SCRATCh

SECURE AND AGILE CONNECTED THINGS

Deliverable 1.2

D1.2 Toolset Framework

Work Package:	WP1
Affected milestone:	MS1
Partners involved:	Irdeto BV AnyWi BV Almende BV NVISIO BV Otaris GmbH consider it GmbH NXP Semiconductors Germany GmbH
Date:	16/10/2020
Deliverable version:	v1.91
Author(s):	See below
Responsible Contact:	Till S. Witt NXP Semiconductors Germany GmbH

Version history

Date	Version	Author	Comment
16/09/2020	1.8	Franklin Selgert Werner Strydom	Initial version, based on the presentation Toolset Framework v1.7
16/10/2020	1.91	Werner Strydom Karsten Sohr Cédric Bassam	Updated with current view of the Toolset Framework. Added overview of tools under development.

Table of contents

1 Introduction

1.1 Purpose

This technical report captures the outcome of task 1.2 and is regularly updated with new findings as the SCRATCh project evolves. The goal of task 1.2 is to define a reference toolkit architecture that can be used to facilitate effective SecDevOps for IoT software development projects.

Task 1.2 takes the requirements identified in task 1.1 into consideration, with specific attention to requirements that are enabled by/have dependencies on security-by-design, secure development, continuous integration/continuous delivery, secure deployment, and secure operations.

This reference toolkit architecture then serves as input *in specific* to the work performed in work package 2 on interoperable tools, and *in general* to any IoT project that wishes to adopt or accelerate an agile Secure Development-Operations (SecDevOps) process.

This document is structured in two parts. The first provides an overview of the generic toolset framework defined in task 1.2 and contains a description of each identified tool. The second part provides more details on specific tools that are being developed by consortium partners in the context of the SCRATCh project, and which fit into this framework.

1.2 Scope

The goal of SecDevOps is to facilitate effective cross-functional collaboration between software development teams and IT operations teams, while maintaining a focus on cybersecurity. SecDevOps is often used in conjunction with *Agile* software development methodologies, from which the *Continuous Integration* and *Continuous Delivery* (CI/CD) concepts originate.

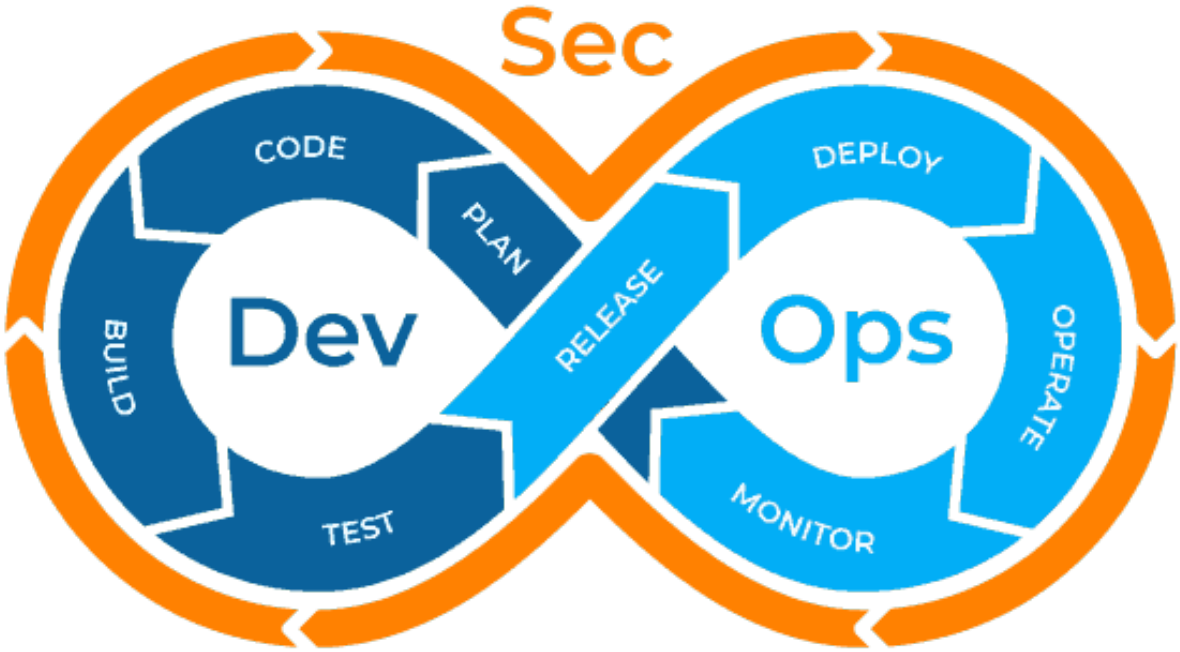

FIGURE 1: SECDEVOPS

The commonly agreed structure for a SecDevOps life cycle consists of 3 high-level disciplines that are broken down into 8 phases as illustrated in Figure 1 above. In order to allow the SCRATCh tool framework to specify a more comprehensive set of tools, it was decided to break the PLAN phase down into two phases (both of which contributes critical inputs to planning), namely REQUIREMENTS and DESIGN. The motivation for doing so is to allow the framework to highlight two related tool categories that are critically important to ensure E2E security within SecDevOps.

The SCRATCh toolset framework therefore considers the following SecDevOps phases:

1. **Requirements Phase** – During this phase the user requirements are documented and analysed. Once the requirements are agreed, they are prioritised, 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 roadmap).

3. **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.

4. **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.

5. **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.

6. **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.

7. **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.

8. **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.

9. **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.

A practically endless array of tools is used during the SecDevOps cycle to automate tasks and make life easier for those involved in the process (see figure 2). The tools generally include commercial tools,

open source tools, and proprietary developed tools. As a consequence, setting up a new end-to-end SecDevOps practice can be overwhelming, especially for smaller organisations.

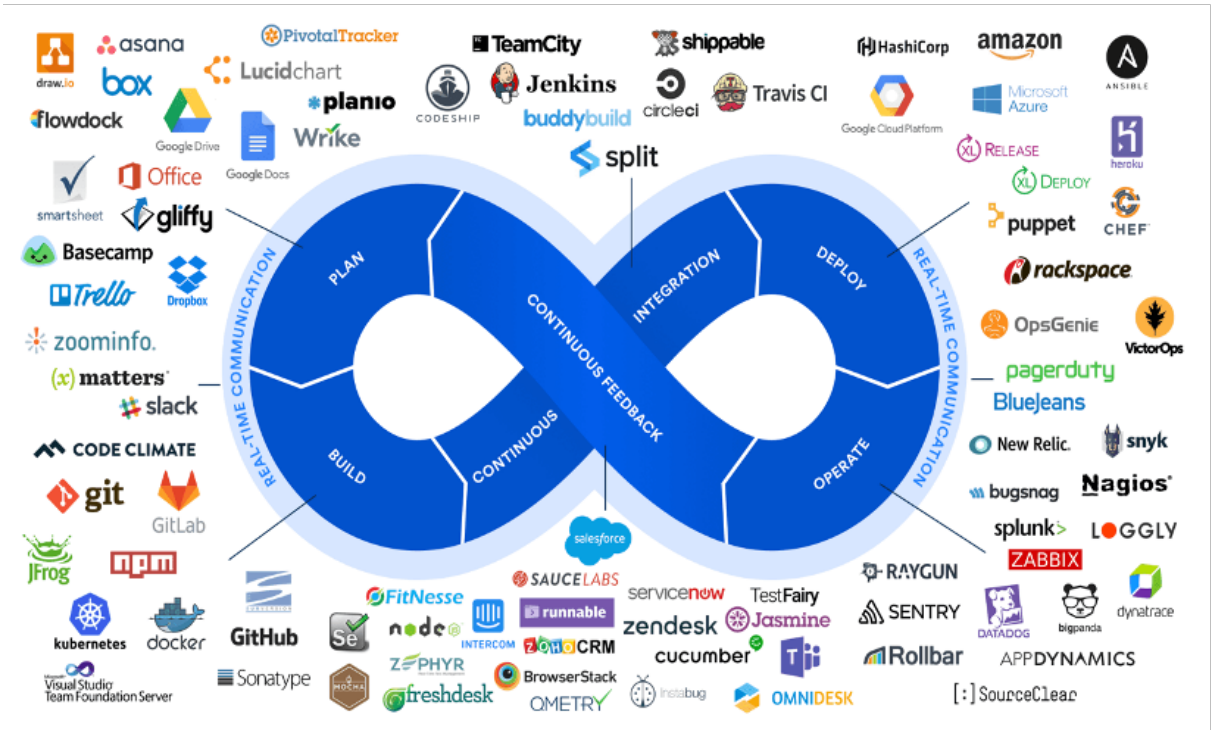

FIGURE 2: TYPICAL SECDEVOPS TOOLS

2 The SCRATCh Toolset Framework

2.1 Purpose of the Toolset Framework

It has become common practice within the traditional Information Technology (IT) domain to automatically update software (computer software applications, web applications, mobile apps) with high frequency to address defects and to add new features. This practice is also well accepted by the end users of these applications. In contrast, this is almost unheard of within the Operational Technology (OT) domain, where firmware and embedded applications for IoT devices may never be updated, or only updated through a convoluted manual process that is inaccessible to most end users.

The goal of the SCRATCh Toolset Framework is to make it easier for software development teams working on OT projects to adopt SecDevOps. The framework is broken down into the 9 phases of SecDevOps as described above and lists the tool categories and best practices that are relevant within each phase. Refer to figure 3 for a visual depiction of the framework.

The remainder of section 2 provides a detailed description of each tool category, along with some examples of commercial and open source tools that fit in this category.

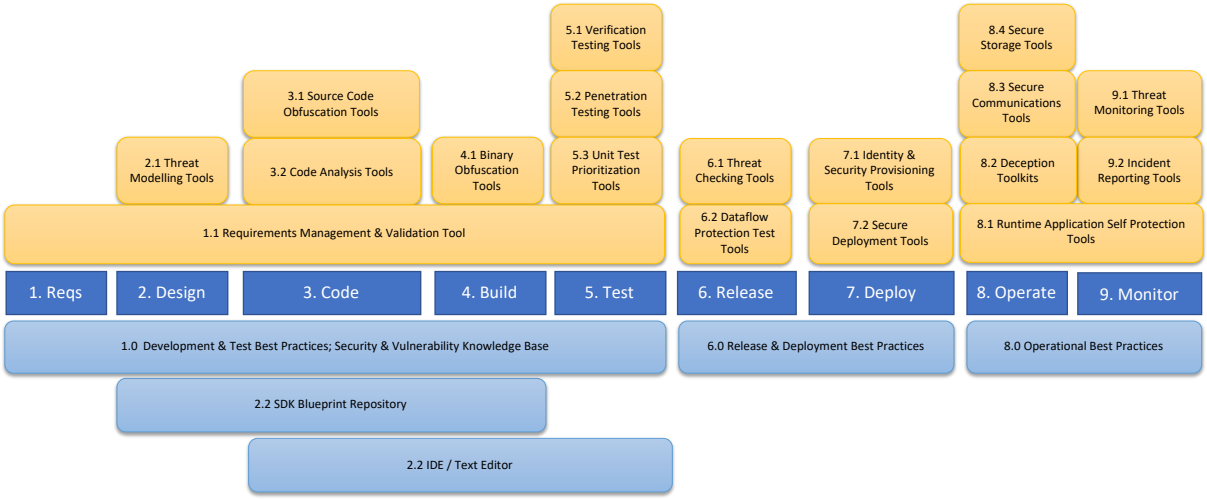

FIGURE 3: THE SCRATCH TOOLSET FRAMEWORK

2.2 Toolset & Best Practice Descriptions

Tool Category	*[1.0] Test & Development Best Practices*
SecDevOps Phases	Requirements, Design, Code, Build, Test
Tool Examples	N/A
Tool Purpose	Provides essential advice and standards for SecDevOps. Includes major established standards.
Inputs Required	N/A
Actions Performed	N/A
Outputs Produced	N/A
Tool Interfaces	N/A
Standards	
Req. Mapping	

Tool Category	*[1.1] Requirements Management & Validation Tools*
SecDevOps Phases	Requirements Design, Code, Test
Tool Examples	Jira (Atlassian), Jama Connect (Jama Software), Rational Doors (IBM), ReQtest
Tool Purpose	Provides full management of the lifecycle of requirements (also called epics, user stories, or features), including definition, modelling of dependencies, visualisation, status tracking during design, coding, & testing, and more.
Inputs Required	User inputs related to requirements
Actions Performed	Evolution of definition of requirements and tracking of various statuses associated with requirements
Outputs Produced	Unambiguous requirements; reports detailing requirements coverage during design, coding, and testing
Tool Interfaces	These tools may have interfaces with the development tools, test subsystem, and documentation tools.
Standards	ANSI/IEEE Guide to Software Requirements STD 830-1984
Req. Mapping	

Tool Category	*[2.1] Threat Modelling Tools*
SecDevOps Phases	Design
Tool Examples	Microsoft Threat Modelling Tool, OWASP Threat Dragon, IriusRisk Threat Modelling Tool, ArchSec (https://archsec.informatik.uni-bremen.de)
Tool Purpose	Threat Modelling (STRIDE) or Architectural Risk Analysis (e.g., as defined by McGraw, https://searchsecurity.techtarget.com/opinion/McGraw-Software-insecurity-and-scaling-architecture-risk-analysis) is an important activity within a Security Development Lifecycle (SDL). Architectural Risk Analysis/Threat Modelling is carried out within the software design phase and attempts to systematically identify architectural security defects. The derived threats are ranked according to their risks in order to later define adequate mitigations. Approaches to risk management include Microsoft DREAD, FAIR, Bug Bars or specific methodologies as defined, for example, by SAP SE.
Inputs Required	Architectural diagrams of the software; high-level security and privacy requirements
Actions Performed	Manual security analysis of the software architecture, threat/risk elicitation and ranking
Outputs Produced	A list with architectural risks and possible mitigations (design-level security requirements)
Tool Interfaces	Results of Threat Modelling can be used as input for dynamic testing (more focused dynamic tests); MS Threat Modelling Tool interfaces with issue trackers
Standards	Common Weakness Enumeration (CWE), ISO Standard 25000, ISO/IEC 25010
Req. Mapping	

Tool Category	*[2.2] SDK Blueprint*
SecDevOps Phases	Design, Code, Build
Tool Examples	SDKs (usually vendor and solution specific) such as NXP MCUXpresso SDK Builder
Tool Purpose	The blueprint of the SDK makes it possible to replicate the SCRATCh SecDevOps environment with the vendor specific tools and (if possible) provide a vendor agnostic solution to incorporate the SCRATCh SecDevOps approach.
Inputs Required	SCRATCh SecDevOps processes and HW/SW selection
Actions Performed	Preparation of SDK to meet the process requirements and get a quick start from requirements through to testing.
Outputs Produced	A testable application either in a simulator or on hardware
Tool Interfaces	Interfaces to process and hardware components
Standards	
Req. Mapping	

Tool Category	*[3.1] Source Code Obfuscation Tools*
SecDevOps Phases	Code
Tool Examples	Cloakware Software Protection (Irdeto), Trusted Software (Irdeto), Jscrambler (Jscrambler), SmartAssembly (Redgate)

Tool Purpose	Provides protection against reverse engineering, tampering, and IP theft of software by modifying the source code (or an intermediate representation thereof) using techniques such as branch protection, control flow flattening, data transformations, variable obfuscation, constant and symbol hiding, function transformations, security in-lining, and white box cryptography. The resulting software, once compiled, is typically hardened against static and dynamic (run-time) analysis. This tool is used as an alternative to 4.1 Binary Protection Tool.
Inputs Required	Source code or intermediate representation of source code, obfuscation settings/parameters
Actions Performed	Obfuscate code and data, inject white box crypto
Outputs Produced	Hardened source code and transformed data, ready for compilation
Tool Interfaces	No hard interfaces; tool is inserted into the build pipeline as an extra step
Standards	ETSI TS 103 718 - External encodings for Advanced Encryption Standard
Req. Mapping	

Tool Category	*[3.2] Code Analysis Tools*
SecDevOps Phases	Code
Tool Examples	Fortify Static Code Analysis Tool (Micro Focus), Checkmarx Static Application Security Testing (Checkmarx), Static Analysis Tool (Veracode), Coverity Static Analysis (Synopsis), AppScan (IBM), SonarQube, Clang Static Analyzer (freely available), TiCS
Tool Purpose	Code analysis tools analyse source code with the help of advanced compiler-construction techniques (e.g., data and control flow analysis) to detect potential low-level security bugs. Typical bugs include buffer/heap/integer overflows, code injection vulnerabilities (e.g., SQL injection, Cross-Site Scripting), cryptographic misuse, simple race conditions. Typically provides TQI, based on consolidation of various software quality metrics like test coverage, abstract interpretation, cyclomatic complexity, compiler warnings etc.
Inputs Required	Source code of the software to be analysed; rules to be checked against the source code (ruleset)
Actions Performed	Automated static analysis of the source code against defined security rules
Outputs Produced	A list with all potential findings (low-level bugs), often a classification as to the severity of the finding
Tool Interfaces	Integration with an IDE sometimes possible, in some cases online service allowing one to upload the source code
Standards	OWASP Top 10, SEI CERT coding guidelines, Common Weakness Enumeration (CWE), ISO Standard 25000, ISO/IEC 25010.
Req. Mapping	

Tool Category	*[3.3] Integrated Development Environment*
SecDevOps Phases	Code
Tool Examples	Visual Studio, IntelliJ IDEA, PyCharm, PhpStorm, Eclipse, WebStorm, Xcode, Syncfusion, NetBeans, Arduino IDE
Tool Purpose	Aid in the construction, formatting and maintenance of large amounts of code within a project. Enable automatic detection of bad coding practices.
Inputs Required	Code

Actions Performed	Auto-completion; detection of errors and weaknesses; simplification and automation of various coding activities
Outputs Produced	Formatted code
Tool Interfaces	Coding language formats; user preferences
Standards	
Req. Mapping	

Tool Category	*[4.1] Binary Obfuscation Tools*
SecDevOps Phases	Build
Tool Examples	Application Protection™ (Arxan), Code Protection™ (WhiteCryption), Denuvo Anti-Tamper (Irdeto)
Tool Purpose	Provides protection against reverse engineering, tampering, and IP theft of software by modifying the binary code using techniques such as branch protection, control flow flattening, data transformations, variable obfuscation, constant and symbol hiding, function transformations, and white box cryptography. The resulting binary is typically hardened against static and dynamic (run-time) analysis. This tool is used as an alternative to 3.1 Source Code Protection Tool.
Inputs Required	Binary code, obfuscation settings/parameters
Actions Performed	Obfuscate binary and data, inject white box crypto
Outputs Produced	Hardened binary code and transformed data
Tool Interfaces	No hard interfaces; tool can be inserted into the build pipeline as an extra step
Standards	ETSI TS 103 718 - External encodings for Advanced Encryption Standard
Req. Mapping	

Tool Category	*[5.1] Verification Testing Tools*
SecDevOps Phases	Test
Tool Examples	IoT security verification standard (NVISO), IoT testing guide (NVISO), Tools to test/read out the configuration of an IoT application (NVISO)
Tool Purpose	Assist security testers in their security testing activities
Inputs Required	An IoT application (code build) together with the physical embedded device. Security assessments can be either: • White box: the security tester is provided with all of the required information to perform the test. For example, accounts, source code, architecture design. • Black box: the security tester is provided only with the physical embedded device running the application.
Actions Performed	Tools are used by the security tester to verify security requirements
Outputs Produced	Pass/Fail
Tool Interfaces	Not relevant to the security verification standard or testing guide. The tools however, will interface with the IoT application through a certain interface (can really be anything depending on the application)
Standards	Security verification standard will be based on the common ground found between many existing IoT security requirement standards. For example, through teaming up with OWASP IoT.
Req. Mapping	

Tool Category	*[5.2] Penetration Testing Tools*
SecDevOps Phases	Test
Tool Examples	- Tools that allow to intercept/alter communication of IoT specific communication protocols. - Tools that allow the extraction of certain configuration properties of certain embedded platforms - Tools that can interface with certain chips
Tool Purpose	Tools will allow a security testers to gain access to the required input required to use the verification testing tools as defined in 5.1
Inputs Required	Embedded application and physical device
Actions Performed	Depends on tool
Outputs Produced	Access to required input for 5.1
Tool Interfaces	N/A
Standards	N/A
Req. Mapping	

Tool Category	*[5.3] Unit Test Prioritization Tools*
SecDevOps Phases	Test
Tool Examples	TSelect, XRay, Jnan
Tool Purpose	Create a market ready test case selection and prioritization tool, which can be delivered as a library, independent tool or a plug-in, to optimize testing of embedded systems in a continuous integration environment.
Inputs Required	Test Suite, Historic data, Requirements, Code Changes (e.g., .xml, .json)
Actions Performed	Test case prioritization according to input data.
Outputs Produced	Prioritized test cases
Tool Interfaces	No hard interfaces. The tool can be inserted as a library, an independent tool or a plug-in in the testing phase of the development cycle.
Standards	N/A
Req. Mapping	

Tool Category	*[6.0] Release & Deployment Best Practices*
SecDevOps Phases	Release, Deployment
Tool Examples	
Tool Purpose	Provides essential advice and standards for secure DevOps. Includes major established standards
Inputs Required	None
Actions Performed	Searching and filtering
Outputs Produced	Important requirements and guidelines for a development process.
Tool Interfaces	N/A
Standards	
Req. Mapping	

Tool Category	*[6.1] Threat Checking Tools*
SecDevOps Phases	Release
Tool Examples	Xray (JFrog), OWASP Dependency-Check (OWASP), CVE-check-tool (Clearlinux)
Tool Purpose	Threat checking tools generally operate on the release artefact repository and perform a composition analysis of the components that make up the

	release of a software application or system. The purpose of this is to identify publicly disclosed vulnerabilities contained within these components.
Inputs Required	Release artefacts, publicly known vulnerabilities (CVE database)
Actions Performed	Analysis of composition of the release
Outputs Produced	Notifications of detected common vulnerabilities
Tool Interfaces	N/A
Standards	
Req. Mapping	

Tool Category	*[6.2] Dataflow Protection Tools*
SecDevOps Phases	Release
Tool Examples	Wireshark, mitmproxy, OTALYZER (extension based on Wireshark and mitmproxy output)
Tool Purpose	Automatically analyse traffic data (e.g., in pcap format) with respect to dataflows that violate privacy and security requirements (pcap input) with the help of a keyword search
Inputs Required	Key word list (e.g., add blocking lists/host files), pcap files, mitmproxy files
Actions Performed	Automatically analyse pcap or mitmproxy files regarding key words to identify privacy violations (e.g., tracking), performing IP address lookups and reverse DNS to identify servers
Outputs Produced	List of findings
Tool Interfaces	pcap, mitmproxy files (traffic captures)
Standards	Pcap format
Req. Mapping	

Tool Category	*[7.1] Identity & Security Provisioning Tools*
SecDevOps Phases	Deploy
Tool Examples	Trust Provisioning Services (NXP), Keys & Credentials Service (Irdeto)
Tool Purpose	Generates and assigns identities and associated security assets to IoT devices in a secure manner. This may include assets such is unique identifiers, X.509 certificates, symmetric keys, passwords, and associated metadata that is required for securely managing a device throughout its entire lifecycle. The generation of security sensitive items is done in a trusted environment, typically using a hardware security module (HSM). The provisioning of these data can be done into silicon chips during a so-called personalisation process, in a manufacturing facility where the device is assembled, or online when the device is first connected to a network.
Inputs Required	Number of identities to generate, optionally with unique identifiers
Actions Performed	Secure generation of certificates, keys, passwords, etc and association of these assets with unique identities. Provisioning of these identities into chips or devices.
Outputs Produced	Certificates, keys, passwords, etc
Tool Interfaces	Configuration (types of assets required), number of identities required, optionally unique identifiers
Standards	
Req. Mapping	

Tool Category	*[7.2] Secure Deployment Tools*
SecDevOps Phases	Deploy
Tool Examples	Crownstone Update Framework (Almende)
Tool Purpose	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.
Inputs Required	Function calls & firmware payload
Actions Performed	Secure transfer and install of firmware payload
Outputs Produced	None
Tool Interfaces	Direct interfacing with firmware code; format and size requirements for payload
Standards	
Req. Mapping	

Tool Category	*[8.0] Operational Best Practices*
SecDevOps Phases	Operate, Monitor
Tool Examples	
Tool Purpose	Provides essential advice and standards for secure DevOps. Includes major established standards.
Inputs Required	None
Actions Performed	
Outputs Produced	Important requirements and guidelines for a development process.
Tool Interfaces	Only human interfacing
Standards	
Req. Mapping	

Tool Category	*[8.1] Runtime Application Self Protection Tools*
SecDevOps Phases	Code, Build, Operate, Monitor
Tool Examples	Rackspace, Travis CI, Puppet, Chef
Tool Purpose	Detects and prevents attacks on applications in real time by analysing both the behaviour of the applications and the context of that behaviour. RASP allows attacks to be identified and mitigated without human intervention. RASP is integrated into the server-side application and typically leaves the client-side application untouched. RASP systems can usually be configured to run in diagnostic mode (for monitoring) or protection mode (for enforcement).
Inputs Required	Server-side code, and binaries at link time (Code, Build) Running system (Operate, Monitor)
Actions Performed	Insert protection and monitoring features in runtime binary (Code, Build) Provides threat monitoring information (Monitor) Prevents attacks on running systems (Operate)
Outputs Produced	Application with runtime protection System with embedded monitoring capability (able to produce relevant telemetry) and the ability to prevent certain types of attacks
Tool Interfaces	No hard interfaces; tool can be inserted into the build pipeline as an extra step
Standards	
Req. Mapping	

Tool Category	*[8.2] Deception Toolkits*
SecDevOps Phases	OSfuscate
Tool Examples	The purpose of this tool is to impede the reconnaissance phase of an attacker. According to the Lockheed Martin Cyber Kill Chain, this phase is the first in a series of hostile activities. A directed manipulation of the hostile reconnaissance output therefore affects the entire process of hostile activity. Ideally, deception allows immediate impact on the prevention of an attack.
Tool Purpose	
Inputs Required	
Actions Performed	
Outputs Produced	
Tool Interfaces	
Standards	
Req. Mapping	

Tool Category	*[8.3] Secure Communications Tools*
SecDevOps Phases	Operate
Tool Examples	Any VPN Client
Tool Purpose	An easily introduced layer of security that makes sure all public network traffic is protected.
Inputs Required	(Unprotected) network traffic.
Actions Performed	Depends on the tool, e.g. end-to-end encryption
Outputs Produced	Protected network traffic.
Tool Interfaces	No hard interfaces; tool can be inserted into the build pipeline as an extra step
Standards	
Req. Mapping	

Tool Category	*[8.4] Secure Storage Tools*
SecDevOps Phases	Operate
Tool Examples	Various software toolkits that allow for data to be stored in encrypted form Secured storage facilitated by hardware
Tool Purpose	Ensure that security related data (e.g. keys, passwords) and privacy sensitive data is stored in a manner that will protect it.
Inputs Required	Either protected or unprotected data, depending on operation
Actions Performed	Encryption / Decryption operations
Outputs Produced	Either unprotected or protected data, depending on operation
Tool Interfaces	
Standards	
Req. Mapping	

Tool Category	*[9.1] Threat Monitoring Tools*
SecDevOps Phases	Monitoring
Tool Examples	Darktrace, FireEye (Cisco), Vectra AI (Vectra), QRadar (IBM)
Tool Purpose	Threat monitoring tools typically monitor network traffic between endpoints on an IT or OT network in order to detect cyber-threats and latent vulnerabilities. These systems are configured to understand - or in some cases use AI to learn - what 'normal' behaviour is, and then flag any deviations from this norm (anomalies). This can generally identify a wide range of threats, from malware to network intrusion.
Inputs Required	Network traffic, configured or learned understanding of 'normal' behaviour
Actions Performed	Analysis of the network traffic and flows
Outputs Produced	Notifications of detected anomalies
Tool Interfaces	Network interface
Standards	
Req. Mapping	

Tool Category	*[9.2] Incident Reporting Tools*
SecDevOps Phases	Monitoring
Tool Examples	Nagios, Veeam, datadog, solarwinds, Famatech Advanced IP Scanner, Icinga, LibreNMS, Wireshark, Zabbix, Graylog
Tool Purpose	Monitoring IoT infrastructure components
Inputs Required	Network traffic, Log files, System events, rules
Actions Performed	Depends on the rules and tool used
Outputs Produced	Messages/visualization of harmful events, blocking of certain devices
Tool Interfaces	Tool dependent
Standards	
Req. Mapping	

3 Tools Developed by the SCRATCh Consortium Members

To facilitate agile SecDevOps for IoT system, the partners within the SCRATCh consortium will use a combination of open source tools, existing commercial-off-the-shelf tools, and finally, tools developed within the context of this project.

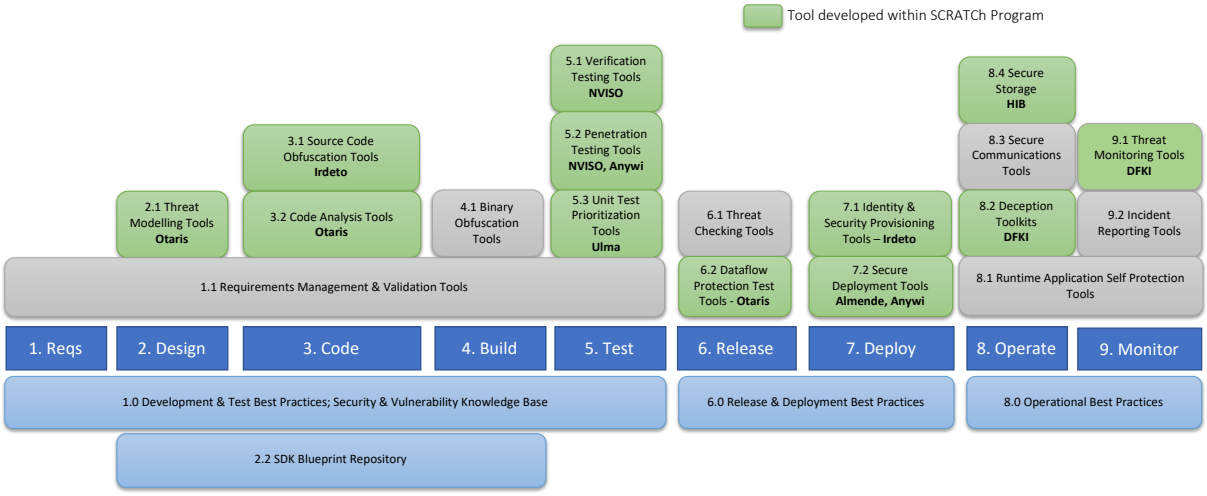

FIGURE 4: SECDEVOPS TOOLS DEVELOPED BY SCRATCH CONSORTIUM PARTNERS

Figure 4 above represents the categories in which SCRATCh partners are current (or plan to) develop tools that fit within the Toolset Framework. More details on these tools are provided in the deliverables associated with Work Package 2 Interoperable Tools.

Additionally, most of these tools will be integrated into the generic demonstrator, or one of the use case specific demonstrators as part of Work Package 4.