SCRATCh

SECURE AND AGILE CONNECTED THINGS

Deliverable

Deliverable

- D1.3a Communication architecture
- D1.4a Device architecture

Work-package:	WP1
Affected milestone:	MS1
Partners involved:	AnyWi
	consider it GmbH
	DFKI
	NXP Semiconductors Germany GmbH
	OTARIS
Date:	18/03/2021
Deliverable version:	v2.1
Editor:	see below
Author(s):	Arne Ehrlich
	Daniel Schneider
	Franklin Selgert
	Karsten Sohr
	Morten Larsen
	Till Witt
Responsible Contact:	Till S. Witt
	NXP Semiconductors Germany GmbH

Copyright © 2018-2022, SCRATCh

Version history

Date	Version	Author	Comment
14.10.2019	0.1	Till Witt (NXP)	Initial version
28.10.2019	0.2	Till Witt (NXP)	Restructuring according to current discussions
30.10.2019	0.3	Franklin Selgert (AnyWi)	Added high level architecture/ domain model and building blocks
22.11.2019	0.9	See authors	Version to be reviewed by consortium
10.12.2019	0.9a	Franklin Selgert (Any WI)	Restructuring and cleanup, adding relation with some standard RA's
16.12.2019	1.0	See authors	Version A of deliverable
01.03.2020	1.1	See authors	Minor rework
11.09.2020	1.2	Daniel Schneider (DFKI)	Added requirements and third threat model (privacy focused)
23.09.2020	1.3	Mehmet Kus, Karsten Sohr (OTARIS)	Added section on technical architecture common in IoT networks; OWASP Risk Rating
26.01.2021	2.0b	Franklin Selgert (AnyWi)	Reshuffling chapters, textual changes to first 4 chapters 4/2 added text DevOps and architecture
18.03.2021	2.1	Till Witt (NXP)	Final review before publishing to ITEA portal

Copyright © 2018-2022, SCRATCh

Table of contents

1. Reference architecture/framework scratch

Used documents:

1. The Industrial Internet of Things Volume G1: Reference Architecture Version 1.9 June 19, 2019
2. Recommendations for commonalities and interoperability profiles of IoT platforms, Revision: 1.00 30-09-2018 Lead partner: ETSI (Create IoT Project)
3. ISO '*ISO/IEC/IEEE 42010:2011*' [2] architecture concepts
4. IoT-A Final architectural reference model for the IoT v3.0 date 15.07.2013

Introduction

This technical report describes a Reference model/architecture to be used as structure for the SCRATCh document. it specifies an IoT Architecture Framework comprising viewpoints and security concerns to add in the development, documentation and communication of the Scratch use cases, toolkit and Secure Development and Operation thoughts (SecDevOps) The reference architecture uses a common vocabulary and a standard-based framework to describe typical viewpoints as business, usage, functional and implementation.

The framework is generic and can be used as an aid to derive concrete architectures and builds on the IoT-A reference model, IIRA and ISO IoT RA concepts, see fig 1.

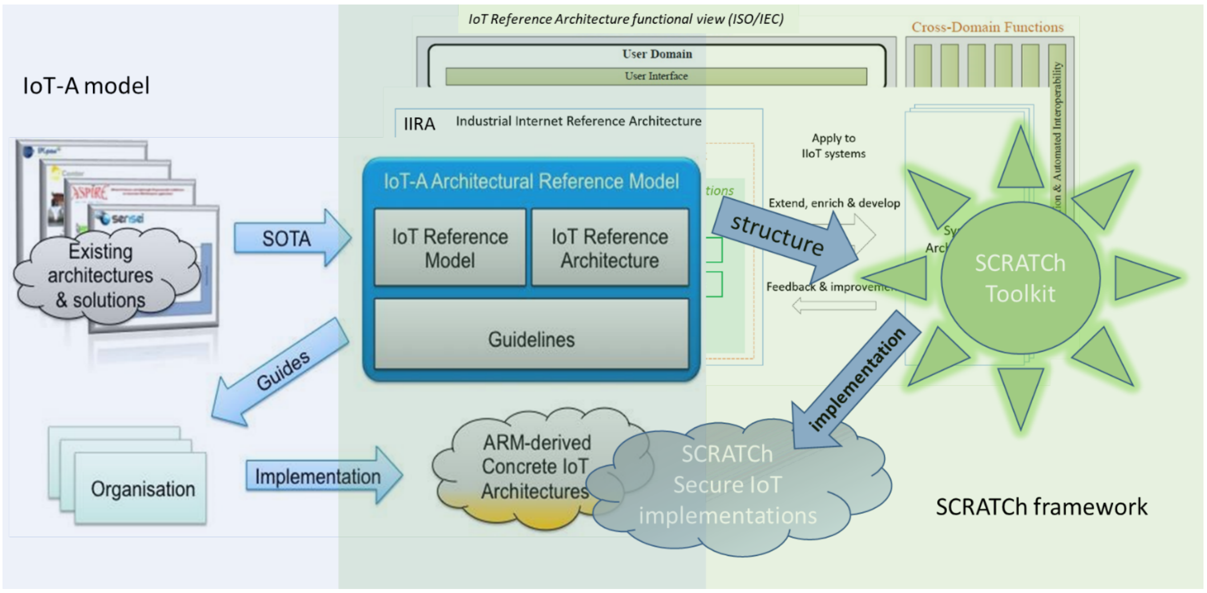

IoT Reference Architecture functional view (ISO/IEC)
Cross-Domain Functions
User Domain
User Interface
IoT-A model
IIRA Industrial Internet Reference Architecture
Existing architectures & solutions
SOTA
IoT-A Architectural Reference Model
IoT Reference Model
IoT Reference Architecture
Guidelines
Guides
Apply to IIoT systems
Extend, enrich & develop
structure
Feedback & improvement
implementation
SCRATCh Toolkit
Organisation
Implementation
ARM-derived Concrete IoT Architectures
SCRATCh Secure IoT implementations
SCRATCh framework

Figure 1: IOT-A, IIRA and ISO models and SCRATCh framework

Purpose

This SCRATCh framework technical report addresses two primary purposes. For all SCRATCh work efforts, it is the framework that links other technical documents and technical activities of the consortium. For a broader IoT community, it provides guidance and assistance in the development, documentation, communication, and deployment of Secure IoT systems.

Scope

This framework is meant as a guidance to be used for constructing specific IoT system architectures. It is not describing all the viewpoint as mentioned in '*ISO/IEC/IEEE 42010,* and the framework is not to be seen as yet another Reference architecture for IoT, but rather as a guide that provides an overview of current thoughts concerning IoT architectures. The framework will highlight the security aspects that needs to be addressed in any to be developed IoT architecture.

The following unique terms and definitions are used in this document:

- architecture framework conventions and common practices for architecture description established within a specific domain or stakeholder community
- *Architecture viewpoint* (viewpoint for short): conventions framing the description and analysis of specific system concerns.
- *Stakeholder*: an individual, team or organization having an interest in a concern and, by extension an interest in, the viewpoint and system.
- *Architecture view*: the collection of ideas describing, analyzing, and resolving the set of specific concerns in a viewpoint using the conventions set forth in that viewpoint. A view includes one or more models.
- *Reference architecture*: the outcome of applying the architecture framework to a class of systems to provide guidance and to identify, analyze and resolve common, important architecture concerns. A reference architecture can be used as a template for concrete architecture of systems of the class.

2. A framework for SCRATCh

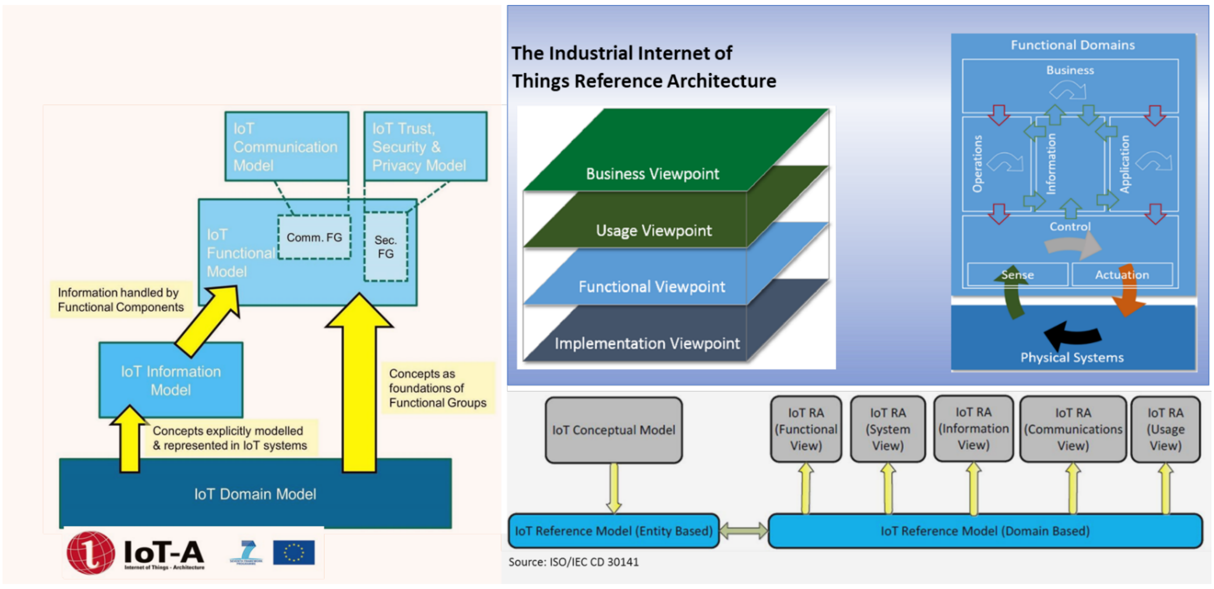


Figure 2: Three Approaches

Many reference architectures exist in the IoT landscape, most of them follow more or less the structure as presented in ISO 42010, but have different ways to present the viewpoints. Most consistent in representation across all reference architectures Is the functional and implementation or physical viewpoint. As a start of the SCRATCh framework, we will start our description using these viewpoints. The Implementation viewpoint is a more project dependent viewpoint and will only be presented in the different use cases. Within the framework we will limit this viewpoint to a schematic three tier architecture to be used if needed in the security assessments (fig3)

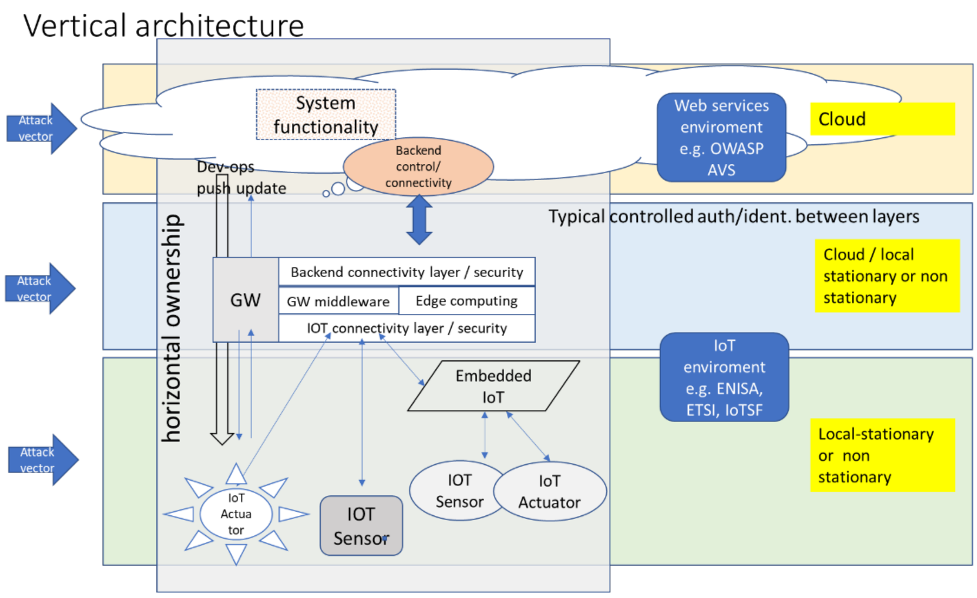

Vertical architecture
Attack vector
System functionality
Backend control/ connectivity
Web services enviroment e.g. OWASP AVS
Cloud
Dev-ops push update
Typical controlled auth/ident. between layers
Attack vector
GW
Backend connectivity layer / security
GW middleware
Edge computing
IOT connectivity layer / security
Cloud / local stationary or non stationary
horizontal ownership
IoT enviroment e.g. ENISA, ETSI, IoTSF
Embedded IoT
Attack vector
Local-stationary or non stationary
IoT Actuator
IOT Sensor
IOT Sensor
IoT Actuator

Figure 3: simple three layered implementation view

3. Functional viewpoints/model

For the functional viewpoint we will use the layered structure as presented in the Create IoT project a structure that maps very well on the ISO Layers. The Create IoT project analyzed several implemented architectures within the Large scale IoT projects of the EU and found these common layers, on top of these layers we map the two functional models of IIRA and IoT-A

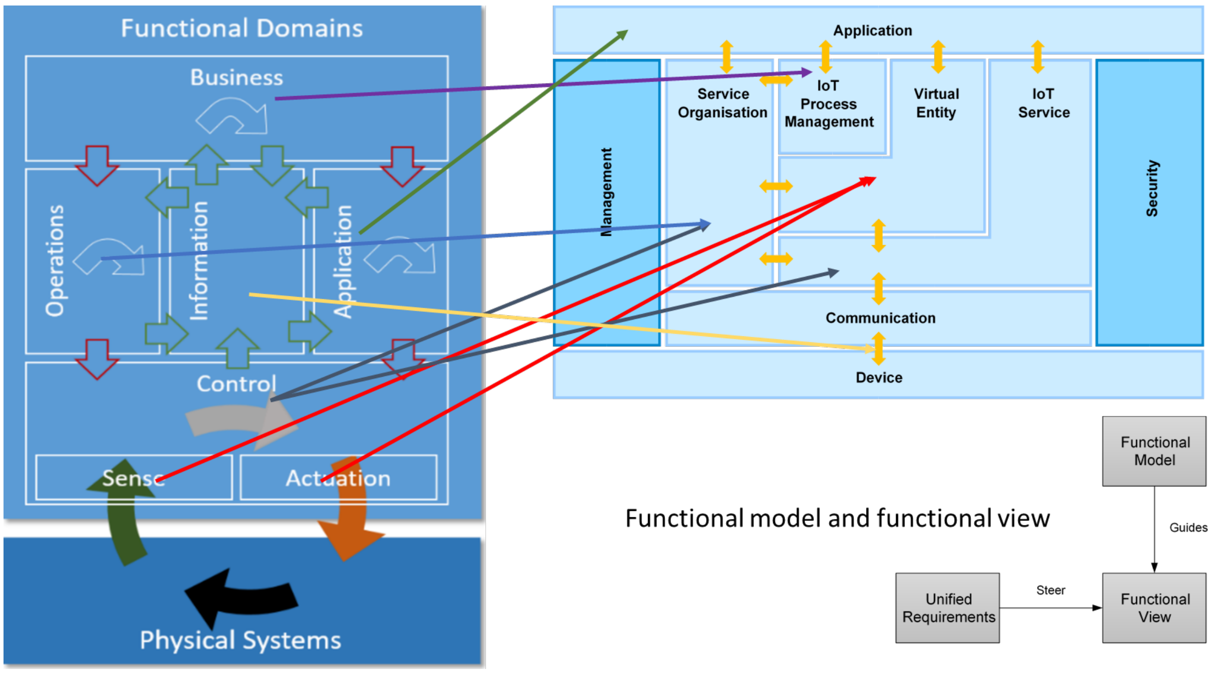


Figure 4: Mapping of IIRA (left) and IoT-A (right)

In the IoT-A reference architecture the terms functional model and functional view are used, combining the model with requirements should lead to a functional view (see fig 4). The Functional View describes the system 's runtime Functional Components, including the components responsibilities, their default functions, their interfaces, and their primary interactions.

The IIRA talks about functional domains as model for decomposition of IoT system, with information flows (green arrows) and decision flows (red arrows). Both models have their own approach but also have commonalities and can be plotted on layers as described by the Create IoT project (see table 1).

Table 1: mapping functional models to the create IoT layers

Layers	Maps to IIRA/IoT-A	Description IIRA and IoT-A
Collaboration and processes	Business domain Process management	functional domain for implementing business functional logic conceptual integration of (business) process management systems
Applications	application	represents the collection of functions implementing application logic that realizes specific business functionalities.
Service	Operations Service Organization	Management and operation of the control domain. It represents the collection of functions responsible for the provisioning, management, monitoring and optimization of the systems in the control domain used for composing and orchestrating Services of different levels of abstraction, it effectively links the Service requests from high level functions such as IoT Process Management, or external applications, to basic services that expose Resources
Abstraction	Virtual entity Information domain	Virtual Entity level models higher-level aspects of the physical world, and these aspects can be used for discovering Services. Examples for interactions between applications and the IoT system on this abstraction level are —Give me the outdoor temperature of Car xyz. Virtual entities can be used for analytic and AI type of applications, without interfering with the actual IoT system operation. The *information domain* is a functional domain for managing and processing data. It represents the collection of functions for gathering data from various domains, most significantly from the control domain, and transforming, persisting, and modeling or analyzing those data to acquire high-level intelligence about the overall system.[1] The data collection and analysis functions in this domain are complementary to those implemented in the control domain
Storage	N/A	Not a separate function in the models, storage is a generic function available for use by several layers and can be external or physical internal.
Processing	Control IoT service	The *control domain* implements industrial control systems. The core of these functions comprises fine-grained closed-loops, reading data from sensors applying

[1] Possibly in a hierarchy, at several levels.

		rules and logic, and exercising control over the physical system through actuators. IoT service is the functional abstraction of the sensor and actuator, effectively providing the interface between the physical and functional world.
Networks & Communications	IIRA the Arrows Communication	Information and control flows between functional and physical domains. Abstracts the variety of interaction schemes derived from the many technologies belonging to IoT systems and provides a common interface to the Service layer. Provides an interface for instantiating and for managing high-level information flow. Starting from the top layers of the ISO/OSI model it considers data representation, end to end path information, addressing issues (i.e. Locator/ID split), network management and device specific features.
Physical / Device Layer	Physical system Device	Physical layer actual devices, gateways networks, modules, drivers, MPU/MCU etc.

This gives us the possibility to map function to layers and draw functional diagrams as figure 4 if needed. The implementation view will give us the means to place the functional components in a logical relation to each other. Management and security in most models are verticals that span multiple layers to fulfil their special role. Implementing this role would mean that every functional block has a security and management component to it, that combined make this vertical overarching control function. In SCRATCh security is viewed from a process point of view, as in addressing it as an essential activity in a most phases of the DevOps process.

4. Device architecture

Architecture concepts mentioned apply to complete systems, if narrowed down to the level of a single device some viewpoint does not need to be considered on the level of a reference architecture, e.g. the business viewpoint. A functional viewpoint and implementation viewpoint can be drafted.

Below figure 6 is an example of a functional viewpoint from the sunrise project. The viewpoint can be used to review all functional blocks using the security viewpoint described in chapter 5 and investigating the architecture on potential weaknesses.

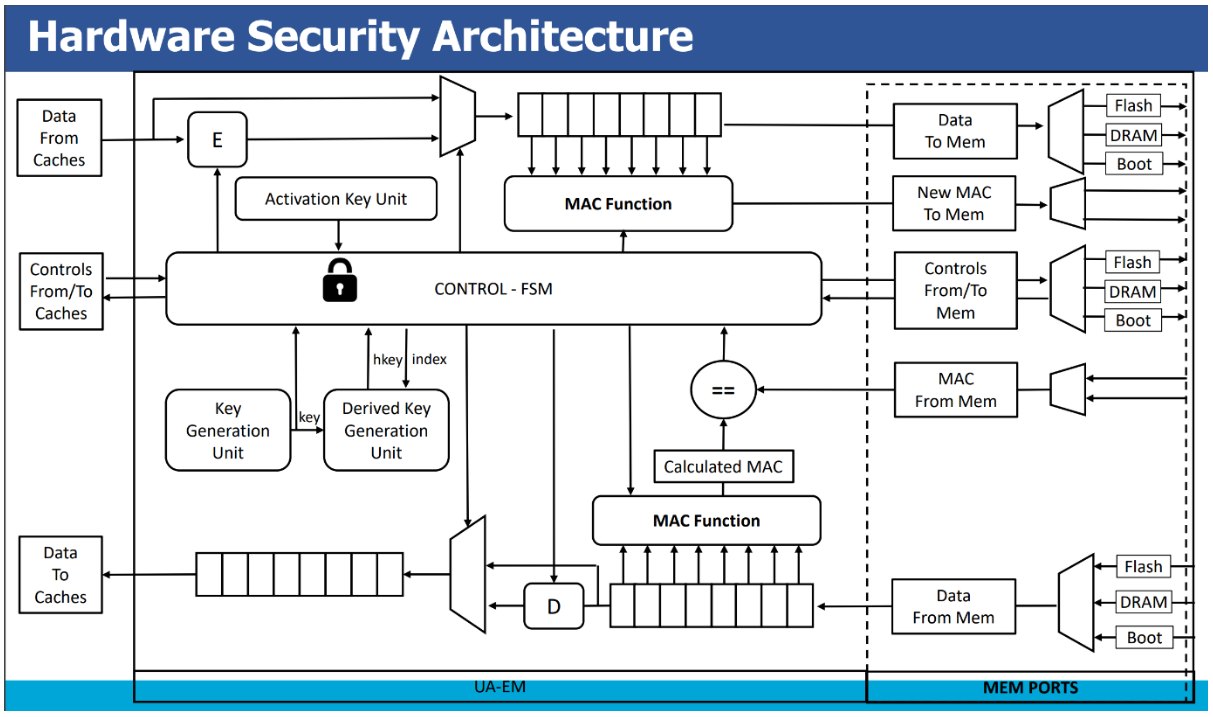


Figure 5 (source TUD project Sunrise)

5. Security view

This architecture is a first step in the holistic approach of SCRATCh in building a type of security framework. Objectives of a security framework* (source H2020 create IoT)

The IoT security framework must consider the following elements:

1. Ensuring IoT security mechanisms
2. Ensuring IoT data protection
3. Ensuring IoT system resilience
4. Providing IoT system/application trust

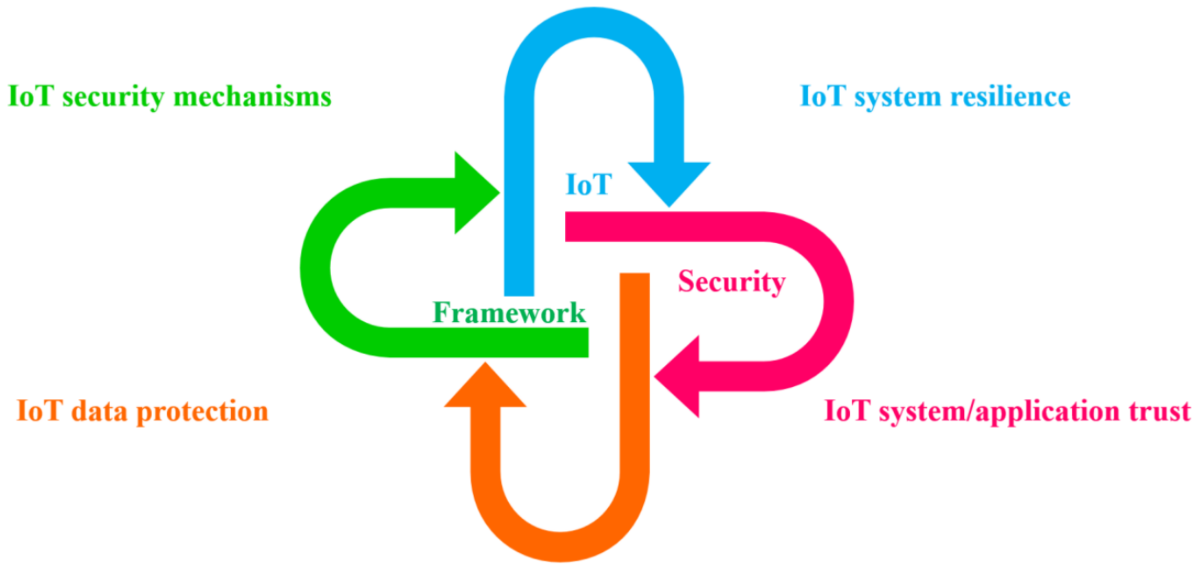


Figure 6 IOT security framework dependencies (source H2020 Create IoT)

Properties for security are often based on the established CIA triad: confidentiality, integrity, and availability* (H2020):

1. Confidentiality ensures that information is not made available or disclosed to unauthorized individuals, entities, or processes. Examples of measures for achieving or enhancing confidentiality include protected transmission of collected data, protected access with suitable authentication schemes, protected processing of data, and protected storage.
2. Integrity ensures the accuracy and completeness of data over its entire life cycle. Examples of measures for achieving or enhancing integrity include schemes such as digital signatures.
3. Availability ensures accessibility and usability upon demand by an authorized entity. Examples of measures for achieving or enhancing availability include preventing service disruptions due to power outages, hardware failures, or security denial of service attacks using schemes such as redundant systems.

One of the meaning for security in the oxford dictionary is “the activities involved in protecting a country, building or person against attack, danger, etc.” Security as an activity to achieve this a framework needs to be embedded in a process in the case of SCRATCh the aim is to use the DevOps process.

6. SecDevOps view

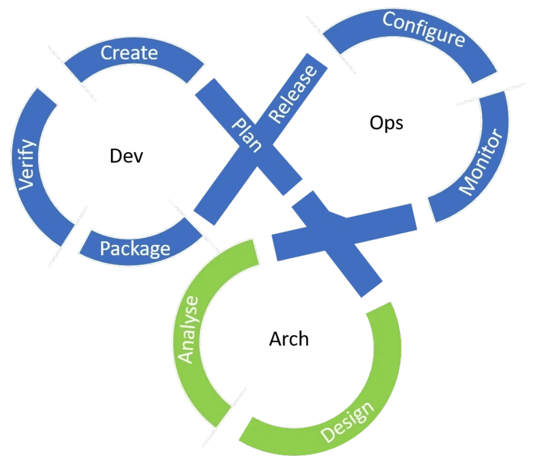


Figure 7 ArchOps: Extending the DevOps Loop (https://smarchy.com/blog/f/archops-part-ii-extending-the-devops-loop)

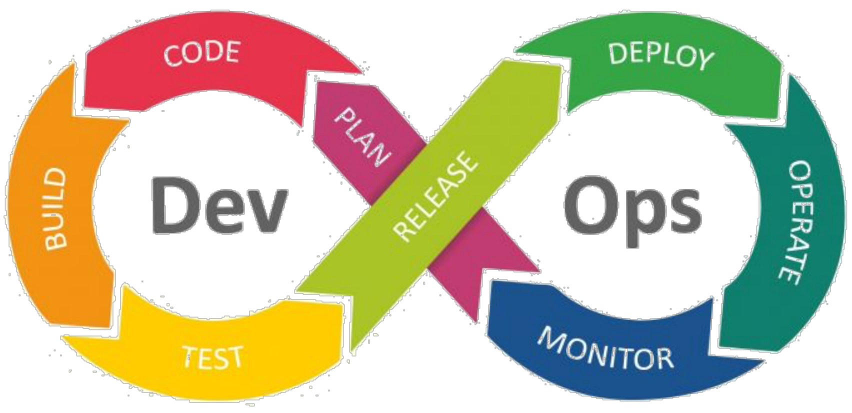


Figure 8 DevOps and architecture

What is the relation between architecture and SecDevOps? There is no direct relation as DevOps is a process-oriented approach and a Refence architecture is an inherent outcome of a process. In the DevOps way of thinking two approaches can be suggested.

1. Incorporate the architecture function in the DevOps Team, 0rganizational approach
2. Extend DevOps with an extra loop between Monitor and Plan to design and analyze the architecture, Process approach

The assumption in SCRATCh is: the plan Phase of DevOps is the phase where normally the system architecture is drafted. From a security perspective important artifact of the process because it is the input for a security analysis like e.g., Stride. In SCRATCH we emphasize a holistic approach meaning not only use the architecture to make a safe system but also design the architecture to keep the system safe.

Security is an often a missing part left to be solved by security experts in the release phase. SCRATCh proposes a methodology to do a proper shift left of security and put is at the start of development in the plan phase. To support this methodology, we want to inject tools in the cycle that build up the trust and Security of the system at hand. Tools and methodology are described in deliverable D2.1.

DevOps and security viewpoint

1. Ensuring IoT security mechanisms
    a. Input of design constrains from best practices and standards
    b. Verify the implementation of the design constrains
    c. Monitor operational system and monitor threats reports on used code libraries
2. Ensuring IoT data protection
    a. The design constrains reflect the level of protection needed.
    b. Implementation of protection mechanism like secure storage, secure elements, and encryption mechanisms
3. Ensuring IoT system resilience
    a. A basic design constrain is the capability of recovery from failure, as a result of this secure recovery mechanism need to be in place, specific manifest of this is a method for secure firm and software update
4. Providing IoT system/application trust
    a. Part of the architecture is the embedding of trust mechanism.
    b. The verification of the trust mechanisms

7. Best Practices and tools to Design a secure system

Having layout, the framework we can start a design process according to certain guidelines and rules. There are certain basic rules that apply:

The components of the system shall combined provide the necessary management and security requirements:

1. Complying with the security framework of chapter 4.
2. Providing the necessary interaction points to make implementation of Sec Dev Ops possible.

The process starts with selecting a proper set of design constrains, as an example one could build on the OWASP top 10 or the best practices of ENISA. Next step would be a draft architecture on system level and if needed for security reasons also on device level. This draft architecture must also describe the interaction points to guarantee the capability to keep the system safe and up to date as well as to gather monitoring data.

Copyright © 2018-2022, SCRATCh

1st proposed step: STRIDE

A simple look at "what can go wrong in this system we're working on" can be achieved using STRIDE. This consist of looking at the threats and the desired property to be achieved:

Threat	Desired property
Spoofing	Authenticity
Tampering	Integrity
Repudiation	Non-repudiability
Information disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

If we look at the ENISA study (Baseline Security Recommendations for IoT in the context of Critical Information Infrastructures November 2019) where numerous threats are listed, none of the Stride threats are named specifically. All threats mentioned can be seen as a result of an exploit of one or more of the threats listed in the STRIDE model.

E.g. (ENISA threats in bold)

1. **Man in the middle attack:** an attempt to tamper with information in order to destroy the integrity of the system. Such an attack will have an effect on several STRIDE factors, e.g. tampering, spoofing, repudiation.
2. **Device modification**: A example of tampering by exploiting a device vulnerability.
3. **Software vulnerabilities:** a more generic category that can lead to information disclosure, authorization issues, etc.
4. **DDoS and Information gathering** are two threats that match directly on Denial of Service and Information disclosure of the STRIDE model

Threat modeling in the design Phase is a start to make the system more resilient. In scratch we will test this hypothesis by starting with STRIDE in the design phase for each use case of the project. After which we will try to abstract the commonalities of this approach into a threat modeling method to be used in the design phase. This covers mainly the architectural viewpoint 1 and 2 the layered and domain model.

2nd proposed step: DREAD

- **D**amage – how bad would an attack be?
- **R**eproducibility – how easy is it to reproduce the attack?
- **E**xploitability – how much work is it to launch the attack?
- **A**ffected users – how many people will be impacted?
- **D**iscoverability – how easy is it to discover the threat?

DREAD lies more in the realm of operation and deployment and should be covered by the processes in the SecDevOps cycle. Why, e.g. damage will depend on how fast you can react on a successful exploit, which again depends on how you monitor your system (discoverability). Reproducibility is heavily dependent on your actual system layout as is affected users. Within Scratch the emphasis from the Dread analysis will be on Reproducibility, Exploitability, and Discoverability. Damage and Affected use cases are very much sector and system dependent. DREAD will be performed in each use case with the mindset to discover commonalities that can enrich the SecDevOps process.

3rd proposed step: LINDDUN

In addition to the security-focused STRIDE and DREAD frameworks, the LINDDUN framework can be employed to strengthen privacy [Robles-Gonzalez.2020]. There is some overlap in the presented threats, since LINDDUN is originally based on STRIDE[Deng.2011].

- **L**inkability - How easy is it to link two or more items of interest (IOIs)?
- **I**dentifiability - How easy is it to identify the subjected associated with an IOI?
- **N**on-repudiation - Can plausible deniability be broken?
- **D**etectability - How easy is it to detect an IOI?
- Information **D**isclosure - Is personal information exposed to unauthorized parties?
- Content **U**nawareness - Is the user aware of the information disclosed to the system?
- Policy **N**oncompliance - Does the system comply with its stated privacy policy?

4th proposed step: OWASP Risk Rating Methodology

Microsoft has deprecated DREAD within their SDL as its results are sometimes arbitrary (see Shostack [10], page 180). One alternative to DREAD is the OWASP Risk Rating Methodology published by the Open Web Application Security Project [11]. The OWASP Risk Rating Methodology has the advantage that it uses the CVSS score to rate architectural risks. This allows one to have a common risk rating methodology for threat modeling, code reviews, and penetration testing such that these different kinds of software security risks can be handled in a unified way.

The OWASP Risk Rating Methodology comprises the following steps:

1. Identifying a Risk
2. Factors for Estimating Likelihood
3. Factors for Estimating Impact
4. Determining Severity of the Risk
5. Deciding What to Fix
6. Customizing Your Risk Rating Model.

Broadly speaking, the methodology defines several factors concerning the likelihood as well as the severity of risks and calculates the product of both as usual in risk management. Both the scales for severity and likelihood are within the range of 0..9. These values are then merged into an overall risk value according to the following two tables (taken from [7]):

Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

Tabelle 1: Likelihood and impact Levels of OWASP Risk Rating Methodology.

Overall Risk Severity				
Impact ↓	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
		Likelihood →		

Tabelle 2: Overall Risk Rating of OWASP RISK Rating Methodology.

The details of the OWASP Risk Rating Methodology including all factors influencing likelihood and impact can be found in [11].

Copyright © 2018-2022, SCRATCh

Requirements

Requirement engineering is an important success factor in software projects [Durmic.2020]. Two types of requirements can be distinguished: functional requirements (FR), which define the result behavior of a system, and non-functional requirements (NFR), which define a system's qualities and constraints [Glinz.2017].

Functional	Non-Functional
Authentication	Confidentiality
Authorization	Integrity
Session Management	Availability
Error and Exception Handling	Non-repudiation
Vulnerability Scan	OS Agnostic Middleware
Code Analysis	Certified Middleware
Test Framework	Certified Secure Element
Gateway Smoke Test	PSA Compliance
Authenticated Logging	Secure and up-to-date 3rd Party libraries
Confidential Logging	Attestation
Authenticated Firmware Upgrade	Accessible Documentation
Confidential Firmware Upgrade	Integration of scratch devices
Secure Boot	Integration of non-scratch devices
Deployment Automation Device Management	Migration of legacy devices
Secure Element personalization	Docum. of Migration of legacy devices
Unique identifier	Un-linkability
Key storage	Anonymity/Pseudonymity
Authorized Device Reset	Plausible Deniability
Automated Software Deploy & Delivery process	Undetectability
Automated Firmware Deploy & Delivery process	Policy compliance
Feedback of the Firmware/Software Deployment	
Automated Testing	
Authenticated Interface Access	

Copyright © 2018-2022, SCRATCh

8. Security in Common Architecture in IoT Networks

Common Architecture in IoT Networks

In Figure 7, an architecture that is common in IoT networks including industrial IoT is shown. One part of this network are IoT devices (e.g., smart home devices, industrial robots, and medical devices). Usually these IoT devices are connected via low-power wireless networks, such as ZigBee and Z-Wave. In addition, these IoT devices are controlled by a more powerful gateway, which in turn is connected to other networks, such as WAN or LAN.

The IoT gateway itself is often controlled with the help of mobile applications as well as an IoT cloud. This IoT cloud takes over the job of device management including software update management, provisioning remote access (e.g., sending remote commands to devices and retrieving events from the IoT devices), remote configuration, and obtaining statistics on the IoT network.

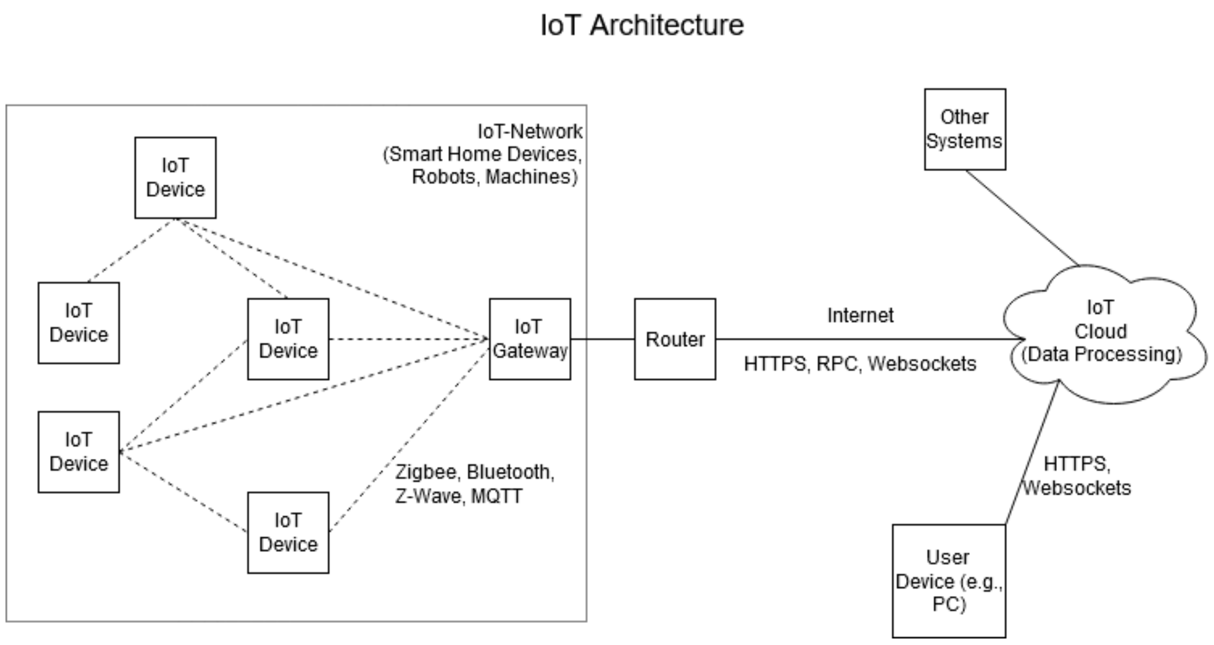


Figure 7: A common IoT architecture.

Security Considerations in IoT Networks

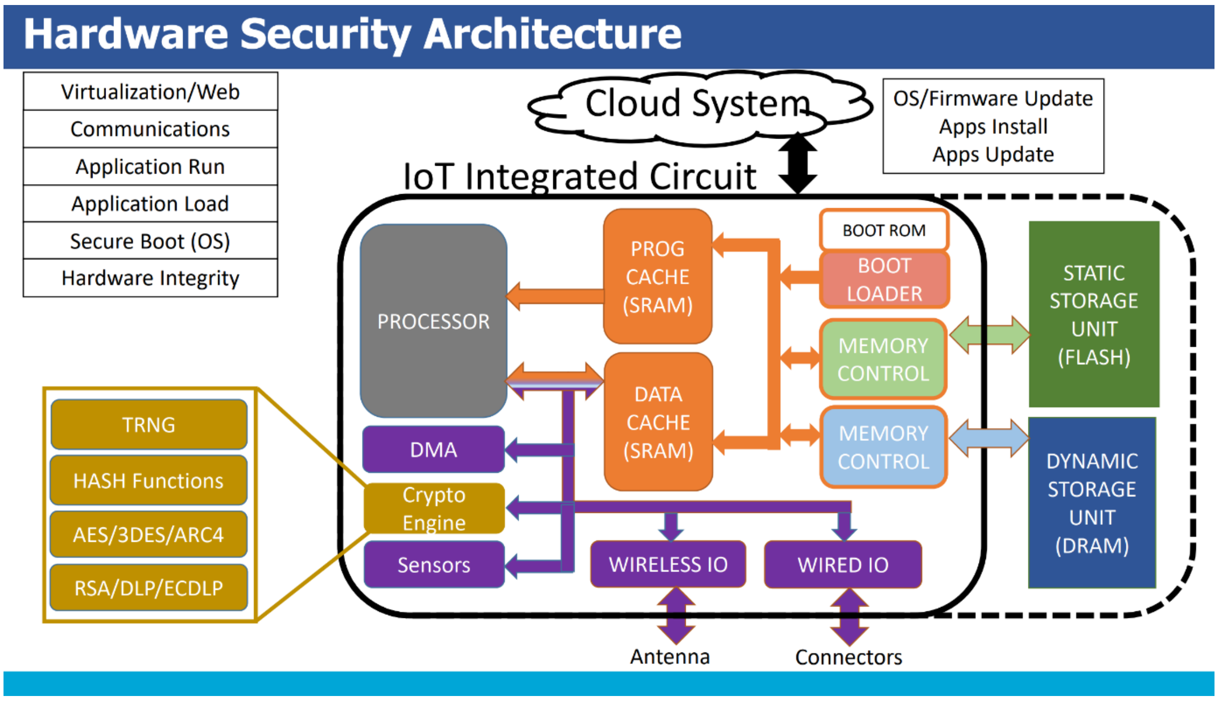
There are different communications channels that must be secured in IoT networks. In particular, security-by-design approaches such as STRIDE analyses have to consider the following communications channels:

- Device to device (M2M): Communication via low-power wireless networks (e.g., ZigBee, Z-Wave, MQTT)
- Gateway to device (and vice versa): Communication via low-power wireless networks
- Gateway to cloud (and vice versa): Communication often via HTTPS/REST, RPC, LoRaWAN, TR-069, and MQTT.
- Possibly smartphone connections (local and remote access): HTTP(S)-based communications, Web sockets.

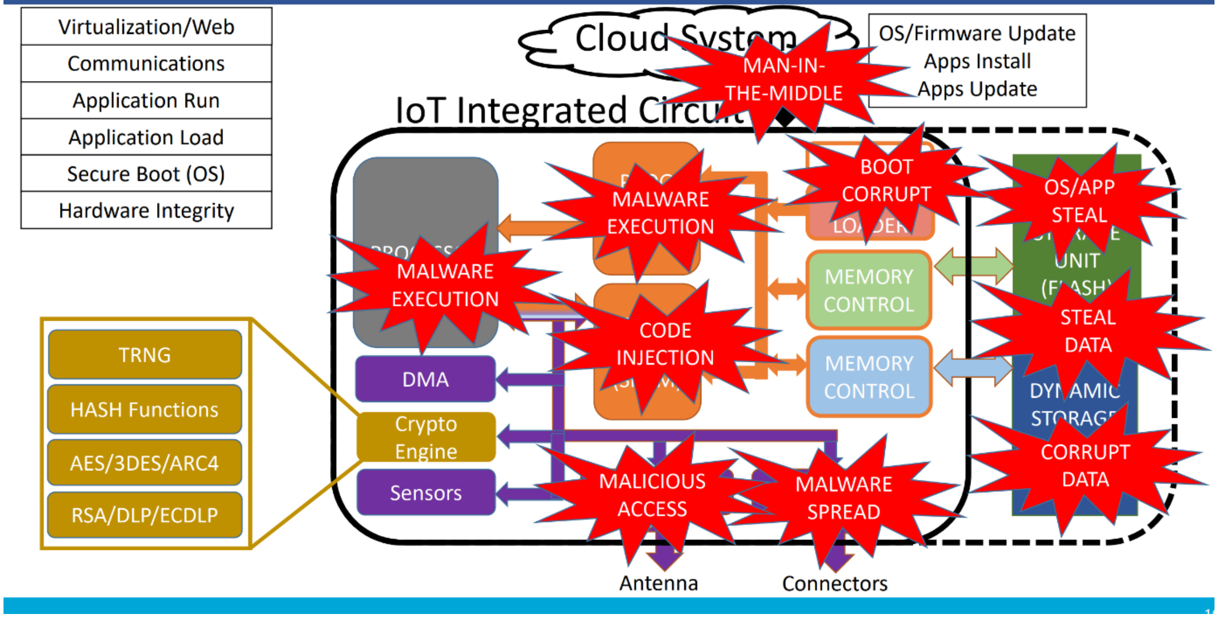
The aforementioned communications channels need to be secured according to the CIA protection goals and mutual authentication mechanisms. Of special interest for STRIDE analyses are the cloud – because it could be a single point of failure of the whole IoT network– and the IoT gateway, which controls all the devices and must deliberately open ports for remote access. Software security issues must be considered foremost for these two central components of an IoT network.

Security Considerations IoT Devices

Source TUD sunrise project



Hardware Security Architecture



9. References

1. The Industrial Internet of Things Volume G1: Reference Architecture Version 1.9 June 19, 2019
2. Recommendations for commonalities and interoperability profiles of IoT platforms, Revision: 1.00 30-09-2018 Lead partner: ETSI (Create IoT Project)
3. ISO '*ISO/IEC/IEEE 42010:2011*' [2] architecture concepts
4. IoT-A Final architectural reference model for the IoT v3.0 date 15.07.2013
5. H2020 CREATE IoT D06_02_WP6_2018
6. Robles-González, Antonio & Parra-Arnau, Javier & Forné, Jordi. (2020). A LINDDUN-Based Framework for Privacy Threat Analysis on Identification and Authentication Processes. Computers & Security
7. Deng, Mina & Wuyts, Kim & Scandariato, Riccardo & Preneel, Bart & Joosen, Wouter. (2011). A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements. Requir. Eng.. 16. 3-32. 10.1007/s00766-010-0115-7.
8. Durmic, Nermina. (2020). Information Systems Project Success Factors: Literature Review. Journal of Natural Sciences and Engineering.
9. Glinz, Martin. (2017). A Glossary of Requirements Engineering Terminology.
10. Adam Shostack. 2014. *Threat Modeling: Designing for Security* (1st. ed.). Wiley Publishing.
11. Jeff Williams. 2020. OWASP Risk Rating Methodology. Accessible under: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology