SCRATCh

SECURE AND AGILE CONNECTED THINGS

Public Document on State of the Art in Secure DevOps

Version history

Date	Version	Author	Comment
27/07/2020	0.01	Morten Larsen	Assembly of initial bullet points from partners
07/08/2020	0.02	Morten Larsen Marcell Marosvolgyi	Partner bullet points organised into chapters
21/09/2020	0.05	Franklin	started with introduction and context
24/09/2020		Till Witt	added NXP input, added CIT graphics on device identity lifecycle
24/09/2020	0.07	Morten Larsen Angel Lagares	Added Nimbeo contribution, reorganised paragraphs in introduction
24/09/2020		Karsten Sohr	Added Section 5.1 (Design Phase)
17/11/2020	0.09	Asier Larrucea	Added/Updated Section 2.1 and 3.1.
30/11/2020	0.10	Morten Larsen	Added tables for SME and UC concerns
14/12/2020	0.10a	Morten Larsen	Typo corrections
14/12/2020	0.10b	Franklin	Changes to intro and design phase
11/01/2021	0.10c	Asier Larrucea	Added Section 5.3 Build/Test (Integration)
15/01/2021	0.10d	Karsten Sohr	Added entries to the tables for SME and UCE concerns
02/01/2021	0.11	Peter Guenther	Update Section 5 intro and 5.2
05/02/2021	0.11a	Karsten Sohr	Revision of Sect. 5.1, Literature added
09/02/2021	0.20	S. Duque Anton	Revision of 5.5, added text, formatted and replaced tables
22-02-2021	0.50	Franklin Selgert	small changes first part,added several remarks e.g. on requirements and moved a section in chapter 5.1
01-03-2021	0.60	Franklin Selgert	Changed column name in table page 17
01-03-2021	0.60a	Karsten Sohr	Changed citations in chapter 5.1 and 5.5
08-03-2021	0.61	Morten Larsen	Some cleaning. If too heavy-handed, let me know.
11-03-2021	0.70	Morten Larsen Marcell Marosvolgyi Franklin Selgert Simon Duque Anton	Editing for polish and clarification
11-03-2021	0.71	Morten Larsen	Implemented to some actions from editing meeting
11-03-2021	0.72	Morten Larsen	Removed first section which was only for internal use. Added captions to tables.
12-03-2021	0.72	Franklin Selgert	solved my remakrs and edit of tables chapter 4.1 and 4.4 (assignments of the tables are missing)
17-03-2021	0.73	Morten Larsen	More cleaning up based on input and comments to v0.72
19-03-2021	1.00	Franklin Selgert	Finalized version for publication

Table of Contents

1 Introduction

The context of this paper is best described by a quote from the IRTF taskforce (RFC 8576) on SoTA IoT security:

> *The Internet of Things (IoT) concept refers to the usage of standard Internet protocols to allow for human-to-thing and thing-to-thing communication. The security needs for IoT systems are well recognized, and many standardization steps to provide security have been taken -- for example, the specification of the Constrained Application Protocol (CoAP)*

Taking this as a starting point, a strong correlation between internet development and IoT development is set. In the same paper from the IRTF the obvious differences also are highlighted. And the answer to tackle the Security issues are mostly protocol-related: let's increase security on the protocols. To answer the question if this is enough one should take a step back and look at the current Internet development playing field. To quickly summarize the tremendous amount of documentation and study in this field one can distinguish three major sides to the IoT security challenge.

1. Process
2. Communication protocols
3. Distribution and control

To develop a secure IoT system one need to consider the constraints of the system and applicable regulation, how to comply to those constraints (process) and finally how to keep in control of a system or IoT deceives in the field (distribution and Control)

This paper takes a holistic view on the question of IoT security as it is a combination of Process and the appliance of technical know-how. No process can lead to a false implementation of SoTA security protocols and no means to detect this. Process and no technical know-how will lead to a well documented system including the security leaks.

As described by the IRTF taskforce Internet things applied to IoT and steps are taken to improve on the technical know-how. This white paper takes the common process practice of many internet companies, namely **DevOps**, and reviews this against prevailing SoTA opinions, within the aspect of open systems. The emphasis is on the use of DevOps process in SME's type of companies.

1.1 Overall considerations about IoT

IoT environments are heterogeneous, distributed and hard to operate from a security perspective.

The general objective of IoT systems is that they can adjust to the behaviour and conduct of the client so that they can expand the overall usefulness of the system. This move towards information-driven assembling is set to change the way that IoT system designers will work in the near future.

The overall aim of IoT systems is that expanding on the usual internet paradigm, where business, government and customer collaborate, IoT will help them manage their interaction with the physical world. IoT will have a colossal effect on the wide scope of the market area including yet not restricted to following [1]:

1. Industry: Industry can utilize IoT to enable sophisticated sensor and actuator networks: detecting area, surveying gear execution, controlling and checking tasks, controlling and observing HVAC(heating, ventilation and cooling) and so on.
2. Agribusiness: IoT can have a big effect on the farming area for example soil examination, enabling increased crop growth or with precision metering that enables a second harvest in one year.

3. Automobile industry: IoT has its potential in-car field regarding city traffic stream, the board of leaving, perceptive key passage, vehicle area, checking vehicle wellbeing, hostile to burglary and so on.
4. Retail: IoT innovation can be utilized for following of resources, looking after stock, displaying and to oversee gracefully retail chains.
5. Ecological: IoT can be used for following endangered species, anticipating the climate and providing key data for policy makers.
6. Medical services: IoT can be utilized for Telehealth, embedded and wearable gadgets. Some gadgets can help individuals to deal with their wellness by tracking and analysing their exercise patterns and routines.
7. Military: IoT innovation can be utilized in the military to screen troops actions, to oversee assets and military logistics.
8. Smart Homes: IoT can be used for perceptive home to control lighting, security devices, warming, cooling and smart metering.

Nonetheless, this future is brimming with threats from the perspective on security. To address the dangers these associated IoT gadgets represent, it is essential to think of a system that will guarantee that each "thing" is periodically checked and its security status assessed.

The IoT environments dictate the tools and methods used and should not be subject to change. IoT environments are very much domain-specific and designed with a purpose in mind. Consequently, security methods need to be tailored to application and network protocols used. That means any IoT monitoring solution that is designed as “general purpose” has to be adaptable to different use cases and structures of IoT environments.

The trend and pull of the market is to transform DevOps into Secure DevOps by using practices and activities already proven to improve security of software applications. The objective is to create a secure environment during the whole life-cycle defined in the DevOps cycle, from the inception of the project to the maintenance phase, including analysis, planning, design, development, build, test, deployment, and support. While the security focus is usually focused on the ‘operations’ side of the cycle, a complete secure environment has to cover the entire organization, therefore all the layers and phases have to include security barriers and controls, and the company has to have a risk-security plan which includes management and training. In order to achieve these goals securing DevOps requires not only the new techniques, tools and technologies available, but also a different organizational approach which includes new strategies, policies and business processes.

The rest of this document will be organized as follows:

- In **Chapter 2** a more complete overview of the **High-level goals** is provided, building on the general outlook of this Introduction.
- **Chapter 3** deals with the **Device Identity life-cycle** aspects necessary in secure IoT ‘things’.
- **Chapter 4** details the different **DevOps Phases** and their challenges as analyzed in the literature and by the SCRATCh consortium. A number of sub-sections are devoted to the different phases and each of the DevOps phases subsections present two tables where SME and use case concerns are listed. The entries in the tables are categorized by aspect and contributor name.
    - Section 4.1 details the **Plan** phase.
    - Section 4.2 describes the **Code** phase.
    - Section 4.3 explains the important **Build and Test** phases
    - Section 4.4 provides insight in the **Release and Deploy** phases.
    - Finally, section 4.5 deals with the **Operation, Monitor** phases and presents an introduction to the **End-of-Life** aspects.

- The document ends with some **Conclusions** in **Chapter 5** and lists all used **References** in **Chapter 6**.

The DevOps phases subsections each have two tables where SME and use case concerns are listed. The entries in the tables are categorized by aspect and contributor name.

2 High level goals

The exponentially increasing connectedness of software applications and their pervasive distribution in modern industry makes security a top-level critical factor for any software producer to deliver attractive products. Over time, different approaches have been taken, starting from a human-centered approximation in which the developer was responsible of both the functional and non-functional (such as security) factors. These initially included manual code inspection and QA analysis with increasing scale: the prototypical sole programmer in many 80s PC applications led to more collective approaches in the 90s (following the famous Linus' law stating that "given enough eyeballs, all bugs are shallow") and continuing with increased focus on effective software production processes such as extreme programming (XP), Lean and Agile approaches such as Scrum and, in the recent years, the DevOps consolidation as the leading methodology combining software production, deployment and operation. The integration and cyclic approach of DevOps is seen as the main driver for improving on software quality.

But while DevOps alone increases the efficiency of software production and system operations, it is often not enough to satisfy all of the security goals for said system. Thus, extensions such as DevSecOps appeared to fill in the gap. The adoption is slowly increasing, especially in fields with high security stakes such as critical infrastructures and IoT deployments. Companies are increasingly adopting these solutions: it is projected than in five years, penetration of SecDevOps will reach around 20 to 50% of the industry[1].

Security on software often takes on a lower priority than other functional areas for most application developers, but this is so much more prevalent for SMEs, where the availability of security-aware developers is low and the pressure to deliver solutions to market is very high, especially in such dynamic markets such as IoT. It is forecast that 44% of SME plan to invest in resources related to the IoT, yet only 20% plan to invest in cyber security[2].

It is then clearly needed to develop solutions to ensure secure software is produced, but taking into account the difficult curve of adoption for the industry players with less resources such as SMEs: simple process management and traceability from use-case to test-case and preferably simple linking to potential vulnerabilities can yield massive benefits to these users, and SCRATCh is well-positioned to deliver results in this direction. In the following lines we deliver the high-level goals that have been collected from the SMEs in this line of industrial research and collect the key drivers for innovation in the field.

Goal 1: We must increase the prevalence of secure design and secure development by making all produced elements more understandable and accessible. As it stands, it takes extensive research and expertise to thoroughly secure a system, which is a core reason for the need for improvements for SMEs. With *new, more user-friendly documentation* and tools, understanding of the threats and the power to combat them can be put in the hands of the technical lead even if they are not a security expert.

Goal 2: Automation of the software production process is essential for all industry players to adopt security enhancing solutions: Automated tools are required to increase the level of security for the variety of applications provided in IoT scenarios. A tool framework that quantifies the security according to known vulnerabilities or heuristics improve the inclusion security features to reduce the

[1] Fortinet: key findings from the 2019 State of DevOps Security Report: https://www.fortinet.com/blog/industry-trends/key-findings-state-of-devops-security-report

[2] "Hackers' delight: Small businesses investing more in Internet of Things, less on cybersecurity", published 4 March 2020 by CNBC.com https://www.cnbc.com/2020/03/04/small-businesses-want-to-invest-more-in-iot-despite-cyberthreats.html

development time of secure systems. For example, a test prioritization tool to be provided within SCRATCh project helps speed up the development of security tools and systems in a semi-automatic manner as it allows to reduce the time for testing. Also, compilation of the Knowledge Base is used to search for the security issues detected in the whole community and can be integrated to flatten the learning curve for smaller SMEs.

Goal 3: Focus improvement in aspects that are critical for IoT applications as this is a critical security field now that will only grow in the near future. *Availability* and *Quality of Service* (QoS) are crucial for IoT environments. IoT devices are used in distributed environments with application specific, heterogeneous entities. Services rely on input from IoT devices, other services require information from servers in order for the user to obtain information and interact with the system. Furthermore, security and safety relevant features, such as physical intrusion detection systems, e.g. door and window sensors or industrial safety switches require uninterrupted connectivity. By techniques such as redundancy and shadow nodes, dynamic routing, as well as heartbeat packets, the disruption of service can be detected and mitigated. Thus, the quality of service can be ensured despite loss of individual devices.

Through our experience in SCRATCh and the continuous contact we have with the industry, we can also target particular secondary objectives for some of the domains of interest in the project such as:

- **Smart Retail:** Real-time in-store marketing through proximity-activated promotions that push notifications in our smartphones or wearables, and personalized CTA's on digital signage, among others. Additionally, store layout optimization as a result of consumers' behavioural data analysis and identification of hotspots and hottest items. **High availability**, secure **session management** and the **integrity** of the generated data are of this highest importance.
- **Smart Home:** One of the main goals of the smart home is to increase daily life by improving user comfort. This is carried out automating some routines as well as giving homeowners the power to manage their home systems remotely. A smart home provides the ability to control electronics and appliances from a smartphone, tablet or laptop. It adds an extra level of convenience and comfort while removing manually maintaining home systems. The critical aspects in Smart Home are **privacy keeping** and **availability**.
- **Wearables:** Smart wearables collect and analyse data, and in some scenarios make a smart decision and provide a response to the user and are finding more and more applications in our daily life. Wearables are a promising solution for the objective, reliable, and remote monitoring, assessment, and support through ambient assisted living. **Privacy** is also extremely relevant in this field, as very sensitive data is collected.
- **Smart City:** the goal of smart cities is to enhance the use of public resources, improving the quality of the services offered to the citizens whereas reducing the costs of the public administrations. This goal can be carried out by the deployment of an urban communication infrastructure that supplies unified, simple, and cheap access to all public services, providing potential synergies and improving transparency to the citizens. In this domain, **scalability** and interoperability of the solution is a critical concern, since IoT deployments can be huge and multivendor in sprawling cities. **Secure maintenance and updates** of the large network are also critical.
- **Smart Grids:** It must enable active participation by consumers in demand response. Moreover, it must operate strongly against physical and cyber-attacks. It also must host different storage options. It allows the creation and development of new products, services and markets. Finally, it optimizes asset utilization and operating efficiency. Given the opportunity in this domain for fraud, the critical aspect in this domain are **authentication** and **attestation** of the devices and **detection of intruder** devices.

- **Industrial Internet:** it is used in many industrial areas, such as manufacturing *(*Industry 4.0*)*, logistics, oil and gas, transportation, energy/utilities, mining and metals, aviation and other industrial sectors. Here the most critical objective is **reliability** to ensure smooth operation of the factories.
- **Connected Cars:** in this area of IoT, the industry is working to improve to introduce incentives for innovation and investment, leading with light-touch regulation that will allow the market to scale while building trust and confidence of consumers, promoting research and development programmes for connected and autonomous vehicles and supporting services, applications and network industry-led standards and interoperability. **High availability** and Safety is paramount, as well as aspects related to the **certification** of the software.
- **Connected Health:** the aim is decreased operational cost, better patient experience, reduce errors, improved outcomes of treatment, improved disease management. This is a very wide subject with many sub domains, but **privacy** and **reliability** are almost always the most important aspects.
- Finally, **Smart Transportation:** In this area, the industry is Enhancing traveller experience with improved customer services, increasing safety with sensor data tracks, reducing energy use and congestion organizing the use in real-time data to better scale resources and meet demands, better operational performance monitoring critical infrastructure and developing efficient processes to minimize operating costs and improve system capacity. **High availability** of the secure solution and efficient **scaling** up are thus the critical assets to consider.

With these high-level goals of the entire SecDevOps for IoT in mind, in the following sections we jump deeper into some of the key topics that are relevant in this area. We will start by providing an overview of one transversal aspect for IoT (Device Identity) that pervades every subsequent consideration and then we will jump into the whole DevOps cycle and its phases, providing examples of typical challenges, available tools and solutions proposed by SCRATCh and the found connections with its Use Cases and the approaches followed by the different SMEs in the project.

3 Device Identity life-cycle

To ensure a Secure DevOps process and thus a secure product, the device identity life-cycle must be considered during the entire DevOps process, up to and including the decommissioning of a IoT system

Depending on the executing (where and how) of each step, different actions within the DevOps cycle needed to be considered as well. E.g., the trust provisioning can happen in a form that a root certificate is provided with the device, allowing individual credential creation by the IoT developer. The device could also be provisioned with pre-generated keys.

The exchange of this information between manufacturer and IoT-device producer must happen with the following DevOps process in mind thus impacting the **plan phase.**

During the plan & code phase - the integration of a physical security token such as a Secure Element must be considered as it may require a different architecture handling the certificates vs. a simple password-based approach. The physical security token will greatly increase the products resilience against attacks if integrated correctly but will increase the complexity of the DevOps pipeline.

The device operation will impact on the entire DevOps Cycle as well.

4 DevOps Phases

In this document we follow the commonly used Phasing of the standard DevOps cycle as found in different publication on the internet, eight phases: plan, code, build, release, deploy, operate, and monitor. In this section, we give an overview of the state of the art with emphasis on *security and IoT*. The current SotA concentrates on the parts with high potential for automation like build, release, deploy, and monitor. To establish security by design principles, we put more emphasis on the plan phase because in the plan phase we set the course for a secure system. Therefore, we subdivided the plan phase into a Requirements phase and a Design phase.

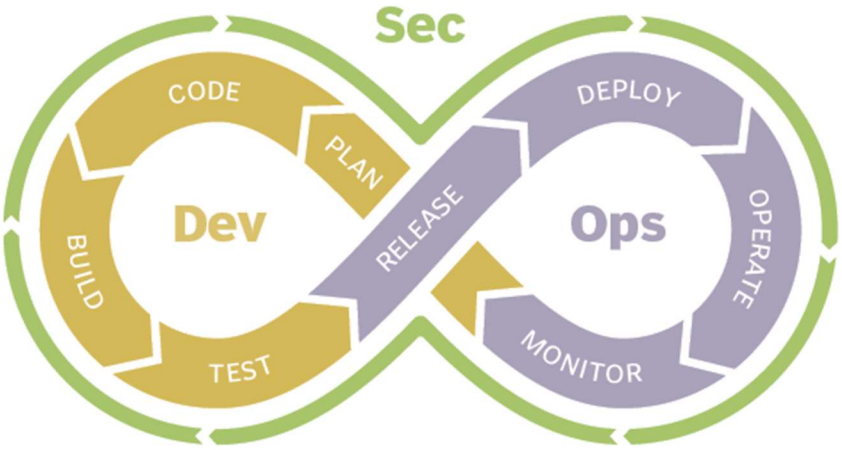

FIGURE 1: SECDEVOPS CYCLE[3].

First, we recall the definition of the SecDevOps Phases from Figure 1. Then, in the following subsections, we provide the SOTA for each phase.

1. **Plan Phase, consists of**

    1. **Requirements Phase** – During this phase the user requirements are documented and analyzed. Once the requirements are agreed, they are prioritized, and change management is enforced. This phase also enables requirements tracking through the rest of the SecDevOps phases.

    2. **Design Phase** – During this phase the requirements are transformed into complete and detailed system and test design specifications. The design is typically broken down to facilitate multiple releases (implementation plan, release plan, product road-map).

2. **Code Phase** – The design is realised during this phase through a process of software code development, and typically includes supporting mechanisms such as code reviews and developer-level unit test development.

3. **Build Phase** – Once the agreed coding tasks for a given release have been completed, the code is committed to a code repository where it is eventually merged with a new shared codebase. All submitted code merges are reviewed and then an automated process is initiated to build a new software release and perform end-to-end, integration, and unit testing.

[3] Quoted from https://searchitoperations.techtarget.com/definition/DevSecOps

4. **Test Phase** – Once a build is completed successfully it is automatically deployed to a staging environment for more comprehensive testing. This involves a series of manual and/or automated tests to validate the design and ultimately the requirements. The process may loop over the code-build-test phases until the agreed quality measures have been met. If more fundamental changes are required, the process may loop back to the design and even the requirements phase.

5. **Release Phase** – Once the pre-defined quality criteria have been met, the build is considered ready for release. This phase may include various automated scans of the release artefacts to detect common vulnerabilities and exploits, and to ensure compliance with code re-use practices. The release is tagged for traceability, deployment packages are prepared, and the release is marked ready for deployment.

6. **Deploy Phase** – A completed release is deployed into the target production environment, ready for use. Deployment typically includes mechanisms to minimize system downtime and may also ensures that it is possible to roll back to a previous release in the event that the new release has critical defects.

7. **Operate Phase** - The new release is in use and performs the intended functions. The operations team ensures the availability and performance of the system through mechanisms such as redundancy, load balancing, and scaling. For critical systems, disaster recovery and business continuity plans are put in place and regularly tested.

8. **Monitor Phase** – While the system is in operation, it also has to be monitored. As such, the last two phases of the cycle happen in parallel. During this phase data is collected about the performance and functionality of the system, which ultimately may be fed back to the start of the SecDevOps cycle.

4.1 Plan Phase (requirements and design)

The plan phase is about "what are we going to develop". This is the phase where one should set system constraints, posed by regulation, environment, type of application, etc. It is also the phase where changes can be made with a minimum of impact on the work. If one does not design a secure method to upgrade a running system from the start, it will be costly to do it later on. The question is what are the methods currently used to make a secure design.

In the puppet-state of DevOps Report 2020[4] (page 32) a distinction is made in four categories of companies: Operational mature, Engineering driven, Governance focused and Ad-hoc. The majority of smaller companies is divided over the Engineering driven and Ad-hoc type of companies. On security issues those type of companies are depending on the ability to quickly re-mediate vulnerabilities.

Currently design is not a formal phase of the DevOps practice, it resides under the "Plan" phase as described in (https://medium.com/taptuit/the-eight-phases-of-a-devops-pipeline-fda53ec9bba): The Plan stage covers everything that happens before the developers start writing code, and it is where a product manager or project manager earns their keep. Requirements and feedback are gathered from stakeholders and customers and are used to build a product road map to guide future development.

[4] https://puppet.com/resources/report/2020-state-of-devops-report/

Most articles on DevOps are not specific to security in this phase. For most SMEs practicing DevOps, being an Ad hoc or Engineering driven company, security is applied or forgotten in a later stage.

What should be done and why:

The design phase encompasses different subtasks including

- Security requirements engineering,
- Abuse case elicitation,
- Attack surface analysis,
- Threat modelling and architectural risk analysis,
- Definition of a security architecture.

First, high-level **security requirements** or security objectives, such as "Medical data must be confidential and authenticated", must be elicited first and documented. As introduced by Sindre and Opdahl [4], security requirements can additionally be deduced from negative use cases, called **abuse cases**, e.g., "an attacker tries to access services without credentials" or "an attacker uses corner cases (e.g., MAXINT) to produce overflows".

As an example of security requirements:

All the IoT communication must use secure protocols. TCP connections between modern devices and the servers should use TLS 1.3 which brings a number of advantages such as shorter handshake times, session resumption, more efficiency and secure cipher suites. On the other side, there are IoT protocols transported over UDP, such as CoAP which must be transported over DTLS 1.2. The DevOps process should be able to verify that all the communications are transported over the last available version of those protocols at build time.

Other requirements can be found in several relevant standards and best practices like from ENISA, ETSI OWASP, IoTSF. Getting a subsection of relevant security constrains depends heavily on type of IoT system and the industry sector it will be implemented in.

Another subtask is determining the **attack surface** of the system/application. The attack surface often includes the entry and exit points of the system under analysis. At these locations, trust boundaries are crossed, which gives adversaries opportunities to attack the system/application.

The main subtask of the design phase is performing a **threat analysis** combined with a subsequent risk rating step. Architectural risks are usually discussed with the help of architectural diagrams, e.g. UML diagrams or **dataflow diagrams** (DFDs), and systematically documented. One widely used methodology is Mircosoft's **Threat Modelling**, also called STRIDE [5]. The acronym STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, and Privilege Elevation. These attack categories are then systematically applied to elements of DFDs, e.g., connections, processes, and data stores. As a result, threats specific to the system represented by the DFD are identified. An example DFD representing a simplified IoT architecture can be found in Figure 2.

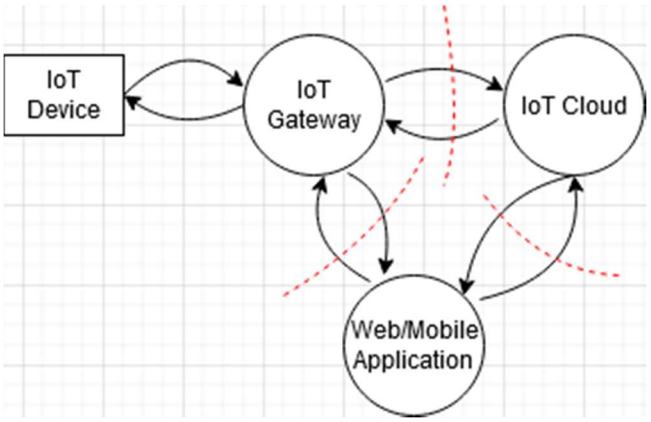

FIGURE 2: AN EXAMPLE DFD DESCRIBING AN IOT ARCHITECTURE (SIMPLIFIED) - DFDS CAN BE USED TO SYSTEMATICALLY IDENTIFY THREATS/RISKS IN SOFTWARE AND SYSTEM ARCHITECTURES, RESPECTIVELY.

Having identified threats, they are rated according to a specific **risk rating methodology**. Typical risk rating methods are DREAD [5], FAIR [5], and OWASP Risk Rating Methodology [6].

A **security architecture** is then built, which mitigates the most serious risks that have been identified in the risk rating step. In this step, the architectural descriptions of the systems or software such as DFDs can be augmented with specific measures [7]. Typical security measures include the encryption of specific connections and integrity protection for data stores/data.

Although the aforementioned steps are essential for a security-by-design approach, it is not easy to apply them in an SME-based environment. The following tables summarize the design phase-related SecDevOps concerns and gaps relevant to SMEs , and the specific concerns related to the SCRATCH use-cases.

TABLE 1: SECDEVOPS CONCERNS SME'S PLAN PHASE AS INDICATED BY PARTICIPANTS SCRATCH

Aspect of concern	SotA of topic of concern for SMEs	Why should this be of particular concern to SMEs?
Requirements	Lack of tool support for requirements management.	Commercial solutions are costly and developed primarily for large engineering companies.
Requirements	Requirements management is ineffective	In a small, specialised team, when working on a product that grows and changes over time, requirements management as we understand it now may completely fail to contribute, and its rigidity may negatively impact DevOps and increase technical debt
Requirements	Requirements management and other design tools do not offer a perspective for business product owners.	Products are led by business decisions yet the design tools are mostly for developers. Some way of connecting the business perspective is needed.
Requirements	Requirement management tools does not offer security specific options to take security aspects into considerations depending on the type of requirement and its transport layer.	Security requirements are typically defined as non-functional requirements, but there is no a systematic way to derive them from the functional requirements.
Threat modelling	Lack of known or established security modelling tools.	Security by design mandates that security architectures are documented. The effort for documentation must be kept as low as possible due to a lack of resources in SMEs. Automation of this

Aspect of concern	SotA of topic of concern for SMEs	Why should this be of particular concern to SMEs?
		step is desirable, but the necessary tool support is still missing.
Skill set	Security mindset is not always present. Engineers might need security training.	IoT solutions bring connectivity and new attack vectors to companies who might not have experience with network security.
Development priorities	Wrong assumptions of underlying technologies. Some secure communication protocols, are insecure by default; protocols like ZigBee (security broken) or Bluetooth LE (encryption often not implemented, access control not correctly implemented).	Implementing application security is often seen as unnecessary, resulting in an insecure implementation.
Methods	Lack of guide of best practices to systematically include security considerations at the requirement definition phase.	It would be useful to have a systematic approach to include all the security considerations which must be taken into account when a new requirement is defined. Depending on the type of requirement it would be helpful to have a list of considerations.
Methods	Security has a huge impact on code. This is easily missed by non-expert developers. Level of competence depicts the outcome	Although an adhoc approach by skilled engineers can deliver a secure system, compliance to a certain regulatory environment results in double work by retro explaining the fulfilment of essential constrains.

TABLE 2: SECDEVOPS CONCERNS PLAN PHASE AS DISCOVERED IN THE USE CASES OF SCRATCH

Use-Case	SotA of topic of concern for the given UC in the plan phase.	Why is this of particular concern to the UC?
Retail	Threat modelling and risk assessment is done on a regular basis. Domain knowledge is used to identify threats. Documentation of threats is done as paperwork.	More automation for management and documentation of threat models is required to reduce the effort for regular re-assessments during the execution of projects.
Retail	Carrying out architectural risk analysis (Threat Modelling/STRIDE) of UCs	Retail processes are security-critical. Security by design approaches, such as STRIDE, are prerequisite to securing security-critical processes and their architectures.
Smartgrids	Security aspects are intricate and not always perfectly known by designers	The security is a critical concern for Smart Grid software, therefore the methodological and guided approach to achieve successful security systems is needed.

4.2 Code Phase

In the code phase, the components from the plan phase are implemented. SecDevOps in the code phase tries to ensure that the code fulfils the security requirements from the plan phase. At the coding stage, we try to identify known vulnerabilities in the code, but also to prevent the introduction of new vulnerabilities.

The SOTA includes the following techniques to improve security. Most of them are not specific to security, but can also be used to address security as a non-functional requirement. Furthermore, some techniques are not specific to the coding phase but have to be considered also at the coding phase to achieve the overall goal.

- **Continuous Integration**: The process of continuously merging increments of the code back into to mainline code base.
- **Issue Tracking**: Track issues like defects and requirements and connect them with continuous integration. This provides a management view on the code base and development activities.
- **Unit Testing**: Functional units are part of the mainline code based. The tests are compiled and tested on a regular basis, e.g., when they are merged into the mainline code during continuous integration.
- **Code Reviews:** The code is regularly reviewed by team members and peers to reduce defects and identify new vulnerabilities.
- **Continuous Compliance**: The code is audited on a regular basis to maintain continuous compliance of a product to relevant regulations and standards.
- **Pair Programming:** A technique to continuously enforce code reviews. One team member implements the code while another member continuously reviews the new code.
- **Configuration as code:** Organize project configurations as part of the continuous integration process as part of the mainline code base. This improves traceability of the configuration and enables regular review or security scanning of the configuration.
- **Infrastructure as code:** Organize infrastructure definitions like e.g. build configurations as part of the code. This improves reproducibility and regular review of the infrastructure.
- **Static application security testing (SAST)**: Automated scanning of source code and configuration to identify vulnerabilities. SAST is the automated counterpart to code reviews and can be hooked into continuous integration.
- **Dependency Analysis**: Continuously scan the project dependencies, e.g., based on the project configuration, to identify the project dependencies. Based on the project dependencies common vulnerabilities and exposures (CVEs) can be identified.
- **Continuous Learning:** Continuously update the secruity knowledge of the development team. Build awareness for security and security background. Also learn from security feedback from the field like new or known vulnerabilities.

The most techniques rely on continuous integration for a consistent view on the shared code base and the traceability of code snapshots through the incremental development process.

The use of tools that help to prevent the introduction of security vulnerabilities is essential. During the code phase the system must be checked for vulnerabilities continuously, even when the project is deployed, the software and firmware have to be checked and updated for security reasons. These tasks have to be performed in a methodical and systematic manner, they also have to be performed considering time constraints, therefore the need of automated tools that ensure that the software is secure. These tools have to identify security flaws and it is recommended that they can interact with the issue tracking software of the company so the developers can be notified if a vulnerability is found. Thus, the organization should integrate the right security automated tools that continuously check the networks, processes, devices, etc. during the whole lifecycle to guarantee that the whole system is secure.

During the development phase, Static Application Security Testing (SAST) tools should be employed. These tools automatically scan the source code for security bugs, e.g., buffer overflows, race conditions, misused cryptography, hard coded passwords and keys, and SQL injections. They use

technologies from compiler construction to analyse the program code and often support different programming languages, such as C/C++, C#, and Java as well as different software libraries - in IoT environments this seems beneficial as of different technologies and languages are used within an IoT application. SAST tools can be run at certain points in time, e.g., during nightly builds. Quality gates should be introduced to only allow one to ship code without bugs that are rated as high or serious[5].

To provide the developer feedback from testing, it is often desired that the static analysis tools tightly integrate with the IDE. Then it is possible to show code bugs identified in SAST in the IDE.

On the other side, it is necessary to check the version and configuration of all the libraries related to protocols and data parsing, which can enable attack vectors in both the IoT devices and the servers. Detecting vulnerabilities at this stage means to reduce costs since the software will not reach any production environment and the fixes will be cheap and quick. When the IoT device has been already deployed any update may be expensive and, in some cases, very hard to carry out.

Specifics of SMEs

In this section, we summarize concerns of SMEs that arise in the development phase.

TABLE 3: SECDEVOPS CONCERNS SME'S CODE PHASE AS INDICATED BY PARTICIPANTS OF SCRATCH

Aspect of concern	SotA of topic of concern for SMEs	Why is this of particular concern to SMEs?
Lack of specific security training	Specific tools for secure software development are available but not part of general practice of coding for SMEs.	Most developers lack security training making it harder to guarantee the code is secure. This is worst for SMEs that might not have a security team or resources to provide this training. In an SME, architecture cannot be totally controlled top-down, and vulnerabilities and technical debt are built by unskilled workers, making security a very expensive aspect in terms of hiring.
Vulnerabilities from dependences	SMEs use software that has known vulnerabilities. Lists of these exist but it is up to the SME to establish a process to deal with them.	Third party dependencies help developers to speed up the product release but also add more security risks. Keeping them up to date is even harder for SMEs with smaller development teams and less resources.
Manual code review	Manual code review is expensive. To automate code reviews, SAST-tools can be employed.	SAST help an SME performing code reviews more cost-efficiently. However, SAST tools are expensive and interpreting analysis results (findings) is tedious and requires expert knowledge.
Static code analytics interpretation	In static code analysis, we analyze the code for vulnerabilities without executing it. There are a number of tools in the SotA for this.	The analysis of the code is executed by a tool but the analysis of the results is complex (e.g., figuring out false positives). This is why, an expert should analyze the results.
Usage of secrets in code is cumbersome with some tools	Handling secrets in application code requires in depth review of the generated code because optimizers may remove security related code.	Overwriting secret Key material is a good practice for security. Garbage collected languages may not even support direct overwriting of key material. Optimizing compilers for languages like C and C++ will remove, memory writing/clearing operations if they figure out

[5] Modern languages like Rust or Nim (compiles to c) will guarantee the absence of buffer overflows, and even general memory safety on compilation. Even for multithreaded applications. See: https://www.rust-lang.org and https://nim-lang.org

Aspect of concern	SotA of topic of concern for SMEs	Why is this of particular concern to SMEs?
		that the wiped area is not read, or the result of a reading is not used. Maybe the presence of secret erasing code could be validated using a tool.

From the table, we see that SecDevOps like code reviews, dependency tracking, and security training address the major concerns of SMEs. But we also see that especially for SMEs, it is difficult to adopt these approaches. The major obstacles are the high costs for tools, the additional workload for managing the processes, and a lack of expertise at the developers.

Specifics of SCRATCh application domains

In this section, we motivate SecDevOps in the development phase by looking at use cases from Smart Grids, Police, and Smart Retail.

TABLE 4: SECDEVOPS CONCERNS CODE PHASE AS DISCOVERED IN THE USE CASES OF SCRATCH

Use Case	SotA of topic of concern for the given UC in development phase.	Why is this of particular concern to the UC?
Police	Secure development reports to be interpreted by Police staff: details need to be present for audits, etc. but some summary is required for non-technical user (police agent, police chief, policy maker)	The security of the development process is very important for the different Police staff sides but details are often misunderstood if too technical for their consumption.
Connected Retail	Up to now, the external focus was on the security at the plan and deploy phase. There were company internal measures to guarantee security at the code phase, such as trainings for secure coding.	New security standards like Payment Card Industry Software Security Standards on development and lifecycle management of software put more focus on the code phase. Furthermore customers have explicit requirements on the development process itself. This requires support by tools to measure and document the security of the development process.
Smart Grids	Security from design phase is not always properly reflected on the developed system	The Smart Grid projects for the development and implementation of a software system are usually big. The planned security measures during the design phase may not be perfectly followed by the developers during the development phase. Methods and techniques to ensure that the security is developed as designed are needed.

We see that all three domains have to consider security in the coding phase. We identified that SecDevOps has the potential to leverage security in these domains. But the application domains also show that there is a gap between SecDevOps SOTA in theory and SOTA in practice. One of the obstacles that we identified is gap between the technical expertise of the customer and the IoT provider. We also recognized that the adoption of new paradigms like SecDevOps in historically grown domains requires a dedicated strategy.

Smart Grids: The Smart Grid is a critical infrastructure that requires solutions to avoid any potential security breach. Specific solutions for DevOps activities such as code building, deployment and release

management targeting communications are needed in this domain. A major challenge in this domain is to transfer security from the plan phase to the code phase and ensure that the implementation follows the specification from the plan phase.

Police: In building applications for the Police, security has to be even more present because data managed by the application can lead not only to admittedly important concerns such as privacy or financial losses, but can be even critical to the health and safety of agents on the field or citizens under their monitoring.

Thus, the development of secure applications must be enforced and the extent of such enforcements (e.g., tools used, tests conducted) should be carefully documented. This should be kept as a register yet not publicly — this can lead to hackers having strings to pull in their actions.

We also have to keep in mind the particular expertise level of the target demographic. Police officers are not engineers or require by the nature of their job to be well-versed in computer security topics. Thus, there is a particular need for any information to be communicated to them to be done in a manner which is easily understandable and concise, as many of these applications are used in stressful environments such as demonstrations or large crowds. Other details need to be reported as well to other agents in the police environment such as chiefs and policy makers. These can be more technical but need to be maintained confidential within the organization as well.

Smart Retail: Several retail products are security critical, like pin entry devices or point of sales terminals with cash management. Some of the functionality has to adhere to security regulations, e.g., from the payment card industry. To reduce delivery time and at the same time, improve the security quality of the code, a high level of automation has to be applied at the code phase. The retail industry and retail IoT development has a long history. Hence, one of the challenges is the integration of SecDevOps into established processes. Furthermore, the awareness of customers has to be increased to accept the adoption of existing processes.

4.3 Build and Test Phase

Heterogeneous suppliers have different processes and need to be integrated into one environment, components of suppliers.

Making sure that no security vulnerabilities (with new code and libraries) are being introduced after any development cycle is very important and so this must be included in the CI pipeline.

Security Assessment Trigger

During development – when a certain milestone is met – there should be a trigger to perform a security assessment as a means to validate that the application/product/... is developed according the security requirements elicited

1. The actual test should not interfere with the vision of DevOps. As such, it should not be seen as an assessment of which the result dictates whether or not a product can be released (no gate).
2. The focus of the test is to validate on an E2E environment that closely resembles the end-result.

To measure the quality of the code we introduce code and dependency analysis. Based on this metrics we define gates for the continuous delivery pipeline to allow or prevent the delivery of a release. Methods that we consider are: scanning for known vulnerabilities, static code analysis, and dynamic

code analysis. During scanning for known vulnerabilities, we check if our code includes dependencies to third party libraries that have reported vulnerabilities. In static code analysis, we analyse the code for vulnerabilities without executing it. In dynamic code analysis, we execute the code, e.g., as part of existing unit test or dedicated security tests to detect additional vulnerabilities that are not captured by a static analysis.

Build

This phase includes cross compile, CVE Scanning on top of standard build processes. A particular concern in this phase is that libraries used by the applications should be regularly checked for the latest updates. This reduces the risk of introducing security holes through libraries. This process can be automatically performed by the build system.

Test

In the test phase of the DevOps cycle, the functional tests are performed. For security, additional types of tests exist that are intended to find vulnerabilities in the deliverable. Different from searching for CVEs, these test search for new vulnerabilities introduced in the coding phase. Different from SAST, these tests dynamically execute parts of the code. Most DASTs are black box tests, i.e., they do not inspect the internal state of the subject. Some tests, like fuzzing tests, also inspect the internal state.

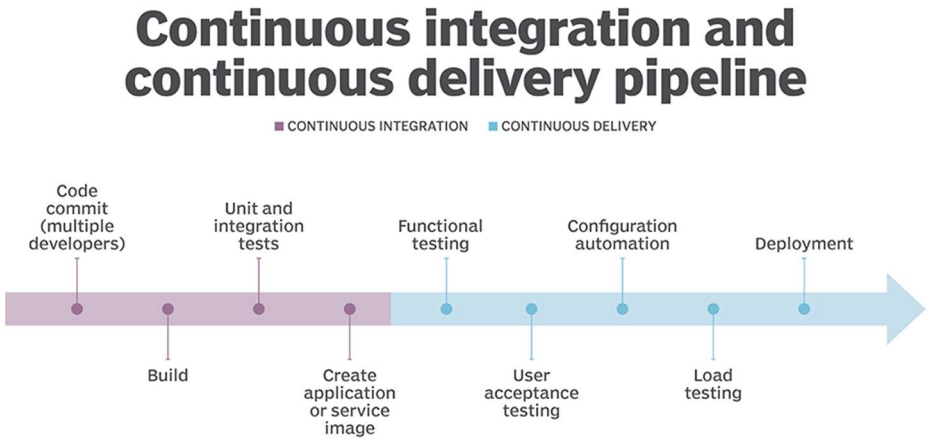

FIGURE 3: CI AND CD PIPELINE
(SOURCE HTTPS://SEARCHSOFTWAREQUALITY.TECHTARGET.COM/DEFINITION/CONTINUOUS-INTEGRATION)

The integration mostly concerns the testing of the product. Before releasing new software, it should be clear that it does not interfere or incapacitates the Identity-Life-Cycle management. So, everything from provisioning, authentication, authorization, credential refreshing and deprovisioning should be tested.

Components which the IoT device relies on for operation need to be tested with special care as they may leave the device bricked and incapable of recovering.

Components which are server-side based may be tested at a different stage, as long as they do not pose a security threat to the device.

Penetration testing

During a penetration test (or pentest for short), an applications or product's overall security posture is measured against a set of verification requirements through simulating cyber-attacks. The result of a

pentest are verified security vulnerabilities that could impact the application's or product's overall security posture in terms of confidentiality, integrity or availability.

Historically, pentests are performed prior to milestone releases. The pentest is then seen as a final validation step on and end-to-end setup and has as main goal to catch potentially overlooked vulnerabilities. If critical risk vulnerabilities are identified, the release would be postponed. The tests performed during a pentest, are often performed manually by a pentester and with the support of tools. These tools can take many forms, ranging from tools that scan for security vulnerabilities fully automatic, to tools that allow manual interaction with an application component, in a way that it was not intended by the developers. Sometimes, and depending on the tooling available, a pentester might even write its own tools to allow testing of very specific cases.

Within the context of SecDevOps, performing a pentest before deployment would not fit the continuous deployment practice as a pentest is not fully automatic and would therefore not allow fully automatic deployments. Within the context of testing connected products, tools that automatically test security are scarce. This can be attributed to a couple of key properties that differentiate products from more traditional applications such as web or mobile applications:

- Diverse and embedded nature – connected products are often designed to have a single use (smart light bulbs, smoke detectors) and are often developed on hardware and software platforms tailored specifically for that use. As a result, there is not one but a highly diverse set of hardware and software platforms to take into account.
- Use of different physical layer communication technologies: products are connected to networks via a wide range of wireless links, such as Bluetooth Low Energy (BLE), 802.11, GSM/UMTS, LoRaWAN.
- Use of different application layer communication technologies: products tend to make use of machine-to-machine communication technologies such as MQTT and AMQP.
- Large attack surface: connected products are part of a large and complex ecosystem. For example, a home alarm will be receiving input from motion detection, cloud and even mobile applications.

Due to the above-mentioned characteristics, fully relying on automated tools for product testing is not yet feasible today. As such, validating security requirements through manual pentesting still remains a crucial part to guarantee a product's resilience against cyber-attacks.

In order to discuss the state-of-the-art of product pentesting in a SecDevOps development process, it makes sense to discuss the challenges and the trends that we observe within this area:

- First of all, there is a lack of security verification requirements and testing guides for connected products. For example, for pentesting traditional application such as web and mobile applications, OWASP provides security verification standards and testing guides that provide details on generic security verification requirements and how these can be validated through testing. For connected products, consolidated guides such as these are only recently being introduced. To help push the status quo, SCRATCh is therefore co-leading the development of the OWASP Internet of Things Security Verification Standard. A standard which provides security verification requirements for connected products within a complex ecosystem of interconnected things. In next steps, this security verification standard could be used as a basis for a security testing guide, which would describe how actual testing can be performed. A testing guide would thus further increase the overall maturity level of pentesting in the field of connected products.

- Second, there is a lack of tooling to support pentesting activities of connected products. For example, for testing the security of web applications, tools such as Burp Suite and OWASP Zap exist. These tools provide features such as inspection and manipulation of HTTP request sent between client and server and automatic vulnerability scanning through fuzzing HTTP requests. While these tools can also be used if the IoT device makes use of HTTP, unfortunately, for many specific IoT technologies as mentioned above, there is a lack of tooling. Interesting to note is that for many of the traditional pentest tools today more and more being automated as well. For example, Burp Suite Enterprise Edition and OWASP ZAP's Docker containers allow for an easy integration of these tool's automatic vulnerability scanning features in a deployment pipeline.

SecDevOps Build and Test phase SME and Use Case-related concerns

The following table shows the Build and Test phase-related SecDevOps concerns of particular relevance to SMEs.

TABLE 5: SECDEVOPS CONCERNS SME'S BUILD AND TEST PHASE AS INDICATED BY PARTICIPANTS OF SCRATCH

Aspect of concern	SOTA of topic of concern for SMEs	Why is this of particular concern to SMEs?
Build/test tools are not integrated with security tools and this requires manual integration	In the build/test-cycle the unit and integration tests are executed. Here we would focus on tests that verify security aspects. These tests should be fully automatic in order not to miss any of the problems detected in the main phase (early detection). The tools for this phase are various and depend on the environment where the development takes place	Even some of the aspects in the build/test-cycle can be accomplished by a non-expert, carrying out the entire mechanism is complex if no specialized person with the correct knowledge and skill is present in this step.
Dynamic testing is not supported	Dynamic tests should be performed, e.g., monitoring Internet connections. This must be automatically done by the tools, and this is not in the SotA now.	Monitored data should be filtered automatically (e.g, by writing filters for Wireshark) and presented in a user-friendly way. This helps SMEs save cost and time.
Feedback loops to the earlier phase	Results of dynamic analyses should be fed back into the implementation and build phase of the software development process.	SMEs need support for feedback tools that automatically communicate the findings of dynamic testing back to the developers. This step reduces time and cost effort.
Cost of testing	A thorough testing of a secure solution is very expensive for SMEs, both in terms of staff and tools required.	SMEs can alleviate costs by leveraging free software so some of the proposed solutions should be free. The integration inside a complete toolchain such as SCRATCh also adds value and thus compensates for any additional cost.

The following table shows the Build and Test phase-related SecDevOps concerns of particular relevance to use cases:

TABLE 6: SECDEVOPS CONCERNS BUILD AND TEST PHASE AS DISCOVERED IN THE USE CASES OF SCRATCH

Use Case	SOTA of topic of concern for the given UC in build/test phases.	Why is this of particular concern to the UC?
All	In dynamic code analysis, we execute the code, e.g., as part of existing unit test or	A unit test execution can detect errors that are shown by the used tool. This

Use Case	SOTA of topic of concern for the given UC in build/test phases.	Why is this of particular concern to the UC?
	dedicated security tests to detect additional vulnerabilities that are not captured by a static analysis.	detection is automatic and no expert is needed.
Retail	Security testing is done before major releases and requires manual work. Some tests depend on the target infrastructure.	For continuous testing, more test automation is required. This is especially difficult for IoT devices with many hardware dependencies.
Police	Penetration testing is extremely critical in Police environments.	Police systems can be connected to very sensitive data of the citizens (e.g., traffic and personal IDs). Thus the access to these resources should be kept secure and all traces of penetration testing should be kept as evidence.

4.4 Release and Deploy Phase

Deployment is a SecDevOps phase that is perhaps one of the most obviously vulnerable and ripe for improvements. When software is deployed to external devices, such communications may be obstructed, read or even tampered with. If an attacker controls deployment, they may be able to update an IoT end device to exhibit any desired behaviour, rendering it fully compromised. Attacks within the Deployment phase are real and happening in the world, and worth defending against. Furthermore, deployment is often limited in capability. Over-The-Air updates are a luxury depending on the context, never mind the ability to update multiple end devices at once.

Types of attack that concern the deployment phase form a wide spectrum. Typical categories of threats here include the reading of status messages or even update payloads, in order to identify vulnerabilities or steal sensitive data; the obstruction of updates in order to eventually render a device vulnerable or broken; denial of service through invalid and/or excessive offering of updates; the abuse of the update mechanism to take control of a device.

Over time, many surprising and specific forms of attack within the deployment phase have been identified. A prominent example is rollback attacks, in which an attacker re-deploys an old update that has become outdated and started to exhibit known vulnerabilities. This is typically prevented by including monotonic sequence numbers with updates, to be able to tell which is more recent. The SUIT documentation gives a good overview of specific attacks as a part of its threat analysis.

Due to the breadth and depth of security threats present within the deployment phase, as well as the inherent clumsiness of having to update hardware-limited end devices across often-limited connections, this is a particularly challenging phase for SMEs. On the one hand, without specially developed in-house deployment systems, deployment taxes manpower and management. On the other, without managing security experts, it is very difficult to maintain a comprehensive understanding of the threats and mitigations.

More specifically, it may be that some or all provisioning happens during the deployment phase, as it may play a part in updates or involve required actions to enable update deployment each time. This,

in turn, may necessitate automated provisioning tests that run during or after deployment. This topic is addressed in e.g. TUF[6], UPTANE[7], SUIT DRAFT[8].

Continuous Delivery, the ideal of automatically deploying committed changes to the production environment, may be impossible in many cases of SME IoT projects. The limitations surrounding the end devices and connections therewith make it very difficult to fully automate testing and deployment.

Notably, many modern deployment schemes involve one or multiple "edge nodes", devices which exist as intermediaries between servers and end devices. Such edge devices may be tasked with downloading updates from a server, verifying them, and securely deploying them to local end nodes. Typically, edge devices are network gateways that act as a bridge between internet communications and local communications.

As it has been for a long time, cryptography is important in this and many situations. Deployment inevitably includes communication between devices, often even through the internet, and this communication must be protected from spying, denial and tampering attacks. Communication cybersecurity is a large topic on its own, and ways of securing communication can become outdated if abandoned.

Key management is also a critical topic concerning deployment. For selecting releases, preparing them and deploying them, keys may be used to ensure authenticity. This means attackers must be kept from obtaining certain keys, or certain combinations of keys, at all costs.

One recent development in deployment security is the increasing prominence and standardisation of firmware manifests. These bundle metadata about updates in set layouts, and are signed and delivered securely, to enable the end device to verify the authenticity and integrity of the update as well as know how to correctly parse and implement it.

Another important mitigation of update tampering is secure boot, and therefore, the presence of a hardware root of trust. Such an immutable base element may be able to detect and block use of unwanted foreign code or data.

As seen in one of SCRATCh's use cases, the retail use case, SecDevOps problems concerning IoT and deployment are made more complex by the integration of different IoT end devices from different vendors, with different connections and protocols, and so on. This makes it difficult to manage deployment for all end devices centrally. Typically, the integrator of the retail store integrates IoT components of different suppliers. For example, a pin entry device of a supplier is integrated into a point of sales terminal of the integrator. With respect to deployment, various supplier release software for their IoT components . This software has to be deployed into the integrated system. At the same time, compatibility between software version has to be maintained by the integrator. Therefore, the integrator has to execute integration tests in an integration environment. If the quality release gate is passed, the composition of different software components is released by the integrator. Before it is deployed into the productive environment of the retail store, the retail IT operator also performs system tests in the retail store staging environment. A deployment system that is secured, e.g., based on signatures, has to support the different roles and software life cycles like component supplier, system integrator, system operator, released for testing, and released for production.

[6] https://theupdateframework.io

[7] https://uptane.github.io/papers/ieee-isto-6100.1.0.0.uptane-standard.html

[8] https://tools.ietf.org/html/draft-ietf-suit-architecture-16

A further complication arises from the fact that some components are subject to certification. Before such a component is released, a certifier evaluates the component and approves or rejects it. The deployment system should also prevent roll out of components that did not pass certification.

Finally, consider deployment for environments with specific constraints on the connectivity. For example, the integrator often cannot connect directly to the IoT components in the retail store or the connectivity is shut down during maintenance.

Next is included a table of specific intersections between the concerns of SMEs and SCRATCh Use Cases and the Deployment phase of DevOps for IoT. The table includes a summary of each point of concern, and an elaboration on its specific relation to the concerned parties.

TABLE 7: SECDEVOPS CONCERNS SME RELEASE AND DEPLOY PHASE AS INDICATED BY PARTICIPANTS OF SCRATCH

Aspect	SotA of topic of concern for SMEs	Why is this of particular concern to SMEs?
Active Security	Secure firmware updates can be realized using existing 'building blocks' (tools and open standards). Tools already exist and have been leveraged and assessed e.g.: [IEEE IoT 2019]. However, to pick the suitable ones for a specific use case and implementing it for consistent use can be a daunting task.	Keeping a system safe means that a secure mechanism to update a device in the field should exist. Implementing a update mechanism and maintaining it is necessary for deploy new IoT devices in the market.
Ease of Use	Manpower/specialisation/peers. Tools can prevent misconfiguration or give sensible errors otherwise. But surveying the entire update mechanism is difficult if no specialized knowledge and skill is available. Is everything covered, no gaps? (How to check yourself?) Blind spots etc.).	Insight in the update method of choice is essential, SME's that have no access to personal trained in specialized security knowledge. Benefit by a tool that simplifies this task and aids the application of an update mechanism
Active Security	Secure configuration updates need to be implemented in addition to firmware ones.	In some SME applications, even if the firmware of the system remains unchanged, configuration updates may need to be performed frequently. This should be done OTA and not require a maintenance staff present at the deployment.

TABLE 8: SECDEVOPS CONCERNS RELEASE AND DEPLOY PHASE AS DISCOVERED IN THE USE CASES OF SCRATCH

Use Case	SotA of topic of concern for the given UC in the release/deploy phase.	Why is this of particular concern to the UC?
Police	Reports of deployment need to be easily understandable.	In Police UC the recipient of the deployment reports is usually either police agents or funding bodies for the Police (e.g., City Halls) that often don't have the training to understand technical detail. Anyway, critical KPIs of the deployment should be delivered to

Use Case	SotA of topic of concern for the given UC in the release/deploy phase.	Why is this of particular concern to the UC?
		them (e.g., degree of updated versions in the deployed devices).
Retail	Continuous deployment into customer infrastructure.	The infrastructure of the customer like the retail store is loosely or not coupled with the network of the component provider. Hence, it is difficult to implement continuous deployment.

4.5 Operate, Monitor, End of Life Phase

Operation and monitoring are closely related in terms of security when considering IoT networks. Operation includes setting up, running and adjusting the system, while monitoring describes the collection of feedback to alter parameters. Both require underlying security features that enable secure control and prevent unauthorised access and tampering. A summary of particular SecDevOps concerns with relevance to SMEs is found in Table 9, SecDevOps-related concerns with relevance for the use cases addressed in project SCRATCh are summarised in Table 10. The topics listed there are discussed in detail in this section.

Dynamic application security testing (DAST) enables identification of security vulnerabilities at runtime (in contrast to static application security testing (SAST), which analyses the source code, e.g., at compile and build-time). DAST include different types of analysis tools, e.g., Web-application checkers (e.g., OWASP® Zed Attack Proxy (ZAP)), fuzzers for specific IoT protocols (e.g., MQTT, Bosch Gateway Protocol, REST interfaces of IoT devices, BLE, Z-Wave, and ZigBee), and network testing tools.

Network testing tools help one monitor the communications in IoT-based networks. For example, one can use the Wireshark tool[9] capture traffic and then automatically analyse the file. Within in the frameworks of the SCRATCh project, the OTAlyzer tool has been developed, which can be used to analyse IoT communications traffic. Other networking test tools check TLS communications, e.g. by trying to carry out man-in-the-middle (MitM) attacks with tools like Burp or mitmproxy. This approach, in particular, is effective in IoT environments as many TLS-stacks of IoT devices show weaknesses due to the fact that dynamic IP addresses are used - dynamic IP addresses do not fit well to X.509 certificates, which are used in TLS implementations [8].

IoT security in the DevOps context, in the operate and, more prominent, monitor phases, is similar to classic IT security practices: Access control and traffic as well as behaviour monitoring. Firewalls with access control, authentication-based access for users and operators, as well as usage of tools, such as Wireshark and SIEMs like Splunk provide a level of security. However, IoT networks are prone to heterogeneity. Behavioural analysis with anomaly detection can provide insights that classic tools cannot.

Apart from the above-mentioned tools that are developed in the context of project SCRATCh, there is a variety of established tooling used to secure networks. IoT-networks are a special type of network, consisting not only of general-purpose computers, servers and databases, and mobile devices, but also of embedded, low-energy and application-specific devices. These devices provide significantly less interfacing for users. Elrawy et al. provide an extensive overview of scientific approaches to IDSs

[9] https://www.wireshark.org/

created for IoT networks [9]. In the context of SCRATCh, an anomaly detection-based approach is chosen that employs features unique to IoT networks. Traffic patterns of M2M communication contain patterns that can be employed for the detection of anomalous behaviour. Furthermore, the topology of communication is expected to not change over time, except for devices being added to or removed from the network. Additionally, a semantic understanding of behaviour on IoT devices is trained, so that the algorithm can detect anomalous behaviour caused by the device. The combination of data sources and integration with a rule-based approach to commonly known attacks promise an excellent coverage of attacks and detection of any unwanted behaviour.

An alternative approach to security monitoring is the introduction of deception technologies into the IoT network. Thinkst Canary provides such a service called canaries, where a device is placed in the network, mimics a productive system and then monitors any access to it [10]. As no productive system is connecting to the canary, any access is a true positive example of an attacker. This concept will be extended in the course of SCRATCh to extend this principle to other implementations of deception, e.g. TCP header fields or resources in a deceptive web resource. If resources that are solely deceptive are offered by a proxy in addition to the actual resources of a web server, any access to the deceptive resources indicates someone trying to access information not intended for her or him. This, again, provides a method with zero (or very few) false negative results and can increase the security of a system with minor cost. Even if an attacker is aware of deception being used, the attacker cannot know, which resources are productive and which are deceptive, so one misjudgement alerts the IDS about the presence of an attacker.

A further important step in SecDevOps contexts is how to implement a feedback loop to the development step. Ideally, here the results of the different tools must be communicated back to the development step automatically (e.g., to a bug tracker). Furthermore, the various DAST tools must be integrated into a common analysis infrastructure to enhance automation, which is crucial to SecDevOps environments, specifically for SMEs.

From the Use Cases' perspective, we can find some considerations on this stage for several of the project Use Cases. For Police, there are specific concerns on two topics, one due to the security of the operations during their execution and the other more related to human factors. The security relation is clear: Police data is very tempting for malicious users, as police agents' devices might be transmitting critical data for the safety of these same malicious users themselves. One clear scenario would be a deployment of video surveillance in search of a target who in turn is able to eavesdrop on the communications of the surveillance system itself. This could put the whole operation in jeopardy and so it is a very directly valuable asset for the interceptor, even more so than in other scenarios. Thus, for this Use Case SCRATCh needs to deliver clear indications that eavesdropping is a key concern. This can be the driver for very focused marketing of the tools for these collectives and also the very defence against eavesdropping (e.g., cryptography) should be very tangible in the operation phase tools.

The other concern is related to the human factor, as the users of these systems in this Use Case are never extremely technically oriented. Even in modern police forces, the personnel in charge of the operations is only moderately aware of the technical details, and all possible help in translating the results of tools into clear, actionable data is very important, Thus, SCRATCh operation phase tools need to focus on delivering these clear and actionable messages to its end-users.

From the retail use case, we see that the feedback from the network infrastructure to the device manufacturer is often restricted by network isolation. By default, the network of the operator does not permit information to be transferred back to the device vendor. To access security feedback information from the field, a secure feedback channel has to be integrated into the operator's network. Furthermore, monitoring and deception tools are executed in the environment of the operator. Hence,

the integration is customer specific and requires cooperation with the operator. Therefore, deploying SCRATCh monitoring and deception tools is especially challenging because it is not under complete control of the device manufacturer.

TABLE 9: SECDEVOPS CONCERNS SME OPERATE, MONITOR, END OF LIFE PHASE AS INDICATED BY PARTICIPANTS OF SCRATCH

Aspect	SotA of topic of concern for SMEs	Why is this of particular concern to SMEs?
Monitoring	Operations Monitoring	Niche product operated by AnyWi, involves other SME, cross product monitoring.
	Dependency Tracking (Dependency monitoring)	The ability to check and list all dependencies in an IoT product is necessary to develop a pro active role to security updates.
	Attack sensors for SME networks	Several vendors offer plug in solutions that are supposed to detect intrusions in the network. Development and refinement of such solutions can enhance the security level of SMEs in an automated fashion.

TABLE 10: SECDEVOPS CONCERNS FOR THE OPERATE, MONITOR, END OF LIFE PHASE AS DISCOVERED IN THE USE CASES OF SCRATCH

Use Case	SotA of topic of concern for the given UC in the operate, monitor and end of life phase.	Why is this of particular concern to the UC?
Police	Operations monitoring requires a link to non-tech oriented users.	In Police UC the intended user of the system (a Police department) often doesn't have very tech-savvy users that can understand the details, yet some aspects (breaches in security) are critical to them. Tools should be provided with both perspectives in mind.
	Eavesdropping of transmissions to the Police HQ is critical.	Police operations are often the target of malicious eavesdroppers. All data that is transmitted can threaten the privacy of personal data and even the integrity of agents if .intercepted
Retail	Feedback of security related information.	Today, IoT devices in the retail domain already generate status information that is stored locally at the device, e.g., for failure analysis. This data is usually not accessible from an external network by the device vendor. To continuously improve the security of products with SCRATCh monitoring tools, the component vendor requires feedback from the operator's network. The secure integration of the feedback mechanisms into the customer IT infrastructure, especially into the network is required.

5 Conclusions

This Document aimed to cover the current available knowledge on IoT and SecDevOps. During the creation of this document, it became clear that whole field of DevOps and specific SecDevOps is immense and divers, not all publications are about IoT and a lot of presentations are about Change management in relation to DevOps. We think despite the abundance of info available on the internet this document gives a good summary of the State of the Art in the context of the SCRATCh project.

6 References

[1] Javed, B., Iqbal, M. W., & Abbas, H. (2017, May). Internet of things (IoT) design considerations for developers and manufacturers. In 2017 IEEE International Conference on Communications Workshops (ICC Workshops) (pp. 834-839). IEEE

[2] IETF, RFC 8886 Secure Device Install

[3] Secure IoT Bootstrapping: A Survey draft-sarikaya-t2trg-sbootstrapping-06

[4] G. Sindre, A. L. Opdahl. 2005. Eliciting security requirements with misuse cases. Requirements Engineering 10, 1 (January 2005), 34-44

[5] A. Shostack. 2014. Threat Modeling: Designing for Security (1st. ed.). Wiley Publishing

[6] J. Williams. 2020. OWASP Risk Rating Methodology. Accessible under: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[7] D. Dhillon, "Developer-Driven Threat Modeling: Lessons Learned in the Trenches," in IEEE Security & Privacy, vol. 9, no. 4, pp. 41-47, July-Aug. 2011, doi: 10.1109/MSP.2011.47

[8] T. Osmers. 2018. security analysis of TLS client implementations of Android applications regarding the communications with IoT devices in local networks, Bachelor Thesis. University of Bremen, Germany

[9] Mohamed Faisal Elrawy, Ali Ismail Awa, Hesham F. A. Hamed. Intrusion detection systems for IoT-based smart environments: a survey. In: Journal of Cloud Computing vol. 7, no. 21, Springer (2018)

[10] https://canary.tools/, last checked 2021-02-09