGGESTIONS FOR THE CODE PHASE

| name | Description | Ref knowledge base |
|---|---|--------------------|
| Knowledge base | Source of information on available standards and certification requirements and best practices | www.trusttab.com |
| Requirements management & validation tool | To manage, trace and track and validate requirements. Commercial and open source tools available. | <u>1.1</u> |
| Code Analysis Tools | Different tools may apply, depending on the used coding standard and development environment | <u>3.2</u> |
| Code editor | Different tools may apply, depending on the used coding standard, editor aids the developer. | |
| Source Code Obfuscation tool | Provides protection against reverse engineering, tampering, and IP theft, use vary per business. | |

Build Phase

Goal: compile created software modules, check for compiler errors, and know security issues in used libraries.

Purpose:

1. Create executable safe software modules that fulfill the set stories and features

Main activities

This process is largely automated, only if a build failed a message is send out to the configuration manager then first priority will be to repair the build. The build process is described in procedure [BUILD-XYZ.DOC](#) Included in the build phase are code analysis, code coverage and automated test cases. The results are captured in a build report, the build report is checked by the configuration manager and identified issues will be put on the backlog, Issues can vary from lack of code coverage to potential security issues in the code or libraries. The build process is executed on the main branch, Builds of sub branches not merged with the main branch are out of scope.

TABLE 4 SCRATCH TOOLSET SUGGESTIONS FOR THE BUILD PHASE

| name | Description | Ref knowledge base |
|-------------------------|--|--------------------|
| Code Analysis Tools | Different tools may apply, depending on the used coding standard and development environment | <u>3.2</u> |
| Code editor | Different tools may apply, depending on the used coding standard, editor aids the developer. | |
| binary Obfuscation tool | Provides protection against reverse engineering, tampering, and IP theft, use vary per business. | |
| Threat checking tool | Tool that checks the code on known vulnerabilities in used libraries | |

Dp-Gate (decision deployment gate)

The goal of this gate is to decide on the maturity of the developed software/product in order to continue to the next cycle deployment. the documents or proof needed to pass this gate are listed in the Gate Review Check List (GRCL)

Deployment

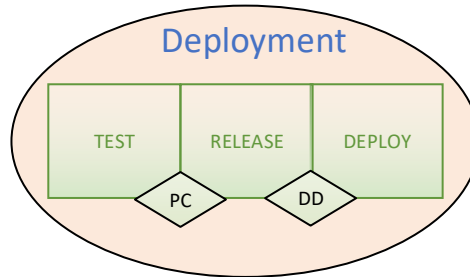


FIGURE 9: SCRATCH METHODOLOGY DEPLOYMENT PHASE

Once the development process is finished, the next stage in the cycle would be to run the **deployment process**, in which the assets produced in development are validated, packaged and installed for their operation.

This process depends on a go of the release a decision made by management. This decision depends on a number of factors which differ per industry sector, in the **CERTIFICATION PLAN** these factors are listed.

Gates:

- The Product committed (PC) gate means that the product passed final tests, if a test failed continuation is only possible if accepted by the QCB
- The Decision to Deploy (DD) is a management decision

Test Phase

Goal: verify the reliability of the product/software

Purpose:

Increase confidence that a product/software can be released to customers

Main activities

In this Phase the software is verified and validated, verification is automated as much as possible, validation depends on type of product /software. Both steps are described in a verification and validation plan. The Content of these plans depend on type of product and applicable certification scheme. E.g. in research projects there is no obligation to provide plans for verification and validation yet.

Unit testing, integration testing and if needed validation of e.g. system requirements (load testing) on a staging environment is done in this phase

TABLE 5 SCRATCH TOOLSET SUGGESTIONS FOR THE TEST PHASE

| name | Description | Ref knowledge base |
|---|--|--------------------|
| Verification testing guidance | Guidance for verification of product/software, e.g. testing protocol for compliance to certain security standards. | <u>5.1</u> |
| Penetration testing tools | A toolset used by expert to test a system on security issues, known and unknown. sometimes mandatory for certification. | <u>5.2</u> |
| Requirements management & validation tool | To manage, trace and track and validate requirements. Commercial and open source tools available. In this phase all requirements should be checked traced accepted or passed | <u>1.1</u> |

Release Phase

Goal: Release the product for deployment.

Purpose:

Check if all conditions for release are met and decide on a deployment plan

Main activities

After the test phase and passing of the PC gate there is enough confidence in the product to release it to the market. In this phase the management team will decide on release based on the results from previous phases. The phase will deliver a deployment plan describing how the product/ software will be released. E.g. deployment can be phased to a certain area or customer base. Also, some last checks on the final build is performed.

The release phase delivers all the information for the decision to deploy gate.

TABLE 6 SCRATCH TOOLSET SUGGESTIONS FOR THE RELEASE PHASE

| name | Description | Ref knowledge base |
|--------------------------------|---|--------------------|
| Dataflow protection test tools | Automatically analyzing traffic data (e.g. in pcap format) w.r.t. dataflows that violate privacy and security requirements pcap input with the help of a keyword search | <u>5.1</u> |
| Threat checking tool | Repeat threat check for updates in the known vulnerabilities databases | |

Deploy

Goal: install or deliver product/software to end customers

Purpose:

Deliver product/ software to end customers according deployment plan.

Main activities

Deployment means release to customers, based on the deployment plan this can be a phased rollout to a selected number of customers, or a deploy to all installed base.

During deployment more effort of support staff is expected to monitor the deployment and repair small errors fast if needed.

Deployment tools are more specific to the implemented architecture, software and hardware. Listed below are some general categories of tools that aid the deployment.

TABLE 7 SCRATCH TOOLSET SUGGESTIONS FOR THE DEPLOYMENT PHASE

| name | Description | Ref knowledge base |
|--|---|--------------------|
| Identity & Security Provisioning Tools | Generates and assigns identities and associated security assets to IoT devices in a secure manner. This may include assets such as unique identifiers, X.509 certificates, symmetric keys, passwords, and associated metadata that is required for securely managing a device throughout its entire lifecycle | <u>7.1</u> |
| Secure Deployment Tools | Facilitate secure firmware updates for networks of IoT devices. Users are able to deliver new firmware to devices for which they are authorised, without risk of “bricking” i.e. getting the device stuck with incomplete code, and without risk of outsider tampering. | <u>7.2</u> |

Operation

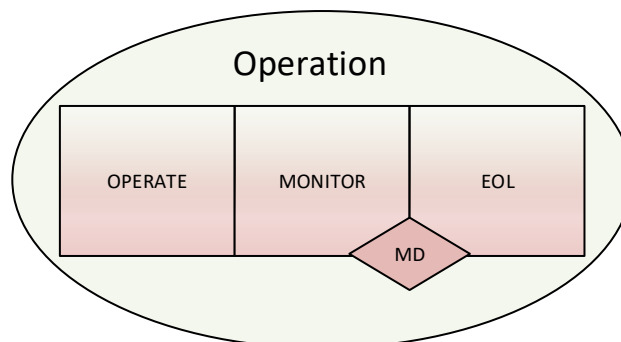


FIGURE 10: SCRATCH METHODOLOGY OPERATION PHASE

After decision to deploy the product/software enters the **operational process**. During the lifecycle an obligation exists to maintain the product/software. Maintenance is agreed upon in a separate agreement (out of scope for this documents). Obligations may vary from complete system to components or only software updates and security patches.

Gates:

- The MD Gate is a management decision to end the lifecycle of a product/software. After this gate a phaseout process starts and after a predefined time all support for the product halts.

Operation Phase

Goal: maintain the products within set specification

Purpose:

Keep a product/software operational and functioning within contractual agreements

Main activities

1. Keep track of security patches published for used software libraries.
2. Analyze and react on messages from the monitoring tools
3. Install security patches if needed.

Tools used in this Phase are listed in the Tools list

Monitor Phase

Goal: monitor installed base on reliability aspects.

Purpose:

Keep a product/software reliable and operational within contractual agreements

Main activities

1. Analyze messages from the monitoring tools
2. Fine tune rules used by monitoring tools

Tools used in this Phase are listed in the Tools list

Document references.

The following are the documents that we defined to collect the process information described in the different phases mentioned in the previous subsections.

TABLE 8 DOCUMENTS REFERENCE FOR MONITOR PHASE

| Document/ template | description | Mandatory |
|---------------------|--|-----------|
| T-BR-XYZ.DOC | Business requirements | Yes |
| CERT-XYZ.DOC | Certification Requirements | No |
| T-UR-XYZ.DOC | User requirements | Yes |
| T-SR-XYZ.DOC | System requirements | Yes |
| T-SA-XYZ.DOC | System architecture | No |
| T-DO-XYZ.DOC | Design outputs | Yes |
| T-DEV_QMS_TOOLS.DOC | Tools used listed | Yes |
| QMS-DOC-CRIT.XYZ | Criteria for documentation | No |
| CMDL | Configuration Management Document List | Yes |
| GRCL | Gate Review Check List | Yes |

3. Requirements M27

What follows in this section is the snapshot of SCRATCH requirements on project milestone 3 in project month M25. As the project continually iterates its requirements in a DevOps fashion, the releases in the deliverables so far (D1.1a released in M16 and this D1.1b released in M27) are better thought as snapshots of the status of the requirements rather than definitive compilations.

However, given that work package WP1 is officially ending in project M27 and also given that the understanding of the work in the project has evolved and solidified considerably, it is expected that no drastic changes will occur until the end of the work.

The requirements expressed here describe essentially the following aspects of the SCRATCH outlook:

- SCRATCH methodology requirements, or requirements upon the processes that companies adherent to the SCRATCH premises need to follow in order to produce secure IoT software.
- SCRATCH toolkit requirements, understood not so much as individual tool requirements but more of the toolkit as a whole.
- Use Case requirements of the demonstrators produced in the project to evaluate the benefits of using SCRATCH. This correspond to the three main domains in the project:
 - o UC-Retail requirements
 - o UC-Police requirements
 - o UC-Smart Grid requirements

Following the discussion on the different formats for requirements that underlies the very width of the project's scope, the requirements are presented here in one of its simplest and more straightforward forms, a table with some categories for each of the requirements. This is done to be consistent with this deliverable being presented as a text document.

The ultimate reference for the requirements is however the SCRATCH Knowledge Base (see description section 2.1). They are available in:

https://trusttab.com/standards/scratch_requirement/list

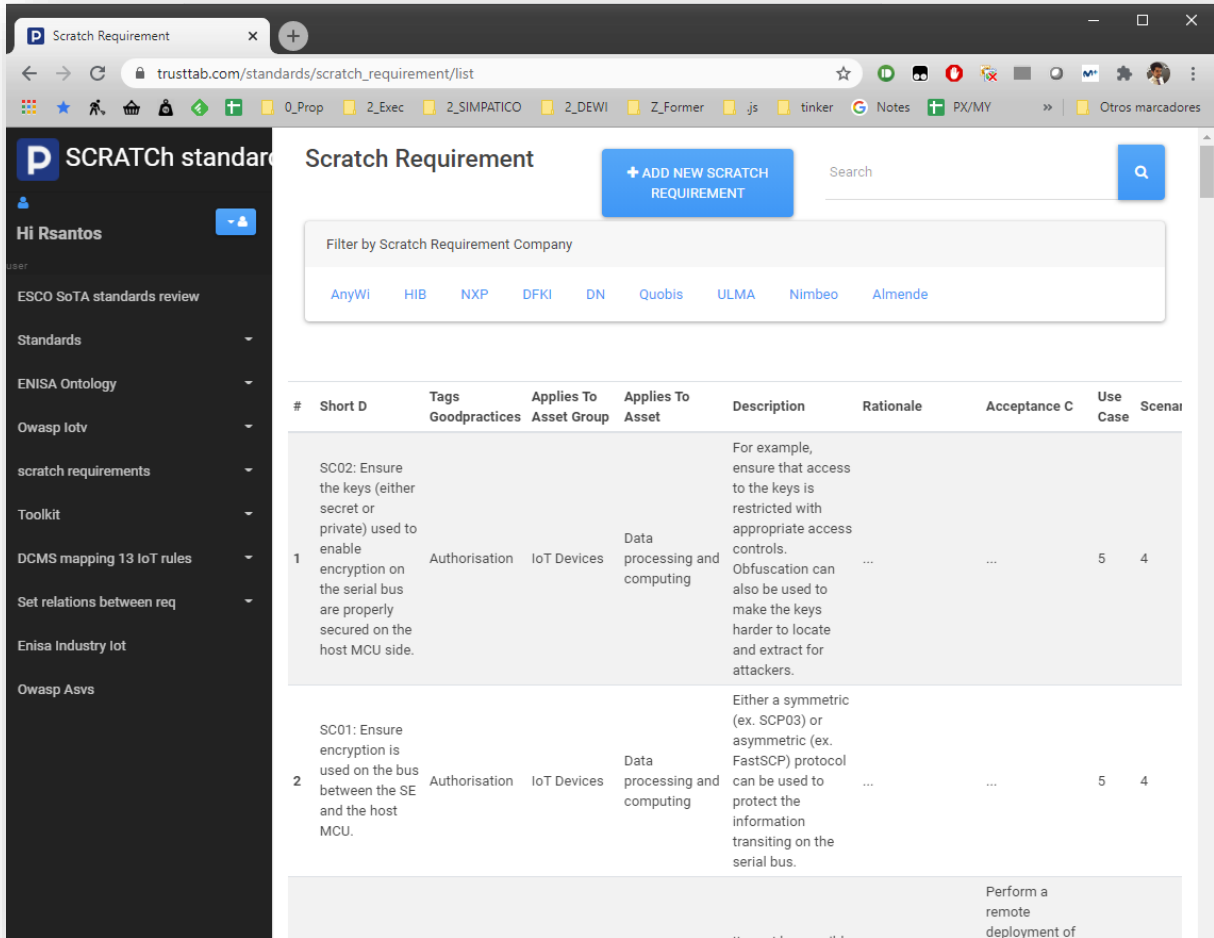


FIGURE 11 TRUSTTAB SECTION FOR SCRATCH REQUIREMENTS

For the reasons given before, the exact form of some of the requirements could be updated with regards to the table in this document, with the online version being considered more accurate.

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--------------------------|--|---|--|-----------------|-----------------|-----------------|--------------|
| 2 | Fail Safe State | Given that a Device securely receives a firmware or configuration update, and while executing the update a failure renders the device inoperable. Then a mechanism shall be in place to restore the device to its last safe working state. | To prevent failure and high repair cost of devices in the field, a mechanism that keeps the device in a connected state is needed | Push or Pull a defect image of configuration file, check if the device fails, check if the device reboots and reconnects automatically | 2 | 7 | medium | AnyWi |
| 3 | Gateway Attestation | The device shall provide functionality to attest part of its state to external entities, e.g., based on TPM measurement of system state | During operations, operator likes to monitor health status of device with strong cryptographic guarantee. Prove health towards system auditing. | Modify state and check if detected in attestation. | 2 | 5 | low | AnyWi |
| 4 | Gateway Trust Relation | IoT devices that are once securely registered to a specific gateway, can also register to other gateways that have a secure trust relation the initial gateway | To enable to build wider IoT network structures, e.g. smart cities. | Test if a registered IoT Device can also register with another gateway that has the trust relation with the original gateway | 5 | 7 | low | AnyWi |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|----------------------------------|--|--|---|-----------------|-----------------|-----------------|--------------|
| 5 | Secure OTA data/file transfer | Given the device software/firmware, configuration and applications have the ability to update Over-The-Air (OTA), and that the update server is secure. when updating the firmware or configuration. Then the update file is only transmitted if the sever and the IoT device negotiated a trust relation based on a hardware token and the file is encrypted. | To prevent updates that could compromise the whole system operational state | If one of the criteria is not met, the device should not perform an update | 2 | 7 | high | AnyWi |
| 6 | Standalone operation edge device | The edge device should be able to operate standalone when disconnected from the cloud and all essential features should continue to work with a loss of cloud connectivity and without chronicle negative impacts from compromised devices or cloud-based systems | this feature will make the system hardened against attacks or unintended failure of other systems, without compromising its own integrity. E.g., cloud connectivity lost the system can still operate with its connected nodes. Los of connectivity to a node or group of nodes remaining nodes should remain operational. | When the connection between cloud and gateway or gateway and device is lost, defined essential features remain functional and can be tested. Essential features such as access control, encryption of storage. When a group of nodes are disconnected remaining nodes remain operational. | 5 | 7 | low | AnyWi |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|------------------------------------|--|---|--|-----------------|-----------------|-----------------|--------------|
| 7 | limit number of open TCP/UDP ports | limit number of open TCP/UDP ports to strictly needed for operation | Minimize the attack plane for a device | When a port scan is conducted on the device only specified ports are accessible | 2 | 7 | high | AnyWi |
| 8 | IoT device register gateway | The IoT Gateway should contain a registry of IoT devices that are allowed to connect with it, this registry should contain a communication profile of each IoT device. | The gateway or edge device can monitor connected devices and act directly and autonomously on deviation between communication profile and communication behaviour of a connected device | Force a connected device to behave outside its communication profile, check if the gateway executes the predefined policy. | | 7 | medium | AnyWi |
| 9 | IoT device monitoring by gateway | The gateway should monitor traffic with IoT device in its register, and compare traffic patterns against stored device profiles. deviations will be checked with the gateway policy manager; actions are dictated by the policy manager. | The gateway or edge device can monitor connected devices and act directly and autonomously on deviation between communication profile and communication behaviour of a connected device | change policy of gateway, e.g. to exclude a device from communication. check if the exception is detected communication is closed and message send to administrator. | 5 | 7 | medium | AnyWi |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|---|--|---|--|-----------------|-----------------|-----------------|--------------|
| 10 | activate new IoT device in infrastructure | If a new IoT device is detected and this device request a registration as IoT systems device, then the gateway will check the device against known configuration changes, in case of unknown change the gateway will send out a configuration change message administrator and to the device of the store manger | To prevent unauthorized devices to gain access to the gateway. | check if an unauthorized registration is detected and lead to the described actions, check if an authorized change does not lead to the describes actions. | 5 | 6 | high | AnyWi |
| 11 | Sphinx-needs editor | The SCRATCH workflow should provide a sandbox environment so that sphinx-needs text can be edited without making commits to the repository. | This reflects a particular frustration when kickstarting the requirements activity. A fork of this particular editor in GitHub could be made. | | | | Low | HIB |
| 12 | Methodology documentation | The SCRATCH workflow should be documented so that all levels of users in the company (developers, managers, business) understand the methodology. | Understanding continuous integration, automated builds, test cases and all of the concepts that we use is not straightforward. We should always keep in mind all actors that could potentially use the documentation. | | | | High | HIB |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|----------------------------------|--|---|----------------------------|-----------------|-----------------|-----------------|--------------|
| 13 | Various communication types | The secure communication architecture must support various use cases. Different use cases will have different aspects to protect. These will consist of a combination of the key concepts of security: -confidentially -integrity -availability -non-repudiation | | | | | | NXP |
| 14 | confidentially must be ensurable | confidentially “is the property, that information is not made available or disclosed to unauthorized individuals, entities, or processes” | | | | | | NXP |
| 15 | integrity must be ensurable | means maintaining and assuring the accuracy and completeness of data over its entire lifecycle | | | | | | NXP |
| 16 | availability must be ensurable | must be available when it is needed | | | | | | NXP |
| 17 | Secure storage in device | The demonstrator will implement storage of critical data in a secure manner. | | | | | High | HIB |
| 18 | Secure facial recognition | The Police demonstrator shall implement a secure algorithm for facial recognition. | This will include access to a secure profile of the face(s) to be recognised. | | | | High | HIB |
| 19 | Secure streaming | The Police demonstrator will stream the video capture from device to server in a secure manner. | | | | | Medium | HIB |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--------------------------------------|---|---|----------------------------|-----------------|-----------------|-----------------|--------------|
| 20 | Authentication | Users need to be authenticated, i.e. it needs to be verified that they are who they claim to be. | | | | | | DFKI |
| 21 | Authorization | As soon as a user is authenticated, their permissions need to be determined. | | | | | | DFKI |
| 22 | Session Management | User sessions need to be tracked via unique and randomized tokens. | | | | | | DFKI |
| 23 | Error and Exception Handling | In order to prevent information leakage, errors and exceptions have to be caught and processed in a way that gives meaningful information to users, extensive information to maintainers, and no useful information to attackers. | | | | | | DFKI |
| 24 | Deployment Environment Specification | The environment in which the software shall be deployed needs to be determined. | In the case of trunk and test environments, the environment needs to be optimized for the software's needs. However, in live environments, it may be necessary to accept pre-defined conditions and optimize the software to fit these. | | | | | DFKI |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--------------------------|--|---|--|-----------------|-----------------|-----------------|--------------|
| 25 | OS Agnostic Middleware | The SCRATCH secure element integration middleware shall be operating system agnostic. Hence, it should support standard real time operating systems (RTOSs) like Arm Mbed or FreeRTOS. | The RTOS in IoT devices is device and application dependent. | Passed compatibility test with different RTOS. | 2 | | | DN |
| 26 | Certified Middleware | The SCRATCH secure element integration middleware should consider certification according to appropriate certification schemes, e.g., BSI Cryptographic Service Provider (CSP). | For application in some domains, certification of the product is required. If the middleware is used in this domain, it shall be possible to certify a device that uses the middleware. | Design review. Pass of penetration testing tools. | 2 | | | DN |
| 27 | Certified Secure Element | The SCRATCH secure element should be certified according to appropriate certification schemes, e.g., Common Criteria with appropriate security target. | For application in some domains, certification of the product is required. If the secure element is used in this domain, it shall be possible to certify a device that uses the secure element. | Passing domain specific certification of secure element. | 2 | | | DN |
| 28 | PSA Compliance | The SCRATCH secure element integration middleware should be compliant to Arm Platform Security Architecture (PSA) or similar frameworks. | To integrate the middleware into the application. | API compatibility test. | 2 | | | DN |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--------------------------|---|--|--|-----------------|-----------------|-----------------|--------------|
| 29 | Vulnerability Scan | The SCRATCH secure development platform shall include an automated Common Vulnerabilities and Exposures (CVE) scan as part of the continuous integration (CI) pipeline. | Reduce number of known vulnerabilities in releases. | Tool detects defined set of CVEs in project. | 2 | | | DN |
| 30 | Code Analysis | The SCRATCH secure development platform shall include an automated code analysis as part of the continuous integration (CI) pipeline. | Reduce number of new vulnerabilities in releases. | Tool detects test defined set of vulnerabilities in project. | 2 | | | DN |
| 31 | Test Framework | The SCRATCH secure development platform shall integrate a framework for tests at all stages of the continuous deployment pipeline, e.g. after build, staging, and deployment. | Increase quality of releases. | | 2 | | | DN |
| 32 | Gateway Smoke Tests | The SCRATCH gateway device shall support continuous deployment of tests for smoke tests at customer staging and productive environment. | Ensure compatibility of releases. | | 2 | | | DN |
| 33 | Authenticated Logging | The SCRATCH secure element integration middleware shall support authenticated logging. | Obtain authentic security related feedback from devices. | | 2 | | | DN |
| 34 | Confidential logging | The SCRATCH secure element integration middleware should support encrypted logging. | Enable logging of confidential information. | | 2 | | | DN |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--------------------------------|---|--|----------------------------|-----------------|-----------------|-----------------|--------------|
| 35 | Authenticated Firmware Upgrade | The SCRATCH secure element integration middleware shall support signed firmware upgrades. | For verification of updates, a verification key is required. The integrity of the key has to be protected. The secure element shall support hosting this key for integrity protection. | | 2 | | | DN |
| 36 | Confidential Firmware Upgrade | The SCRATCH secure element integration middleware should support encrypted firmware upgrades through a key encapsulation mechanism. | For confidential firmware upgrades, the encryption keys are shared by a series of devices to avoid encrypting firmware for individual devices. Confidentiality of these shared keys have to be protected at the devices by the secure element. | | 2 | | | DN |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|---|--|---|----------------------------|-----------------|-----------------|-----------------|--------------|
| 37 | Attestation | The SCRATCH secure element integration middleware should support remote attestation. | If the device attestation keys have to be store in the secure element, the middleware shall provide the interfaces that are required to perform device attestation (hashing of state information and signing) . | | 2 | | | DN |
| 38 | Secure boot | The SCRATCH secure element integration middleware should support secure boot, i.e. a verified boot chain from the hardware root of trust up to the RTOS. | For verification of the boot chain, a verification key is required. The integrity of the key has to be protected. The secure element shall support hosting this key for integrity protection. | | 2 | | | DN |
| 39 | Deployment Automation Device Management | The SCRATCH deployment automation components and tools shall provide a device management service for updating remote IoT components as part of the deployment process. | Unify the update process for different device types. | | 2 | | | DN |
| 40 | Secure Element personalization | The SCRATCH secure element integration middleware shall support personalization of the secure element with key material at the manufacturer or OEM. | For unique device identification. | | 2 | | | DN |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--|---|--|----------------------------|-----------------|-----------------|-----------------|--------------|
| 41 | Firmware downgrade protection | The SCRATCH device shall be able to reject firmware downgrade. | Mitigate downgrade attacks that re-introduce old vulnerabilities. | | 2 | | | DN |
| 42 | Unique identifier | Every SCRATCH device must provide a unique identifier, which cannot be changed after production via software. | Used for device lifecycle management. | | 2 | | | DN |
| 43 | Key storage | All keys and sensitive data shall be stored inside the secure storage. | Protection against attackers with local access to device. | | 2 | | | DN |
| 44 | Secure monitoring | Monitoring data secured and authentic. | To avoid attacker tampering with security feedback. | | 2 | | | DN |
| 45 | Device Reset | Successfully resetting the device back to operation shall need authorization | To avoid unauthorized manipulation of device state and communication infrastructure. | | 2 | | | DN |
| 46 | 3rd Party libraries | All third-party libraries and application should be up to date and have to be supported. | | | 2 | | | DN |
| 47 | 3rd Party libraries vulnerabilities | All third-party libraries and application should not include known vulnerabilities. | | | 2 | | | DN |
| 48 | Automated Software Deployment and Delivery process | The SCRATCH software and firmware deployment and delivery process should be automated and include least manual steps. | To maintain security in unattended environments. | | 2 | | | DN |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|----------------------------------|--|--|----------------------------|-----------------|-----------------|-----------------|--------------|
| 49 | Feedback of the fw/sw deployment | Feedback of the fw/sw deployment process shall be directly notified to the developers. | | | 2 | | | DN |
| 50 | Automated testing | SCRATCH deployment process shall include automated testing in different test environments. | Separation of staging from operations. Customer specific staging environments. | | 2 | | | DN |
| 51 | Automated unit test | SCRATCH deployment process shall include automated unit tests | To verify that implementations meet specification. | | 2 | | | DN |
| 52 | Automated system test | SCRATCH deployment process shall include automated system tests. | To detect incompatibilities with other components. | | 2 | | | DN |
| 53 | Automated smoke test | SCRATCH deployment process shall include automated smoke tests | To reject updates that break availability of services. | | 2 | | | DN |
| 54 | Documentation | SCRATCH documentation should be provided for all developed components and devices. | | | 2 | | | DN |
| 55 | Interface Documentation | All interfaces should be documented. | For integration into use case specific applications. | | 2 | | | DN |
| 56 | Interface Access Restricted | Interfaces allowing modification of settings or configuration data shall be restricted by using authentication mechanisms. | | | 2 | | | DN |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--|--|--|----------------------------|-----------------|-----------------|-----------------|--------------|
| 57 | Integration of SCRATCH devices | SCRATCH-ready devices must be integrable securely with other SCRATCH-ready devices or services. | | | 2 | | | DN |
| 58 | Integration of non-SCRATCH devices | Non-Scratch-ready devices must be integrable securely with scratch-ready devices or services. | Non-SCRATCH devices security should benefit from this integration and not open a security flaw | | 2 | | | DN |
| 59 | Migration of legacy devices | It shall be possible to migrate legacy devices into scratch ready device. | | | 2 | | | DN |
| 60 | Documentation of Migration of legacy devices | It shall be documented how to migrate legacy devices into SCRATCH ready device. | | | 2 | | | DN |
| 61 | Chatbot sensor values | The chatbot must allow obtaining sensor values from chat messages. | | | | | high | Quobis |
| 62 | Chatbot sensor list | The chatbot must provide a list of the sensors/actuators available in the system. | | | | | high | Quobis |
| 63 | Sensors presence information | The sensors must be able to publish their status through presence information | | | | | medium | Quobis |
| 64 | Chatbot message reception | The chatbot should be able to receive chat messages from the system, preferably using the XMPP protocol. | | | | | high | Quobis |
| 65 | Communication TLS version | All communications must be secure using TLS1.3 preferably, and TLS1.2 as backup in case 1.3 is not supported by any end. | | | | | medium | Quobis |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|---------------------------------|--|------------------|----------------------------|-----------------|-----------------|-----------------|--------------|
| 68 | Tests prioritisation historical | The test prioritisation framework shall obtain the test case execution historical, including: - detected error number - execution time - code coverage - successful execution cycle number | | | | | | ULMA |
| 69 | Test prioritisation report | The test prioritization framework shall execute test cases and provide a report with obtained results. | | | | | | ULMA |
| 70 | Energy loss monitor | The system must provide a module that detects and shows the energy losses. | | | | | | Nimbeo |
| 71 | Energy response module | The system must provide a module for the Response demand, where the consumer is informed about their consumption and the system offers solutions to balance the system load. | | | | | | Nimbeo |
| 72 | Energy interruptions management | The system must provide a module for the Management of interruptions, where AMI data is used for the early detection of interruptions and to help early restoration. | | | | | | Nimbeo |
| 73 | Energy predictive analysis | The system must provide a module for the Predictive analysis of network behaviour, which is a result of the execution of the algorithms that analyze the AMI data. | | | | | | Nimbeo |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|---------------------------|--|---|----------------------------|-----------------|-----------------|-----------------|--------------|
| 74 | PKI API CSR | The provisioning service shall process X509.v3 certificate signing requests. | To provision devices, we need a certificate authority that processes the signing requests that are issued at device provisioning. | | 2 | | | DN |
| 75 | PKI API CRL | The demonstrator PKI shall provide an interface for managing and requesting certificate revocation lists. | For device lifecycle management we need to revoke devices at end of lifetime or if corrupted. | | 2 | | | DN |
| 76 | PKI API AUTH | The demonstrator PKI registration authority (RA) shall provide authentication mechanisms to restrict access to the CA. | We need to technically enforce a role concept such that only authorized entities are able to receive certificates. | | 2 | | | DN |
| 77 | TLS1.3 | The secure communication protocols shall support TLS1.3. | For secure communication at transport layer in the demonstrator, we plan to migrate to TLS1.3. | | 2 | | | DN |
| 78 | Retail IoT Identification | The SCRATCH retail demonstrator shall be able to identify SCRATCH IoT demonstrator devices. | | | 2 | | | DN |
| 79 | Retail IoT Provisioning | The demonstrator shall be able to provision SCRATCH IoT demonstrator devices with credentials. | | | 2 | | | DN |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--|---|--|---|-----------------|-----------------|-----------------|--------------|
| 79 | Retail IoT Monitoring | The demonstrator shall provide facilities for device monitoring including device health status. | | | 2 | | | DN |
| 80 | Retail network deception | The demonstrator shall use deceive TCP headers to defend fingerprinting. | | | 2 | | | DN |
| 81 | Retail network anomaly detection. | The demonstrator shall monitor the network for anomalous traffic. | | | 2 | | | DN |
| 82 | Retail Localization | The demonstrator shall be able to track coordinates of shopping baskets. | Based on customer localization, we trigger events like check out, advertisement. | | 2 | | | DN |
| 83 | Retail telemetry | The demonstrator shall provide a publish/subscribe based mechanism to distribute telemetric data like coordinates, temperature, power consumption of demonstrator devices. | | | 2 | | | DN |
| 85 | No vulnerabilities must be introduced by new libraries | All new libraries used must be analysed to detect any know vulnerability before be merged into the release branch. The toolkit must be able to warn the developer about any found issues. | It is not difficult to introduce new vulnerabilities when adding needed libraries to cover some new feature. The toolkit should let the developer know that this happened in an automatic way. | When a new library is added to the code it must be detected and the tool must check if it has any know vulnerability. | | | high | Quobis |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|--|--|---|---|-----------------|-----------------|-----------------|--------------|
| 86 | Licenses of libraries can be easily listed and check by developers | The developer must be aware of the licenses of the libraries used in any piece of software since it may not be compliant with the license of the IoT software. | Open source libraries are commonly used in commercial software, but there are constraints about what libraries can be used depending in the licenses. This must be checked to avoid legal issues. | When a new library is added the licenses of the used libraries is checked and showed in an clear way to the developer | | | medium | Quobis |
| 87 | Integration of Crownstones | Crownstones must be implemented in the use case demonstrator and connected to important devices, including particularly point of sales terminals and ATMs as well as monitoring devices. | In order to explore the security benefits of Crownstone monitoring of important devices in a retail store, they must be connected first. | A point of sales terminal is connected to a Crownstone; an ATM is connected to a Crownstone; a server is connected to a Crownstone; if present, a gateway is connected to a Crownstone. | 2 | | | Almende |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|---|---|--|---|-----------------|-----------------|-----------------|--------------|
| 88 | Analysis of power measurements from Crownstones | Power usage of connected devices should be measured using Crownstones and analysed for insights important to the use case, e.g. when it comes to the power bill for the store as well as fluctuations indicative of threats to the store and customers. | Various forms of tampering will involve sudden changes in power usage, e.g. because a device is being surged or turned off. It should be possible to detect this. Secondly, relevant to the use case but not necessarily a security matter, it may be interesting to observe power usage within a store and compare the demand of the various important devices. | Measure power usage of connected devices in the demonstrator and look for useful results; notice any devices that are outstanding for high power usage; simulate various attacks and look for symptoms in measured power usage. | 2 | | | Almende |
| 89 | Security monitoring through Crownstones | By measuring power usage, and particularly by detecting drops in power usage, it may be possible to detect device failure and other malfunctions. This should be tested and demonstrated. | Whether it is a conscious attack or an accident, failure or malfunction in important devices is worth detecting and alerting staff about, and may even indicate immediate danger. This avenue of monitoring should be explored. | Detect device shutdown and unusual power surges. | 2 | | | Almende |

| <i>ID</i> | <i>Short description</i> | <i>Description</i> | <i>Rationale</i> | <i>Acceptance Criteria</i> | <i>Use Case</i> | <i>Scenario</i> | <i>Priority</i> | <i>Owner</i> |
|-----------|------------------------------|--|------------------|---|-----------------|-----------------|-----------------|--------------|
| 90 | Secure update of Crownstones | It must be possible to deploy Crownstone firmware updates securely and remotely into the demonstrator. | | Perform a remote deployment of Crownstone firmware to the demonstrator without violating SCRATCH firmware update security requirements. | 2 | | | Almende |

4. Conclusions and next steps

4.1. Conclusions

In this deliverable we have presented the following:

- The final account of the work undertaken in tasks T1.1a and T1.1b regarding the expression, formatting and management of requirements, particularly for the SCRATCH projects but with conclusions also applicable for requirement tools in the SCRATCH toolkit. This vision has evolved during the project and here we present the findings and lessons learnt from the process.
- The month 27 SCRATCH project requirements themselves, that define aspects of the SCRATCH methodology, overall toolkit usage and finally the particular use cases in which the SCRATCH technology is evaluated in the course of the project.

With these results we have completed the goals for task T1.1 in SCRATCH and delivered the final strategy for requirements collection in the project's methodology as well as the final release of the formally documented requirements. All subsequent changes to the requirements will be updated in other outlets such as the requirements section of the TrustTab Knowledge Base website³.

4.2. Next steps

Task T1.1 has formally ended with the release of this document. However, requirements elicitation in a DevOps focused project such as SCRATCH is never a fully finished process. Thus, we will continue to apply the requirements management methodology and tools presented here and produce new requirements that we will use internally to drive the final months of the project.

³ TrustTab website: https://trusttab.com/standards/scratch_requirement/list

5. References

- [1] IEEE Computer Society. Software Engineering Standards Committee, & IEEE-SA Standards Board. (1998). IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers.
- [2] Sphinx-needs website: <https://sphinxcontrib-needs.readthedocs.io/en/latest/index.html>