A Vulnerability Management Solution for constrained IoT devices with a Trusted Execution Environment using a Hardware Root of Trust

Dissertation zur Erlangung des akademischen Grades einer Doktorin der Naturwissenschaften (Dr. rer. nat.)

am Fachbereich Mathematik und Informatik der Freien Universität Berlin

vorgelegt von

Larissa Groth

Berlin, den 01.11.2022

Erstgutachter: Prof. Dr.-Ing. Jochen H. Schiller, FU Berlin
Zweitgutachter: Prof. Dr. rer. nat. Jochen Seitz, TU Ilmenau

Tag der Disputation: 19.01.2023

Name: Groth
Vorname: Larissa

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Dissertation selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Dissertation wurde in gleicher oder ähnlicher Form noch in keinem früheren Promotionsverfahren eingereicht. Mit einer Prüfung meiner Arbeit durch ein Plagiatsprüfungsprogramm erkläre ich mich einverstanden.

Datum: 01.11.2022

(Larissa Groth)

Abstract

The popularity and prevalence of Internet of Things (IoT) devices has been ever increasing. They have found their way into our everyday lives and increasingly transform our living environments into smart homes. However, most of these constrained devices do not possess sufficient computational power, memory, and battery runtime in order to implement security features that are common for general purpose personal computers. Hence, the increasing numbers of interconnected consumer IoT devices are followed by an increase of their attack surface and vulnerabilities.

The following thesis approaches this security issue by providing a novel approach for a Runtime IoT Security Score that provides the inexperienced user of a smart home system with profound insight into the security state of the connected IoT devices during runtime. This is achieved by combining Vulnerability Assessment with Trustworthiness Assessment of the connected devices, which has never been proposed before and represents a very valuable contribution to the state of current research.

In addition to the Runtime Security Score, a holistic concept for a Vulnerability Assessment and Management (VAM) solution is proposed as another main contribution of this thesis. The effective and functional interoperability of all relevant components specified in this concept is shown with a Proof of Concept implementation.

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
CBOR	Concise Binary Object Representation
CoAP	Constrained Application Protocol
CM	Continuous Monitoring
CoSWID	Constrained Software Identification
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DHCP	Dynamic Host Configuration Protocol
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HW	Hardware
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ITAM	Information Technology Asset Management
IoT	Internet of Things
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LLDP	Link Layer Discovery Protocol
MDK	Microcontroller Development Kit

MPU	Memory Protection Unit
MUD	Manufacturer Usage Description
MMU	Memory Management Unit
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
OS	Operating System
PC	Personal Computer
PMP	Physical Memory Protection
PoC	Proof of Concept
RA	Remote Attestation
RATS	Remote Attestation Procedures Architectures
RFC	Request for Comments
RISC	Reduced Instruction Set Computer
RoT	Root of Trust
SW	Software
SBOM	Software Bill of Material
SWID	Software Identification, or software identification tag
SM	Software Module
SP	Software Provider
SPDX	Software Package Data Exchange
SoC	System on Chip
TA	Trustworthiness Assessment
TAM	Trusted Application Manager
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
UDP	User Datagram Protocol
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
UUID	Universally Unique Identifier
VA	Vulnerability Assessment
VM	Vulnerability Management
VAM	Vulnerability Assessment and Management
VMDR	Vulnerability Management, Detection, and Response
WSGI	Web Server Gateway Interface
XML	Extensible Markup Language
XSD	XML Schema Definition
YANG	Yet Another Next Generation

Chapter 1

Introduction

1.1 Motivation

In the last few years, the popularity of Internet of Things (IoT) devices has been ever increasing. This results in more and more devices being connected to the Internet while becoming smaller and smaller in terms of form factor, CPU performance, memory, and energy consumption. The Internet Engineering Task Force (IETF) refers to this device class as ”constrained devices” [10]. With the rise of the IoT, simple appliances became smart and interconnected and found their way into smart homes and our everyday lives. Whereas in 2017, the growth of IoT devices had been estimated to reach 8.4 billion IoT devices by 2017 and 20.4 billion by 2020 [23], a more recent forecast from 2020 anticipates 28 billion IoT devices by 2025 [32].

In addition to the aforementioned trend of increasing numbers of interconnected consumer IoT devices, more and more embedded constrained devices support software extensibility and updateability [40]. The combination of both network connectivity and software updateability has vastly extended the attack surface. Prominent attacks in the IoT smart home domain have e.g. been the Mirai botnet [68], that represents a multi-tier Distributed Denial of Service Attack as classified and described by Djuitcheu et al. in [15], and the Zigbee worm proposed by Ronen et al. [52].

In contrast to traditional Personal Computers (PC), IoT devices most often lack security by design, leading to common vulnerabilities [54]. In the IoT domain, hardware and software are closely coupled; often, they are provided by one manufacturer and are both proprietary. Consequently, users of an IoT device are most often not able to install additional (security) software but have to rely on the security measures implemented by manufacturers [54]. Additionally, the devices' limitations in terms of computational power, memory and power consumption lead to the fact that well-known security measures for high-performance processors such as virtual memory, memory protection or privilege levels are not easily applicable to IoT devices [42]. Their network connectivity opposes additional limitations: Such constrained devices need to rely on low-power network communication protocols as they cannot afford the power consumption required by Wi-Fi [72]. All this leaves the devices prone to a range of malware and other attacks.

In order to counter these threats, a number of security architectures for networked embedded devices are available today. Among the most prominent representatives are ARM's Trustzone [44], RISC-V [51] and Sancus [42]. All of them have in common that they come without a Memory Management Unit (MMU). Hence, process isolation, which is a common security feature for PCs, cannot be implemented through virtual address spaces. Instead, a more reduced and resource-saving memory protection is implemented using different Hardware Root of Trust (RoT) mechanisms. Although the specific implementations differ, all approaches divide the processor into at least two areas: a secure Trusted Execution Environment (TEE) and a non-secure world. The transition between both areas is well-defined and the secure world enjoys a special hardware protection. This approach does not provide the same extensive possibilities as process isolation using an MMU does, but the underlying idea is comparable and allows for a basic separation of software.

Although it can be concluded here that IoT devices have an increasing impact not only on our everyday lives, but also on critical infrastructures, they are most often not well-secured and "'security' is not a word that gets associated with this category of devices, leaving consumers potentially exposed to massive attacks" [54]. As a result, calls for "security by design" to be built in and to secure the device on several system levels are getting louder all the time. Hence, making the device's security state measurable and therefore comparable would then put the end user in a much more comfortable position.

In order to improve the consumer's purchasing decision, informative labels are known from many sectors. Nutrition facts labels are supposed to improve consumers' food choices [66]. The fuel economy and environment label provides car buyers with information about the fuel economy, fuel cost, and environmental impacts of the considered vehicle [65]. Information on the energy consumption and an energy efficiency classifier are provided by the EU energy labels for many classes of electrical devices [19]. Analogously, informative pre-purchase labels that represent an IoT device's security or privacy state is subject of ongoing research. Amongst others, Shen et al. propose such an IoT Security and Privacy Label [54].

According to Shen et al., pre-purchase labels may not be limited to static device factors. Security vulnerabilities, e.g. in the IoT device's firmware, can be patched through firmware updates. Therefore, the security and privacy state of the device under consideration may change over time. This idea of dynamic metrics can be extended towards online security states of the IoT device in operation.

I propose such an extension towards a set of security metrics that measure the security state of the device during operation in a smart home network. Such a security index may support the user's buying decision (”which of the considered devices is the most secure?”) but can also serve as online index for the current system security state at runtime. Vulnerability Assessment (VA) and Vulnerability Management (VM) are wide areas of research both in industry and academia. Whereas protective measures are well-established for PCs (firewalls, e-mail filters, anti-virus scanners, etc.), solutions dedicated to the IoT domain are scarce. Nevertheless, the aforementioned security architectures provide functions enabled through hardware that can be utilized to actually attest the state of software running on the device and thereby provide interesting opportunities for enhancing this security index. To the best of our knowledge, combining the idea of a runtime security index with fundamental hardware support that provides a hardware root of trust is a novel approach.

1.2 Contributions

Constrained IoT devices offer a wide field of attack and suffer from a large number of vulnerabilities. Hence, they require security measures by design on all system levels. Those security measures are not as well-known and generic as those for PCs are nowadays. Actually quite the contrary is true: several approaches by various manufacturers exist. Therefore, users would benefit from a security state that is measurable and comparable across different devices. With this thesis, I make the following contributions to meet this need.

I propose the concept of a software system architecture for networked constrained IoT devices, that can provide the user with a profound insight into the security state of the connected devices during runtime. I achieve this by combining findings from the following areas of ongoing research. Approaches for pre-buying security labeling exist and the general need for a solution during runtime is acknowledged, but not solved. Approaches for Vulnerability Assessment exist, but do not target IoT devices in a smart home environment. Approaches for security indices for IoT devices exist, but do not utilize Hardware Root of Trust mechanisms.

Vulnerability Assessment always requires an underlying understanding of the software fragment possessing the vulnerability. Approaches for this partitioning of Software into Software Modules and models for its description as Software Bills of Material (SBOM) exist and represent a decisive building block for automated Vulnerability Assessment.

Therefore, I propose a precise specification for a standardized SBOM that can be used in a network with heterogeneous IoT devices.

Another important premise for Vulnerability Assessment is the mapping of Software Modules to published vulnerabilities. The largest and most widespread system to correlate data of publicly disclosed vulnerabilities with a standardized description across tools, databases, and people is provided by the Common Vulnerabilities and Exposures (CVE) system. There needs to be an entity that takes responsibility for assigning CVE entries to Software Modules. Most often, though not necessarily, this is going to be the Manufacturer.

Therefore, I propose a concept for the effective interaction of the proposed Vulnerability Assessment and Management solution with the constrained IoT device, its Manufacturer, and a CVE database that includes all relevant specifications and descriptions of interfaces.

In order to provide the user with a profound insight into the security state of the smart home system at runtime, I propose a precise specification for the calculation of a novel online IoT security score that represents a significant enhancement of existing pre-purchase IoT security indices and brings together several branches of research.

Based on these specifications, I propose a Device and Vulnerability Management concept for the initial connection of a constrained IoT device within the network to its operation. This includes the collection of all data on vulnerabilities and trustworthiness necessary to compute the aforementioned security score during normal operation of the constrained IoT device.

To show the general practical feasibility, I propose a proof-of-concept implementation based on the actual implementation of a Gateway application and a mock-up of a constrained IoT device that provides the following features:

- the initial device connect with transmission of an SBOM in accordance with the proposed SBOM file specification

- an online Vulnerability Assessment for all registered Software Modules with periodic checks for new disclosed vulnerabilities and updated vulnerability information

- an online Trustworthiness Assessment of all registered Software Modules in order to assess the modules' integrity

- an online computation of the aforementioned security score for constrained IoT devices

- an online Vulnerability Management for all registered Software Modules with individual risk assessment on a per-device level and warnings and recommended user actions derived therefrom
- a basic webbrowser-based Graphical User Interface (GUI) that presents an overview of all connected devices, their security scores, a more detailed security report, and allows access to individual per-device risk assessment

The remainder of this thesis is structured as follows.

Chapter 2 provides a thorough investigation and detailed presentation of the related work and necessary background. Since this thesis provides a concept for the very first attempt of combining Vulnerability Assessment aspects with an assessment of the device's trustworthiness, a security score that can be interpreted easily by an inexperienced user, and Vulnerability Management functionality, this analysis of the current state of research lays the necessary foundation and has a large share in this thesis.

Chapter 3 provides the specification of a novel Runtime Security Score that serves as the basis and combines the findings from Section 2.4 with the possibilities offered by Hardware Root of Trust security architectures (see Section 2.5) and Trustworthiness Assessment through Remote Attestation (see Section 2.6). This is a novel approach that enables the user of a smart home network to continuously monitor the security state of operated devices.

Chapter 4 transfers the ideas of this Runtime Security Score in an actual software system architecture for a holistic Vulnerability Management Solution for constrained IoT devices. It provides a system model definition that incorporates the modular components and entities of the concept. The well-defined interfaces provide interchangeability and extensibility of the components.

In addition to this specification of the underlying architecture, Chapter 5 provides a detailed description of a Proof of Concept (PoC) implementation that shows the feasibility and functionality of the proposed approach and the cooperating components.

Chapter 6 provides a qualitative evaluation of the proposed software system architecture and the reference implementation. The added value of this concept with regard to the resulting overhead is discussed here.

Chapter 7 concludes this thesis and provides a detailed presentation of the possibilities of future extension of the proposed software system architecture. Since this thesis represents the first work in this new research area, its main contribution is the thoroughly designed architecture itself. Due to the carefully specified interfaces and the modular design, it is very well extendable and lays the foundation for future work in many directions. A comprehensive presentation of those areas for future work is provided with this chapter as well and gives an outlook for the upcoming years of research.

Chapter 2

Background and Related Work

Among the main contributions of this thesis is a holistic architectural concept for a Vulnerability Assessment and Management solution for IoT devices with a Hardware Root of Trust. Due to the scale and complexity of this approach, many large research areas are touched upon. This requires extensive study of related work in the different research areas, that therefore lays the foundation for further design decisions.

The main driver of the research that led to this thesis is the idea of presenting security information in a comprehensible way to the user of an IoT device in a smart home environment. Approaches for pre-purchase combined security and privacy labels exist and are discussed in Section 2.1. However, the proposed label designs all focus on pre-purchase information, whereas the aim of the work presented in this thesis is to assess the security state of devices in operation.

The abstract objective of representing device security during runtime then leads to Vulnerability Assessment and Vulnerability Management, since disclosed vulnerabilities of a device may have a decisive impact on the device's security state. However, assessing the vulnerability of a device requires an understanding of and concept for representing the software components present on the device under consideration. Approaches for appropriate file formats of such software inventory information and for the retrieving procedure exist and are discussed in Section 2.2.

Approaches for Vulnerability Management solutions can then be divided into scanning-based solutions and database-based solutions. Representatives for both classes are discussed in Section 2.3.

When the results of such Vulnerability Assessment are then combined with further risk assessment, a numerical security score can be derived from the gathered information. A selection of appropriate IoT security scoring systems is discussed in Section 2.4.

Trustworthiness Assessment is a very interesting enhancement of such security scoring systems. However, this requires a concept of trust and the presence of Trusted Execution Environments (TEE). In the IoT domain, security architectures that provide TEEs through a Hardware Root of Trust are rare. Influential representatives are discussed in Section 2.5.

A means for assessing the trustworthiness of software components is Remote Attestation (RA). An attempt for generalizing RA procedures is proposed by the IETF and discussed in Section 2.6.

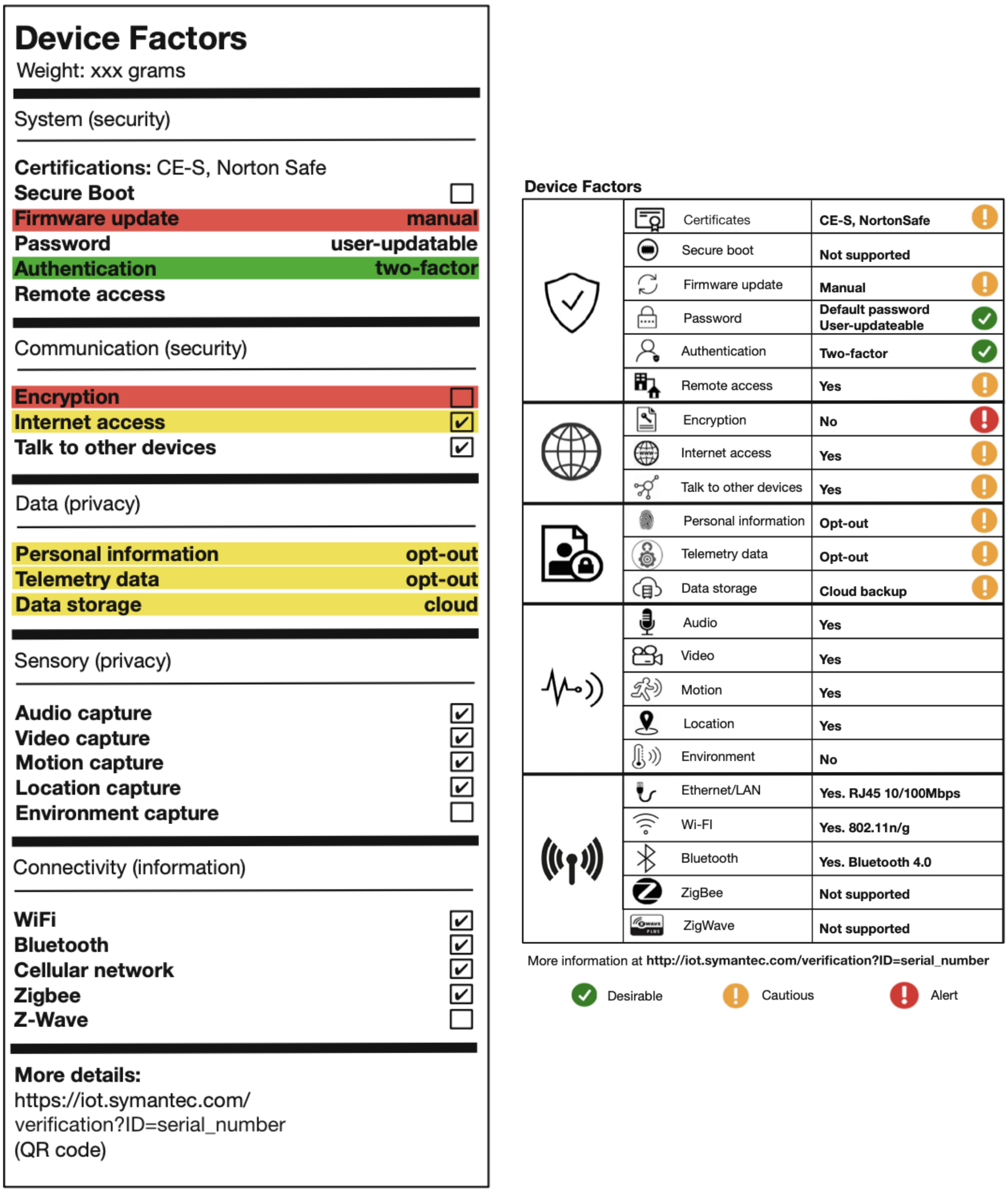
With this extensive literature review, the foundation is provided for the development of an enhanced runtime security score, presented in Chapter 3, and the architectural design of a comprehensive Vulnerability Management Solution for the IoT, presented in Chapter 4.

2.1 Security Labels

Informative consumer labels are widely used in many areas: there exist the nutrition facts label [66], the fuel economy and environment label [65], and the EU energy labels for many classes of electrical devices [19], just to name a few. Analogously, informative pre-purchase labels that represent an IoT device's security or privacy state is subject of ongoing research.

Whereas other efforts in this research area mainly focus on privacy labels [29, 30], Shen et al. [54] and Emami-Naeini et al. [18] both propose a hypothetical design for combined security and privacy labels. Since this thesis focusses on providing the user with valuable information about the security state of the operated smart home network, this section provides an overview of the approaches for security labels proposed by Shen et al. and Emami-Naeini et al.

In [54], Shen et al. propose a prototype label design based on the authors' expert knowledge. An exploratory study with customers or experts has, to the best of our knowledge, not yet been conducted. Opposed to that, Emami-Naeini et al. follow a strongly interview-driven approach. Their first prototype label has been published in 2019 [18] and was followed by an enhanced label design based on expert and consumer interviews in 2020 [17]. Their research efforts result in a detailed specification of a label design that coincides with the proposed prototype labels.



Device Factors
Weight: xxx grams

System (security)

Certifications: CE-S, Norton Safe
Secure Boot □
Firmware update **manual**
Password **user-updatable**
Authentication **two-factor**
Remote access

Communication (security)

Encryption □
Internet access ☑
Talk to other devices ☑

Data (privacy)

Personal information **opt-out**
Telemetry data **opt-out**
Data storage **cloud**

Sensory (privacy)

Audio capture ☑
Video capture ☑
Motion capture ☑
Location capture ☑
Environment capture □

Connectivity (information)

WiFi ☑
Bluetooth ☑
Cellular network ☑
Zigbee ☑
Z-Wave □

More details:
https://iot.symantec.com/
verification?ID=serial_number
(QR code)

Device Factors

Certificates	**CE-S, NortonSafe**	⚠
Secure boot	**Not supported**	
Firmware update	**Manual**	⚠
Password	**Default password** **User-updateable**	✓
Authentication	**Two-factor**	✓
Remote access	**Yes**	⚠
Encryption	**No**	❗
Internet access	**Yes**	⚠
Talk to other devices	**Yes**	⚠
Personal information	**Opt-out**	⚠
Telemetry data	**Opt-out**	⚠
Data storage	**Cloud backup**	⚠
Audio	**Yes**	
Video	**Yes**	
Motion	**Yes**	
Location	**Yes**	
Environment	**No**	
Ethernet/LAN	**Yes. RJ45 10/100Mbps**	
Wi-FI	**Yes. 802.11n/g**	
Bluetooth	**Yes. Bluetooth 4.0**	
ZigBee	**Not supported**	
ZigWave	**Not supported**	

More information at **http://iot.symantec.com/verification?ID=serial_number**

✓ Desirable ⚠ Cautious ❗ Alert

Figure 2.1: Visual layout 1 (a) and 2 (b) for displaying the IoT facts explained in 2.1, taken from [54]

According to Shen et al., their label design provides both concise and completely "independent quality metrics", so called "IoT facts" [54]. They are divided into security metrics, privacy metrics and plain information. The security metrics consist of system-related, and communication-related facts.

The system-related security metrics are supposed to "improve the consumers' security awareness of any IoT product". These metrics include:

- a list of certificates that have been granted to this IoT device
- a checkbox that indicates whether the product has a secure boot mechanism
- brief descriptions of the provided firmware update, authentication and remote access mechanisms and of the password requirements and its updateability

The communication-related security metrics are supposed to "preserve users' privacy and devices' security". These metrics include checkboxes that indicate:

- whether all network communication is encrypted,
- whether Internet access is mandatory for the device to be fully functional,
- and whether the device aims for communication with other devices in the local network

The privacy metrics consist of data-related and sensory-related facts. The data-related privacy metrics are intended to provide the user with an overview and control over the use of their personal data. These metrics include brief descriptions of the types of:

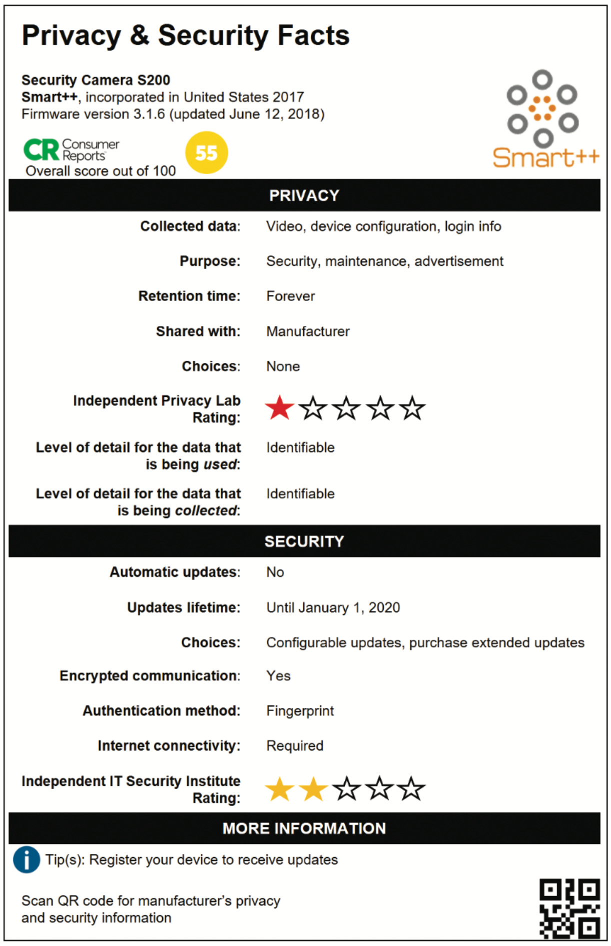
- collected personal data,
- collected telemetry data,
- and storage

The sensory-related privacy metrics are intended to give the user full information about the sensors the IoT device is equipped with. These metrics include checkboxes for:

- audio,
- video,
- motion,
- geolocation,
- and any other sensors

Plain information is given about the device connectivity, hence, the communication protocols used by the IoT device. The protocols considered here are Ethernet/LAN, Wi-Fi, Bluetooth, Zigbee, and Z-Wave.

Further information on the metrics is not given in the referenced publication. The two proposed layouts to display these metrics are presented in Figure 2.1. A consumer study to investigate the practical relevance of the developed label design is announced as future work that has, to the best of our knowledge, not yet been published.



Privacy & Security Facts

Security Camera S200
Smart++, incorporated in United States 2017
Firmware version 3.1.6 (updated June 12, 2018)

CR Consumer Reports 55
Overall score out of 100

Smart++

PRIVACY

Collected data:	Video, device configuration, login info
Purpose:	Security, maintenance, advertisement
Retention time:	Forever
Shared with:	Manufacturer
Choices:	None
Independent Privacy Lab Rating:	★☆☆☆☆
Level of detail for the data that is being *used*:	Identifiable
Level of detail for the data that is being *collected*:	Identifiable

SECURITY

Automatic updates:	No
Updates lifetime:	Until January 1, 2020
Choices:	Configurable updates, purchase extended updates
Encrypted communication:	Yes
Authentication method:	Fingerprint
Internet connectivity:	Required
Independent IT Security Institute Rating:	★★☆☆☆

MORE INFORMATION

ℹ Tip(s): Register your device to receive updates

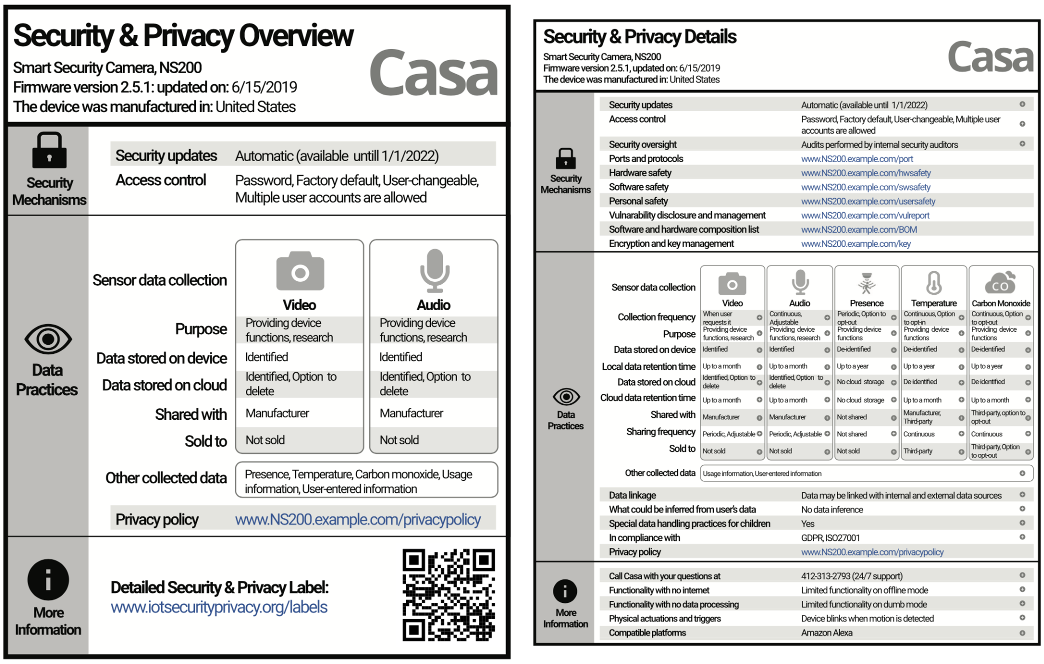
Scan QR code for manufacturer's privacy and security information

Figure 2.2: Exemplary prototype label for a hypothetical security camera with poor privacy and security practices, taken from [18]

A strongly interview-driven approach for the design of a privacy and security label has been conducted by Emami-Naeini et al. In [18], Emami-Naeini et al. propose the prototype of a privacy and security label and present the results of both an in-depth and a follow-up study. A link to the advertised "online appendix" is not apparent directly in the publication. Nevertheless, it can be assumed that the "Specification for CMU [Carnegie Mellon University] IoT Security and

Privacy Label" by Emami-Naeini et al. [16] summarized below includes the relevant information concerning the details of their label design.

To develop the prototype label design presented in Figure 2.2, Emami-Naeini et al. conducted a user study with in-depth interviews with 24 participants. The proposed label design was thereby rated as valuable and worth to be incorporated in the users' purchase decision. In order to estimate the influence of privacy and security information, the authors conducted an additional 200-participant Mechanical Turk survey. The objective of both studies was to gain information about users' pre- and post-purchase behavior and their general concern for privacy and security issues. The results show that most participants consider privacy and security information important before or after the purchase and said they would be willing to pay a premium for the availability of such information at purchase time (see [18] for further details on the study results). An interesting finding is that, according to their in-depth analysis, most participants would be interested in label information about publicly reported vulnerabilities.



Security & Privacy Overview

Smart Security Camera, NS200
Firmware version 2.5.1: updated on: 6/15/2019
The device was manufactured in: United States

Casa

Security Mechanisms

Security updates	Automatic (available untill 1/1/2022)
Access control	Password, Factory default, User-changeable, Multiple user accounts are allowed

Data Practices

Sensor data collection	Video	Audio
Purpose	Providing device functions, research	Providing device functions, research
Data stored on device	Identified	Identified
Data stored on cloud	Identified, Option to delete	Identified, Option to delete
Shared with	Manufacturer	Manufacturer
Sold to	Not sold	Not sold

Other collected data	Presence, Temperature, Carbon monoxide, Usage information, User-entered information
Privacy policy	www.NS200.example.com/privacypolicy

More Information

Detailed Security & Privacy Label:
www.iotsecurityprivacy.org/labels

Security & Privacy Details

Smart Security Camera, NS200
Firmware version 2.5.1, updated on: 6/15/2019
The device was manufactured in: United States

Casa

Security Mechanisms

Security updates	Automatic (available until 1/1/2022)
Access control	Password, Factory default, User-changeable, Multiple user accounts are allowed
Security oversight	Audits performed by internal security auditors
Ports and protocols	www.NS200.example.com/port
Hardware safety	www.NS200.example.com/hwsafety
Software safety	www.NS200.example.com/swsafety
Personal safety	www.NS200.example.com/usersafety
Vulnerability disclosure and management	www.NS200.example.com/vulreport
Software and hardware composition list	www.NS200.example.com/BOM
Encryption and key management	www.NS200.example.com/key

Data Practices

Sensor data collection	Video	Audio	Presence	Temperature	Carbon Monoxide
Collection frequency	When user requests it	Continuous, Adjustable	Periodic, Option to opt-out	Continuous, Option to opt-in	Continuous, Option to opt-out
Purpose	Providing device functions, research	Providing device functions, research	Providing device functions	Providing device functions	Providing device functions
Data stored on device	Identified	Identified	De-identified	De-identified	De-identified
Local data retention time	Up to a month	Up to a month	Up to a year	Up to a year	Up to a year
Data stored on cloud	Identified, Option to delete	Identified, Option to delete	No cloud storage	De-identified	De-identified
Cloud data retention time	Up to a month	Up to a month	No cloud storage	Up to a month	Up to a month
Shared with	Manufacturer	Manufacturer	Not shared	Manufacturer, Third-party	Third-party, option to opt-out
Sharing frequency	Periodic, Adjustable	Periodic, Adjustable	Not shared	Continuous	Continuous
Sold to	Not sold	Not sold	Not sold	Third-party	Third-party, Option to opt-out

Other collected data	Usage information, User-entered information
Data linkage	Data may be linked with internal and external data sources
What could be inferred from user's data	No data inference
Special data handling practices for children	Yes
In compliance with	GDPR, ISO27001
Privacy policy	www.NS200.example.com/privacypolicy

More Information

Call Casa with your questions at	412-313-2793 (24/7 support)
Functionality with no internet	Limited functionality on offline mode
Functionality with no data processing	Limited functionality on dumb mode
Physical actuations and triggers	Device blinks when motion is detected
Compatible platforms	Amazon Alexa

Figure 2.3: Visual layout for displaying the primary layer (a) and secondary layer (b), taken from [17]

In a following publication in 2020, Emami-Naeini et al. state that "Other proposals for IoT privacy and security labels fail to specify the specific information that consumers should be presented with on the label." [17]. To fill this gap, Emami-Naeini et al. conducted interviews with 22 privacy and security experts from academia, industry, government and NGOs. Based on this elicitation study, they propose another layered prototype label (see Figure 2.3) that has been presented to 15 consumers in semi-structured interviews. In a layered label approach, the primary layer is supposed to be printed on the device packaging or advertised at the shelf in store. A link or QR-code in the primary layer then refers the customer to a secondary layer found online. Since this secondary layer resides on a homepage that can be updated, it may contain information that is more likely to change over time.

Among the factors that a majority of the expert participants recommended are a star rating to help customers to easily asses and compare the security and privacy state of the considered device, and a security rating ”from an independent security assessment organization”. But since no such rating formula or rating organization exist at the moment, these factors have not been included in the actual primary layer label illustrated above.

Factors that, according to the experts, should be excluded completely from the label were e.g. a software and hardware bill of material, because of ”the lack of relevance to privacy and security and inability to convey risk to consumers”, and information on whether the device manufacturer has a vulnerability disclosure and management program. Nevertheless, Emami-Naeini et al. decided to include both on their prototype label on the secondary layer and received supportive feedback during the consumer survey.

In addition to the aforementioned publications, Emami-Naeini et al. propose a Label Specification that meets the requirements of the prototype labels described above [16]. Selected security factors are briefly summarised below. Possible values for each factor can be one or more of the indicated options.

- Security Updates on primary and secondary layer
  possible values: automatic, manual, consent-based, no security updates
  optional attribute: expiration date
  optional information: description of how updates are secured
- Access Control on primary and secondary layer
  possible values: password, biometric, multi-factor authentication, no control over access, multiple user accounts, single required user account, optional user account, no user accounts
  optional password attributes: factory-default (user-changeable or not changeable by user), user-generated
- Security Oversight on secondary layer
  possible values: internal audit, external audit, internal and external audits, no audit

optional information: findings of the audits, what the manufacturer will do with the findings of the audits

- Ports and Protocols on secondary layer
possible values: link to list and justification of all physical interfaces, network ports, listening services
optional link information: safeguards for each interface to prevent misuse, Manufacturer Usage Description (MUD) file describing device's normal network behaviour

- Hardware Safety on secondary layer
possible values: link to list of safeguards to protect the device's hardware

- Software Safety on secondary layer
possible values: link to list of safeguards to protect the device's software
optional information: types of risks introduced by libraries, the code complexity, code coverage, number and types of crashes

- Vulnerability Disclosure and Management on secondary layer
possible values: link to description of "How transparent and timely the manufacturer has been in disclosing the discovered vulnerabilities, managing them, and mitigating their potential harms" [16].

- Software and Hardware Composition List on secondary layer
possible values: link to list of device's software and hardware components

Additional to the specification of each label field, best practices are given, supposedly addressed to device manufacturers. Recommended best practices for the device's hardware safety are an irrevocable secure boot process and a hardware-based Root of Trust for updates. Best practices for the device's software safety include that executables of critical applications shall be stored in read-only memory section, that software shall not be "overly complex" and "not susceptible to crashes" [16]. Additionally, "system software should be tested to check for publicly disclosed and undisclosed vulnerabilities" and "should not use unsafe libraries". Concerning Encryption and Key Management, the use of a Trusted Execution Environment (TEE) is recommended to be used for key storage and encryption.

Starting with the first prototype privacy and security label, Emami-Naeini conducted expert and consumer interviews to come up with the privacy and security label specification described above.

The impact of the aforementioned label design on the consumers' perception of risk has also been examined by Emami-Naeini et al. in a recent Mechanical Turk survey with 1,371 participants [37]. However, further insight into the consumers' risk perception and willingness to pay makes up its own area of research and is considered out of scope for this work.

Although Emami-Naeini et al. acknowledge the relevance of post-purchase security information for the consumer, an online security label that is updated and displayed at runtime has not yet been the subject of research. Additionally, in their label specification in [16], the use of TEEs is mentioned as best practise, but the label design itself does not focus on hardware security architectures that provide a TEE. Based on the findings for pre-purchase label design discussed above and on the findings for security scoring systems in the IoT domain discussed in Section 2.4, the prototype of a security report is included in the Vulnerability Assessment and Management solution proposed by this thesis and described in Chapter 4.

2.2 Software Inventory Information

Vulnerability Assessment solutions that are not based on approaches for discovering vulnerabilities by analysing the system behaviour, rely on a check against a database of disclosed vulnerabilities. This requires a definition of the software fragment possessing the vulnerability. Then, this software transparency lays the crucial foundation for all further software vulnerability handling. Only if all software running on the device under consideration is well-known, its vulnerabilities can be properly assessed and managed. Therefore, main building blocks for Vulnerability Assessment are a universal identifier for Software Modules and models for its description as Software Bills of Material (SBOM). Similarly, an understanding for partitioning software into Software Modules and unambiguously identifying them is a decisive building block for any remote attestation approach described in Chapter 2.6. In the following chapter, a detailed overview of major proposals is given.

Apart from Vulnerability Assessment, SBOMs also play an important role concerning software licensing aspects. Nevertheless, software licensing is considered out of scope for this thesis. Additionally, discovery tools that automatically detect and collect the Software Modules installed on a device without a predefined description are not the focus of this thesis.

Beforehand, the relevant terminology has to be clarified, since lots of terms exist in this research area. A "Software Module" is a fragment of software concluded by the Software Provider. The terms "Software Package" and "Software Component" are considered synonyms.
Then, "Software Inventory Information" describe all Software Modules deployed on the IoT device. The term "Software Bill of Material" is considered a synonym.

Especially in the IoT domain, heterogeneous networks with devices from various manufacturers with different software platforms and different communication protocols are prevalent. Therefore, it is a common use case that a heterogeneous fleet of devices and software platforms is required to interoperate neatly in order to provide the best user experience and in order not to miss disclosed vulnerabilities [28]. SBOMs act as important interfaces between device manufacturers, software providers, vulnerability databases and Vulnerability Assessment solutions. Parallel standardization efforts for a common format to describe an SBOM are ongoing work of several initiatives and since standardization is of such great importance in this research area, those are in the focus of this thesis and individual proposals are omitted. Note, though, that e.g. Emami-Naeini et al. propose a "list of software and hardware components that are used in the device" (see [16]) without giving further details in their "Specification for CMU IoT Security and Privacy Label" discussed in Chapter 2.1.

Therefore, the requirements for an appropriate software inventory information format in the context of this thesis can be summarized as:

1. it enables a partitioning of software into Software Modules
2. it provides a universal unique identifier for each and every Software Module in that very version
3. it provides a model and a format for its description as SBOM

The ISO standard 19770-2 proposes Software Identification (SWID) tags as a software module identifier format [28]. Based on SWID, the IETF is actively working on a draft for a Concise Software Identification Tag (CoSWID) [6]. The Linux Foundation proposes the Software Package Data Exchange format [59]. All of these approaches are described in detail in the following Section 2.2.1.

An overview of selected fields of all software inventory information formats discussed here is presented in Table 2.1. Where applicable, similar fields are grouped in one row.

In addition to file formats for software inventory information, there exist proposals for a standardization of the SBOM file retrieval process as well. The Internet Engineering Task Force (IETF) is actively working on a draft for discovering and retrieving an SBOM [34] based on the IETF RFC 8520 for a Manufacturer Usage Description (MUD) [33]. Both approaches are described in detail in the following Section 2.2.2.

ISO SWID [28]	**IETF CoSWID [integer index] text label** [6]	**SPDX** [59]
SoftwareIdentity.name	**[1] concise-swid-tag.software-name**	**PackageName**
SoftwareIdentity.tagId	**[0] concise-swid-tag.tag-id**	**SPDXID**
SoftwareIdentity.version	[13] concise-swid-tag.software-version	PackageVersion
SoftwareIdentity.corpus	[8] concise-swid-tag.corpus	-
for tagCreator: **Entity.role**	**[33] concise-swid-tag.entity-entry.role**	-
for tagCreator: **Entity.regid**	[32] concise-swid-tag.entity-entry.reg-id	-
for tagCreator: **Entity.name**	**[31] concise-swid-tag.entity-entry.entity-name**	**Creator**
Payload.File.name	[17] concise-swid-tag.payload.file.filesystem-item	FileName
Payload.File.size	[20] concise-swid-tag.payload.file.size	-
Payload.File.hash	[7] concise-swid-tag.payload.file.hash	FileChecksum

Table 2.1: Overview of selected fields and their identifiers of the described software inventory information formats. Mandatory fields are written in bold letters. Where applicable, similar fields are grouped in one row.

2.2.1 Software Inventory Information File Formats

The International Organization for Standardization (ISO) proposes a format to consistently identify software modules in Part 2 ”Software Identification Tag” of their standard 19770-2 on ”Information technology - Software asset management” [28]. Thereby, it aims at providing interoperability of software management data and enabling automated authentication of software modules and linking to vulnerability databases.

The proposed description is called SWID tag and is created by the SWID tag producer. This could e.g. be a Software Provider. This SWID tag is then used by the SWID tag consumer, e.g. an IT discovery and processing tool. An SBOM of a device could then consist of the SWID tags of all software modules deployed on the device.

The file format for the SWID tag file is required to be an XML data structure. The corresponding defect-corrected XML schema definition is made available here: `http://standards.iso.org/iso/19770/-2/2015-current/schema.xsd`.

Selected fields of the SWID tag file specified in this ISO standard are presented in Table 2.1 in column 1.

- The *SoftwareIdentity.name* contains the name of the software module as it would e.g. be used in the package manager on a Linux machine.
- The *SoftwareIdentity.tagId* is a globally unique ID and serves as unique and unambiguous reference for this software module in this very version. It is recommended to compose the *SoftwareIdentity.tagId* as follows: ”$Entity.regid + productName + SoftwareIdentity.version + edition$ $+revision+\ldots$”. There is no central registration agency for assigning unique *SoftwareIdentity.tagId*s. Instead, the tag producers have to take responsibility for their uniqueness.
- The *SoftwareIdentity.version* describes the software development version as assigned by the Software Provider.
- The *SoftwareIdentity.corpus* is a boolean that signals if this SWID tag contains information concerning pre-installation data of the Software Module.
- For the entity creating the SWID tag, the tag has to contain at least one *Entity*-block with the attribute *Entity.role* set to *tagCreator* and the attribute *Entity.name* set to the organization’s name.
- The *Entity.regid* is a unique registration ID for this entity. In order to achieve uniqueness, it is recommended to base this ID on an absolute-URI

reference in http-scheme starting with a domain name such as "http://www.example.com".

- The *Payload*-block indicates the files that are used for the installation process itself (e.g. an installer.exe) and the files that are deployed on the device during installation.
- Accordingly, *Payload.File.name* and *Payload.File.size* contain the name and size of one such file. For authorization reasons, a hash of the file (e.g. SHA256) can be included using the *Payload.File.hash*.

An example for a minimal XML file for the presented SWID tag specification can be found in Appendix 7.

As opposed to specifications such as CoSWID or MUD (see below), the SWID tag specification is not dedicated to IoT devices. Hence, the recommended transmission of the SWID tag file is based on an API or on a well-defined location in the file system where it is stored and from where it can be retrieved. For devices without a file system, which applies to constrained IoT devices , the specification is less precise. Nevertheless, the SWID tag file is supposed to be stored locally on the device itself and either be accessible through its pre-defined location in memory or through a pre-defined URI. Both the storage location and the file format are to be defined by the platform provider, which is the entity responsible for the platform, hence in the context of this thesis, the device manufacturer. In this case, the file format may deviate from XML.

Among the key benefits claimed for the proposed SWID tag specification are the following (see[28], p.vi-vii):

- "The ability to consistently and authoritatively identify software products that need to be managed for any purpose, such as for licensing, security, logistics, or for the specification of dependencies. Software identification tags provide the meta-data necessary to support more accurate identification than other software identification techniques. [...]
- The ability to automatically relate installed software with other information such as patch installations, configuration issues, or other vulnerabilities.
- Facilitate interoperability of software information between different software creators, different software platforms, different IT management tools, and within software creator organizations, as well as between SWID tag producers and SWID tag consumers.
- Through the optional use of digital signatures by organizations creating software identification tags, the ability to validate that information is authoritative and has not been maliciously tampered with.

- The opportunity for entities other than original software creators (e.g. independent providers or in-house personnel) to create software identification tags for legacy software, and for software from software creators who do not provide software identification tags themselves."

Based on the ISO SWID tag format described above, the IETF proposes a concise representation of SWID tags targeting constrained IoT devices as described by the IETF in RFC 7228 [10]: the Concise Software Identification (CoSWID) tags [6]. This draft is declared as work in progress and hence, it may be subject to change or removal. All information given here are based on the draft version from March 7, 2022.

As highlighted in Table 2.1, the number of mandatory fields in SWID tags is relatively low. Nevertheless, there can be many optional fields included in the SBOM file that result in a file size that is too large to be handled by constrained IoT devices. Hence, a Concise Binary Object Representation (CBOR) [11] is proposed to reduce the size of the SBOM file. By proposing a more dense scaffolding with the CoSWID representation, a reduction of file size of 50 to 85 percent compared to SWID tags in XML format is claimed to be achievable for generic usage scenarios without changing the actual values stored in the tag. However, for CoSWID tags, the resulting file is not human-readable anymore due to the binary object representation.

Selected fields of the CoSWID tag file specified in this draft are presented in Table 2.1 in column 2. The underlying implicit information model is supposed to be identical between SWID tags and CoSWID tags. Therefore, mappings for all mandatory and optional fields in the ISO/IEC 19770-2:2015 standard and the CoSWID specification in the referenced version are provided. Note, that "future revisions of ISO/IEC 19770-2:2015 or this specification might cause this implicit information model to diverge, since these specifications are maintained by different standards groups" (see [6], p. 3). The textual labels in the CoSWID tag are closely-coupled to their SWID tag analogue. In most cases, the naming convention has only been changed from the CamelCase notation to the hyphen- separated KebabCase notation. Therefore, it is referred to the description of all fields in the SWID tag model given above.

The last software inventory information format presented here is the open source Software Package Data Exchange (SPDX) format in Version 2.2 proposed by the Linux Foundation [59] and listed as appropriate SBOM format in the IETF Internet-Draft "Discovering and Retrieving Software Transparency and Vulnerability Information" described below. It is advertised as standard format for communicating software inventory information. The term "software package" used in the standard specification coincides to the term "Software Module" used throughout this thesis.

In contrast to the CoSWID tag format described above, the SPDX format is required to be human readable. The underlying implicit information model can be represented in several data formats, e.g. YAML, JSON or as flat text file using $tag : value$ notation. Support of XML format is currently work in progress and expected for Version 2.3.

Selected fields of the SPDX file are presented in Table 2.1 in column 3. For better readability, the $tag : value$ notation is used.

In addition to the specification of fields in the SPDX file, a recommendation for the composition of a URI to access the SPDX file on the public Internet is also given as $http : //[CreatorWebsite]/[pathToSpdx]/[DocumentName] - [UUID]$. Note, however, that although a scheme (e.g. http) has to be provided, this URI does not necessarily have to be accessible as Internet resource. Its purpose can also be to provide a unique URI for identifying the SPDX file. Hence, it is also used as $SPDXID$.

- $CreatorWebsite$ refers to the domain of the entity providing the SPDX file.
- $pathToSPDX$ contains the path to the SPDX file.
- $DocumentName$ contains the name of the SPDX file, which is recommended to be constituted from the Software Module's name and its version number.
- $UUID$ contains a Universally Unique Identifier (UUID) that is unique for this specific SPDX file version.

An example for a SPDX file can be found in Section 4.2.1.

Similar to the SWID tag format described above, the SPDX format is not focussed on IoT devices. Nevertheless, in addition to the regular SPDX format, a reduced file format called "SPDX Lite" is specified as well. The SPDX Lite format contains a subset of the regular SPDX fields with more basic information.

The SPDX Lite format contains all of the fields presented in Table 2.1 except for the $FileName$ and the $FileChecksum$. However, the less fine-grained field $PackageFileName$ is included to refer to files that have been aggregated into a package.

2.2.2 Retrieving Software Inventory Information

The Internet-Draft "Discovering and Retrieving Software Transparency and Vulnerability Information" published by the Internet Engineering Task Force (IETF) ([34]) proposes a model to access software inventory information from an IoT device. This draft is declared as work in progress and hence, it may be subject to change or removal. All information given here are based on the draft version from October 13, 2020. Although the draft's abstract does not advertise it as being dedicated to the IoT domain, the depicted use cases target IoT devices. Hence, it is assumed that this draft is dedicated to the IoT world.

This draft does not propose a new SBOM format, but refers to the aforementioned SWID tags and the Software Package Data Exchange (SPDX) format described above. In extension to those SBOM formats, this IETF Internet-Draft elaborates on possible approaches for transmitting SBOM files and proposes a data model for transmitting the SBOM URI.

According to this draft, an SBOM can be provided in three different ways: directly by the device, at a dedicated URI on a website or through another dedicated communication channel with the vendor (e.g. by sending an e-mail to the vendor). In the first case, the device provides an interface to retrieve the SBOM file directly. HTTP, CoAP, and any proprietary protocol are named as possible communication protocols, but there is no further more precise specification of the necessary mechanisms and messages given in the draft.

In the second case, the device only provides an interface for retrieving the URI for the SBOM file hosted on a Manufacturer webserver. In order to transmit this URI, a model extension for a MUD file is proposed. This so-called "mud-sbom extension" is realized as YANG augmentation of the MUD YANG model and is described below.

In the last case, the vendor is able to add conditions to the dispatching of the SBOM (e.g. a registration or authentication) or to monitor and evaluate SBOM accesses. Moreover, another communication channel could be established by making the SBOM URI available in the device documentation or in a QR-code printed on the device packaging. Of course, this process could not be fully automated then.

As described above, the Manufacturer Usage Description (MUD) published by the IETF as RFC 8520 [33] serves as a basis for the SBOM file retrieval model proposed in the IETF Draft "Discovering and Retrieving Software Transparency and Vulnerability Information". This RFC focusses on "Things" that, in contrast to other general purpose machines, have only a small number of very dedicated purposes or uses. The term "Things" coincides to the term "IoT device" used throughout this thesis. The term "Manufacturer" here refers to the entity in the device's supply chain that takes responsibility for informing the network that integrates the device about its purpose. Hence, it is not necessarily the entity that constructs the Thing, but could be a system integrator or a component provider. The purpose of a MUD file is to inform the network about the intended network functionality and access limitations of the device.

The device manufacturer is responsible for configuring the IoT device in such a way that it emits the MUD URL in https-scheme after being requested by the MUD manager via GET-method [20]. Based on the MUD URL, the MUD manager requests the MUD file from the corresponding MUD file server, a simple web server that hosts the MUD file. The MUD file itself contains a description of the Thing in YANG-based JSON and associated suggested specific network behaviour.

```
.......................................
.                      ____________   .           _____________
.                     |            |  .          |             |
.                     |    MUD     |-->get URL-->|   MUD       |
.                     | Manager    |  .(https)   | File Server |
.  End system network |____________|<-MUD file<-<|_____________|
.                             .       .
.                             .       .
. _______                 _________   .
.|       | (DHCP et al.) | router  |  .
.| Thing |---->MUD URL-->|   or    |  .
.|_______|               | switch  |  .
.                        |_________|  .
.......................................
```

Figure 2.4: MUD Architecture taken from [33]

In [33], different means for emitting the MUD URL are proposed. A general overview of the MUD architecture is given in Figure 2.4.

The MUD URL can be included as additional option in the *Dynamic Host Configuration Protocol (DHCP)* so that the DHCP client (here: the IoT device) informs the DHCP server (here: the MUD manager) about the MUD URL. Alternatively, the MUD URL can serve as a non-critical extension to the IEEE standard 802.1AR (the "IEEE Standard for Local and metropolitan area networks - Secure Device Identity" [27]), or it can be emitted as a Link Layer Discovery Protocol (LLDP) frame. In case the IoT device has too limited capabilities to emit the MUD URL, it may be possible for the manufacturer to map the device's identity (e.g. according to its serial number or public key) to the MUD URL or the MUD file using an alternative resolution mechanism.

As soon as the MUD URL has been discovered, it is forwarded to the MUD Manager (a network management system) for further processing. Then, a HTTP-GET-request is sent to the MUD File Server that answers with the MUD file. If the MUD Manager is not able to retrieve the MUD file, an appropriate timeout should be awaited before the failure is logged.

By default, the MUD file does not expire. Nevertheless, its description can be used for communicating an expiration date through the cache-validity node [34].

According to the current RFC version of March 2019, the MUD file format is the YANG Data Modeling Language [8], serialized using JSON [12]. An example MUD file using the mud-sbom extension can be found in Listing 2.1. Selected fields of the mud-sbom extension are described based on this example. The full mud-sbom augmentation of the MUD YANG model may be found in Appendix 7.

Listing 2.1: An example MUD file using the mud-sbom extension, taken from [34]

```
{
    "ietf-mud:mud": {
      "mud-version": 1,
      "mud-url": "https://iot-device.example.com/dnsname",
      "last-update": "2019-01-15T10:22:47+00:00",
      "cache-validity": 48,
      "is-supported": true,
      "systeminfo": "device␣that␣wants␣to␣talk␣to␣a␣cloud␣service",
      "mfg-name": "Example,␣Inc.",
      "documentation": "https://frobinator.example.com/doc/frob2000",
      "model-name": "Frobinator␣2000",
      "extensions" : [
"sbom" ],
"sboms" : [ {
           "version-info" : "FrobOS␣Release␣1.1",
           "sbom-url" : "https://frobinator.example.com/sboms/f20001.1",
        }
] }
}
```

$sboms.version - info$ includes a string with version control information for this software module provided by the Manufacturer.

$sbom - type$ can be one of the following cases. $sboms.sbom - url$ refers to a statically located Internet URI. $sboms.sbom - local$ can either state *coap*, *coaps*, *http* or *https* as the protocol of choice for retrieving the SBOM file from a well-known location on the device. $sboms.contact - uri$ can either state a *tel*, *http*, *https* or *mailto* URI schema in order to contact the Manufacturer for the SBOM file.

The proposed mud-sbom extension can optionally be extended with access lists. Since access considerations are not the focus of this thesis, details are omitted here.

2.2.3 Conclusion

As described in the introduction to this Chapter, the 3 main requirements for software inventory information formats are:

1. they enable a partitioning of software into Software Modules
2. they provide a universal identifier for each and every Software Module in that very version
3. they provide a model and a format for its description as SBOM file

Three proposals for software inventory information formats have been described above. The first requirement is met by all three proposals due to the initial assumption that there is an entity providing the software inventory information for the Software Module the entity is responsible for.

Both the SWID tag specification and the CoSWID specification are developed by internationally operating and recognized standardization organizations. The SPDX specification is proposed by the Linux Foundation, but is referenced in the IETF Internet-Draft for "Discovering and Retrieving Software Transparency and Vulnerability Information" as appropriate software inventory information format. Standardization is key on the way to finding an agreement between device manufacturers, Software Providers, vulnerability databases and Vulnerability Assessment solutions, which makes the discussed proposals very valuable. Of course, none of these candidates has yet prevailed.

The ISO SWID tag specification contains a globally unique *tagId* for every Software Module, hence it is fulfilling requirement (2). The SWID tag file is an XML and can contain one SoftwareIdentity-region for every Software Module, hence it is fulfilling requirement (3). Nevertheless, this specification is not dedicated to IoT devices and does not provide precise recommendations for storing and retrieving the SWID tag file of a constrained IoT device, where the file can not be stored on the device itself. Therefore, it is not the best suited candidate for the IoT domain and hence, for the Vulnerability Management solution proposed by this thesis.

This issue is aimed at with the CoSWID tags proposed by the IETF. Those are based on the same underlying implicit information model as SWID tags, which is why they fulfill the requirements (2) und (3) just as well. But they require less mandatory fields and allow for a resource-efficient binary representation. However, the resulting software inventory information file is not human readable anymore and handling the CBOR format requires additional implementation overhead. Since it is not a main requirement for the Vulnerability Management solution proposed in this thesis to store the software inventory information file directly on the IoT device, the file size is neglectable.

On the other hand, the SPDX format is required to be human readable, allows for several data formats (thereby fulfilling requirement (3)) and does not impose any demands as to where the SPDX file is supposed to be stored or how it is supposed to be retrieved. The specification also provides a recommendation for the composition of a universally unique URI (thereby fulfilling requirement (2)) to access the SPDX file on the public Internet. Thus, the SPDX format is particularly suitable for combining it with the IETF proposal for "Discovering and Retrieving Software Transparency and Vulnerability Information" and is chosen for the Vulnerability Management solution proposed in this thesis.

This IETF Internet-Draft for retrieving software inventory information is suitable for the IoT domain. Since the Vulnerability Management solution proposed in this thesis is supposed to also be suitable for constrained devices, the software inventory information file is not supposed to be stored directly on the device. Therefore, the second proposed mechanism, where the IoT device only provides an interface for retrieving the URL to the Manufacturer webserver, is chosen in combination with the SPDX format for the SBOM file. The third proposed mechanism is neglectable as it only represents an alternative communication channel with the Manufacturer and may not be fully automated. In a future extension, a Manufacturer Usage Description (MUD) file could be used for transmitting the SBOM URL in a unified, standardized way.

2.3 Vulnerability Assessment (VA) and Vulnerability Management (VM)

As described above, using Software Bills of Material to represent software inventory information serves very well as basis for Vulnerability Assessment. But ”generating an SBOM is only one half of the story” [36]. After all software deployed on a device is partitioned into Software Modules that are described with SBOM files, the Software Modules have to be linked against a database of disclosed vulnerabilities in order to find threats the device may be susceptible for. A selection of different Vulnerability Assessment approaches is presented in the following section.

Due to the heterogeneous landscape of devices, applications, vendors, industries, and cultures, there is no strict definition of a ”vulnerability”. For this work, we adhere to the distinction proposed by the Common Vulnerabilities and Exposures (CVE) Program (see [62]):

> ”In general, a vulnerability is defined as a weakness in the computational logic (e.g., code) found in software and hardware components that, when exploited, results in a negative impact to confidentiality, integrity, OR availability. [. . .]
>
> 1. If a product owner considers an issue to be a vulnerability in its product, then the issue MUST be considered a vulnerability, regardless of whether other parties (e.g., other vendors whose products share the affected code) agree.
> 2. If the CNA [CVE Numbering Authority] determines that an issue violates the security policy of a product, then the issue SHOULD be considered a vulnerability.
> 3. If a CNA receives a report about a new vulnerability that has a negative impact, then the reported vulnerability MAY be considered a vulnerability.”

Vulnerability Assessment (VA) solutions are valuable tools to rate a system’s overall security state and prevent vulnerabilities from being exploited. Therefore, VA opens up a vast research field with well-established solutions for general applications. In 2021, Gartner published a Market Guide for Vulnerability Assessment [24], giving an overview of requirements and the most popular industry solutions. According to Gartner and Cybersecurity Insiders [14], the VA vendors with the most significant market shares are Tenable (Nessus Scanner), Qualys (Qualys Vulnerability Management, Detection, and Response), and Rapid7 (InsightVM). A brief discussion of these popular solutions is proposed in Section 2.3.1.

In general, VA can be categorized into approaches that perform scanning techniques (e.g. scanning for connected devices, available IPs, open ports, available services) and approaches that link software inventory information to a vulnerability database. Initially, this thesis focusses on the latter. However, Amro performed a Systematic Literature Review for vulnerability scanning approaches, published in Computer Security in 2020 [3]. With only 10 unique works focussing on IoT vulnerability scanning taken into thorough consideration, the freshness of this research area is highlighted. Hence, combining both techniques is promising for further reducing the attack surface in the future.

By combining VA with vulnerability prioritization, patching functionality, and continuous monitoring, it can be extended to powerful Vulnerability Management (VM) tools. VM solutions allow the user to continuously gain extensive insight into the network with all valuable assets, make predictions, and initiate countermeasures to improve the overall security. However, as of now, commercial solutions mainly focus on larger companies with dedicated IT security teams [58, 47, 50]. Although the IoT domain as a nonstandard IT asset is named in Gartner's market guide, it does not provide further information about which vendors provide VA or VM solutions targeting the IoT sector. Apparently, the largely inexperienced end-user operating commercial IoT devices in a smart home network is not a common target group for such applications. Nevertheless, a smart home end-user may operate IoT devices with critical functionality, such as smart door locks or smart smoke detectors, and would strongly benefit from proper vulnerability management.

For the rest of this thesis, the term "Vulnerability Assessment" includes identifying and collecting vulnerabilities of a device or system under consideration. The term "Vulnerability Management" includes Vulnerability Assessment, but extends it with additional management functionality to counteract or prevent vulnerabilities. This includes, but is not limited to, prioritizing vulnerabilities, deploying software updates or patches that eliminate a disclosed vulnerability after this vulnerability has been discovered to be present on a managed device, and continuous monitoring.

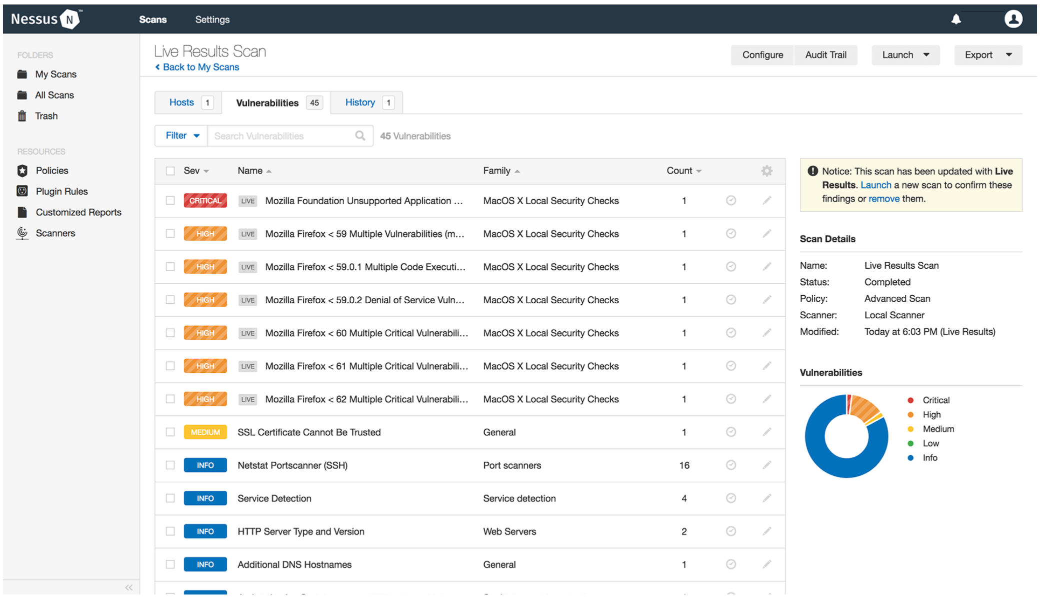
A comparison of the most popular industrial VM solutions and a detailed description of the CVE system are provided in the following sections.

The results of this chapter are concluded in a concept for a VM solution for the IoT described in Chapter 4.

2.3.1 Scanning-based Vulnerability Management Solutions

Nessus is a proprietary industry solution for comprehensive vulnerability assessment for organizations for various platforms including Raspberry Pi [58]. It is advertised to provide ”the deepest and broadest vulnerability coverage in the industry”. To proof that claim, a comparison of Nessus professional with OpenVAS and Rapid7 Nexpose is provided [57, 45]. This comparison covers e.g. CVE coverage (for Nessus professional: 67K CVEs), scanning accuracy (for Nessus professional: ”Industry’s lowest false-positive rate – better than six-sigma accuracy” (99.99966%), and speed of vulnerability check release (for Nessus professional: releases within an average of 24 hours after vulnerability disclosure). However, there is no evidence provided with these claims.

An important feature of Nessus professional are live results (see Figure 2.5 for an exemplary screenshot). In order to achieve real-time visibility, an offline scan for newly published vulnerabilities using historical scan data is performed. Indication exists that scanning for open ports using *nmap* is part of the vulnerability assessment procedure [67] and the claim of very high CVE coverage raises the presumption that linking to a CVE database is also performed. However, further information on how the scanning is performed is not provided. The range of supported Operating Systems (OS) does not include an OS typically used in the IoT domain.



□	Sev	Name	Family	Count
□	CRITICAL	LIVE Mozilla Foundation Unsupported Application ...	MacOS X Local Security Checks	1
□	HIGH	LIVE Mozilla Firefox < 59 Multiple Vulnerabilities (m...	MacOS X Local Security Checks	1
□	HIGH	LIVE Mozilla Firefox < 59.0.1 Multiple Code Executi...	MacOS X Local Security Checks	1
□	HIGH	LIVE Mozilla Firefox < 59.0.2 Denial of Service Vuln...	MacOS X Local Security Checks	1
□	HIGH	LIVE Mozilla Firefox < 60 Multiple Critical Vulnerabili...	MacOS X Local Security Checks	1
□	HIGH	LIVE Mozilla Firefox < 61 Multiple Critical Vulnerabili...	MacOS X Local Security Checks	1
□	HIGH	LIVE Mozilla Firefox < 62 Multiple Critical Vulnerabili...	MacOS X Local Security Checks	1
□	MEDIUM	SSL Certificate Cannot Be Trusted	General	1
□	INFO	Netstat Portscanner (SSH)	Port scanners	16
□	INFO	Service Detection	Service detection	4
□	INFO	HTTP Server Type and Version	Web Servers	2
□	INFO	Additional DNS Hostnames	General	1

Notice: This scan has been updated with **Live Results**. Launch a new scan to confirm these findings or remove them.

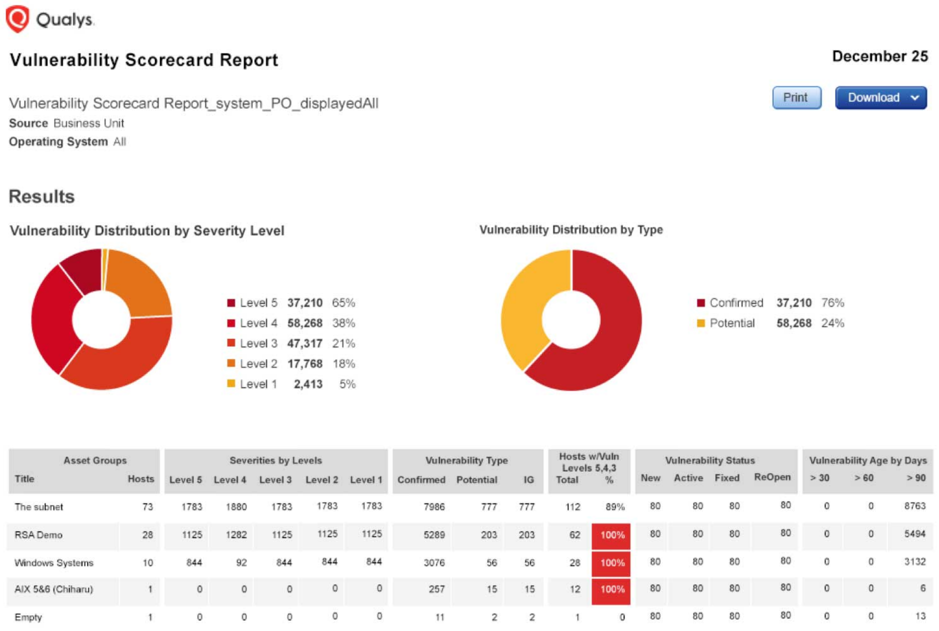
Scan Details

Name:	Live Results Scan
Status:	Completed
Policy:	Advanced Scan
Scanner:	Local Scanner
Modified:	Today at 6:03 PM (Live Results)

Figure 2.5: Screenshot with exemplary Live Results of the Nessus professional scanner, taken from [58]

Due to the lack of support of appropriate OS and business customers being the target group, the Nessus Scanner is not an appropriate choice for an end-user smart home environment.

Qualys Vulnerability Management, Detection, and Response (VMDR) is another proprietary industry solution for vulnerability management that advertises a scanning accuracy of six sigma [47] and integrates CVE and CVSS standards [46]. In addition to Vulnerability Assessment, Qualys VMDR also offers threat prioritization and patch deployment. When combined with Qualys Continuous Monitoring (CM), Vulnerability Assessment during runtime can be performed manually, on schedule, or continuously [48]. In [46], it is suggested to perform vulnerability scans at least daily. Qualys Zero-Day Risk Analyzer provides a prediction of hosts most threatened by zero-day exploits. A grouping and rating of vulnerabilities is also possible (see Figure 2.6). Configurations are compared with Center for Internet Security (CIS)-benchmarks to identify critical misconfiguration.



Vulnerability Scorecard Report

December 25

Vulnerability Scorecard Report_system_PO_displayedAll
Source Business Unit
Operating System All

Results

Vulnerability Distribution by Severity Level

🟥 Level 5	**37,210**	65%
🟥 Level 4	**58,268**	38%
🟥 Level 3	**47,317**	21%
🟧 Level 2	**17,768**	18%
🟨 Level 1	**2,413**	5%

Vulnerability Distribution by Type

🟥 Confirmed	**37,210**	76%
🟨 Potential	**58,268**	24%

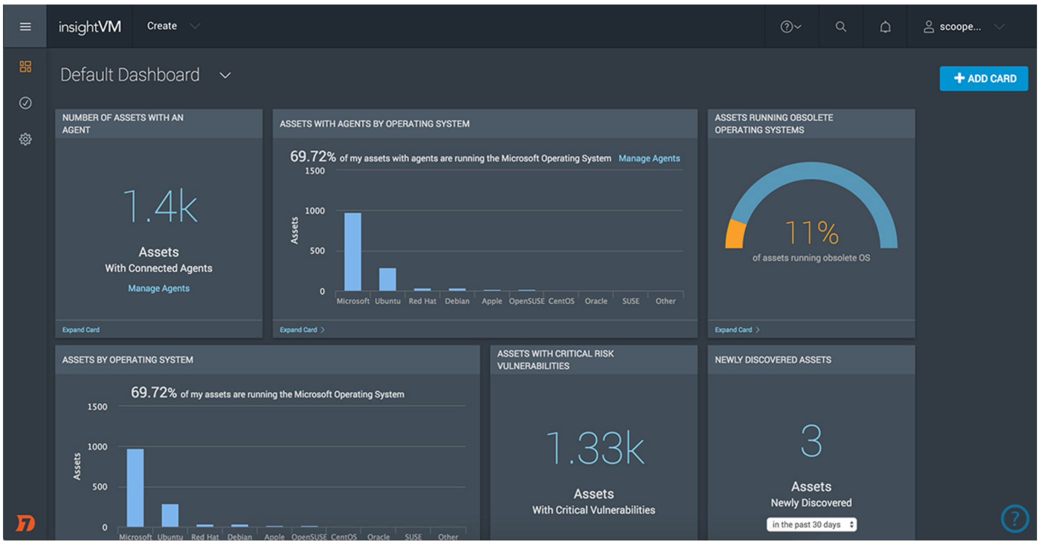
Asset Groups		Severities by Levels					Vulnerability Type			Hosts w/Vuln Levels 5,4,3		Vulnerability Status				Vulnerability Age by Days		
Title	Hosts	Level 5	Level 4	Level 3	Level 2	Level 1	Confirmed	Potential	IG	Total	%	New	Active	Fixed	ReOpen	> 30	> 60	> 90
The subnet	73	1783	1880	1783	1783	1783	7986	777	777	112	89%	80	80	80	80	0	0	8763
RSA Demo	28	1125	1282	1125	1125	1125	5289	203	203	62	100%	80	80	80	80	0	0	5494
Windows Systems	10	844	92	844	844	844	3076	56	56	28	100%	80	80	80	80	0	0	3132
AIX 5&6 (Chiharu)	1	0	0	0	0	0	257	15	15	12	100%	80	80	80	80	0	0	6
Empty	1	0	0	0	0	0	11	2	2	1	0	80	80	80	80	0	0	13

Figure 2.6: Screenshot with exemplary vulnerability scorecard report of Qualys VM, taken from [48]

According to [46], vulnerabilities are divided into common classes first, and then combined with sets of rules. Based on these rules, hosts that appear or disappear in the network or run an unexpected operating system are detected. When an alert is detected, the host's name, IP address, DNS name, NetBIOS name, operating system, open ports, installed software, and vulnerabilities affect-ing the host are collected. Further details as to how vulnerabilities are assessed and scored into severity levels, or on the process of suggesting appropriate patches are not provided.

Information on which devices or operating systems are supported by Qualys is not available online. There is no indication whether their solutions cover IoT devices. Additionally, Qualys' target group are industrial applications, no smart home end-users.

With InsightVM, Rapid7 offers yet another proprietary industry software solution for (network) vulnerability assessment and management. Rapid7 claims to combine the Common Vulnerability Score System (CVSS) with their rating of exploitability, exposure to malware and vulnerability age resulting in a "Real Risk Score" between 1 and 1000 [50]. In accordance with Nessus professional and QualysVM, InsightVM extends mere vulnerability assessment with management and real-time reporting functionality [49]. An exemplary screenshot of the resulting default dashboard is given in Figure 2.7.



69.72% of my assets with agents are running the Microsoft Operating System

of assets running obsolete OS

69.72% of my assets are running the Microsoft Operating System

Figure 2.7: Screenshot of the default dashboard of InsightVM, taken from [49]

Again, further information on how the actual scanning and scoring are performed and which devices or operating systems are supported is not provided online. There is also no indication that Rapid7 targets IoT devices.

2.3.2 Database-based Vulnerability Management Solutions

A straightforward way of assessing vulnerabilities of the software deployed on IoT devices is proposed by the IETF [34]. Similar to SBOMs for describing software inventory information, there exist several data formats for describing vulnerability information. They can be used to compile reports about the system state, but also be combined with SBOMs. The Vulnerability Assessment procedure proposed by the IETF includes the assessment of all SBOMs of the devices under consideration. These are then linked to a vulnerability database to find disclosed vulnerabilities related to the software modules.

An extensive system to correlate data of publicly disclosed vulnerabilities with standardized description across tools, databases, and people is provided by the Common Vulnerabilities and Exposures (CVE) system [60]. It is maintained by the U.S. National Cybersecurity Federally Funded Research and Development Center (FFRDC) that is operated by MITRE Corporation and also supports the U.S. National Institute of Standards and Technology (NIST). Its main contribution is the list of CVE records that allows standardized, centralized, and automated Vulnerability Management [45].

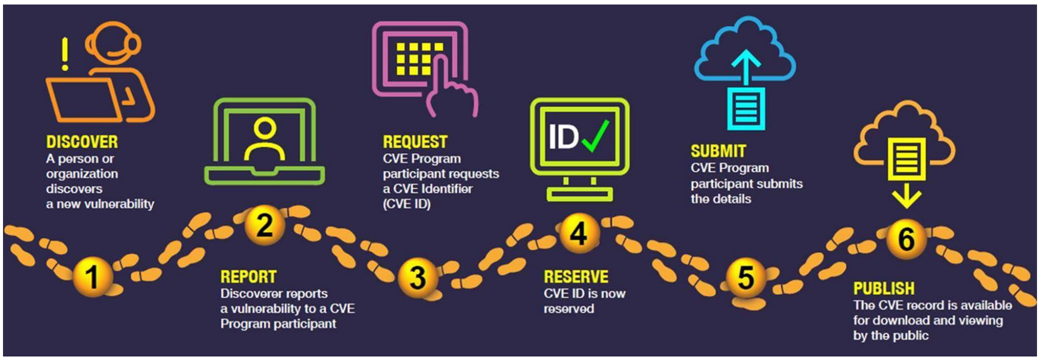

DISCOVER
A person or organization discovers a new vulnerability
1
2
REPORT
Discoverer reports a vulnerability to a CVE Program participant
REQUEST
CVE Program participant requests a CVE Identifier (CVE ID)
3
ID✓
4
RESERVE
CVE ID is now reserved
SUBMIT
CVE Program participant submits the details
5
6
PUBLISH
The CVE record is available for download and viewing by the public

Figure 2.8: Lifecycle for any CVE Record in the CVE List, taken from [63]

The lifecycle of every such CVE record is depicted in Figure 2.8. Whenever a new, independently fixable vulnerability is discovered by either a private individual or an organization, it is reported to a CVE Program participant. The CVE Program participant is then responsible for requesting a unique CVE ID that is reserved upon request by a CVE Numbering Authority (CNA). As next step towards publishing the new vulnerability, details concerning the reserved CVE ID are to be submitted. ”Details include but are not limited to affected product(s); affected or fixed product versions; vulnerability type, root cause, or impact; and at least one public reference.” [63] Finally, when the minimum required information are included in the CVE Record, it is published by the responsible CNA and added to the official, publicly disclosed CVE List.

The CVE ID is composed of the prefix ”CVE”, the year either the CVE ID was reserved or the vulnerability was made public and a sequence of four or more arbitrary digits. Hence, for example **CVE-2021-12345**.

The minimum required information for a CVE Record to be published is defined as follows [62]: the CVE ID, the name of an affected product, the affected or fixed version(s), at least one public reference, a prose description written in English, an indication whether this vulnerability only affects end-of-life products, and at least one of the following: the vulnerability type, its root cause, or its impact. In addition to that, the CVE Program encourages CVE CNAs to provide additional information.

The full XML Schema Design for the CVE List available for download can be found in Listing 1 in the appendix. Vulnerabilities that result in a new CVE record can be found by private individuals, organizations or the affected product vendor itself.

2.3.3 Conclusion

To conclude, the most popular Vulnerability Assessment and Management solutions according to Gartner [24] are all targeting business customers and no end-users. Information on whether IoT devices and typical IoT operating systems are supported is sparse, if available at all. Since all discussed solutions are proprietary, deep information on how the scanning process is implemented or how the scoring and prioritization of vulnerabilities is performed are not provided.

Additionally, Vulnerability Assessment in the IoT domain is a relatively new research field [3]. Hence, assessment approaches that are based on actively scanning the network and devices is considered a meaningful future enhancement for the VAM solution proposed by this thesis. Nevertheless, a linking to a CVE-database seems to be performed by all presented solutions. Qualys VMDR even bases their vulnerability scoring on the CVSS described in Section 2.4.

The CVE database is fast growing with tens of new entries per day [9]. It is widely used among many large organizations, but not yet as popular in the IoT domain. Nevertheless, it can be expected that vendors of IoT devices will incorporate CVEs in their product vulnerability management in the future, since it is one of the largest public sources of vulnerability information:

"Across organizations, IT-security solutions vendors, and security experts, CVE has become the de facto standard of sharing information on known vulnerabilities and exposures." [9] Therefore, the CVE database has been chosen for the VAM solution proposed by this thesis for linking SBOMs to it.

2.4 IoT Security Scoring Systems

After the Vulnerability Assessment process is completed (see Section 2.3), the vulnerabilities present in the monitored network have to be rated. Depending on the chosen evaluation criteria, vulnerabilities can have different grades of severity and pose a different risk to the device or the network.

One widely used approach is to translate characteristics of the vulnerability to a single score that represents the severity. The actual risk assessment then strongly depends on individual details of the affected device and its environment: A vulnerability with medium severity in a device that serves as a smart door lock may pose a higher risk to the user than a vulnerability with high severity in a smart light bulb. Due to the strong dependency on individual circumstances, considerations regarding risk assessment will not be discussed here.

However, being able to rate the severity of assessed vulnerabilities not only serves as the foundation for further risk assessment, but also abstracts the details that might overwhelm an inexperienced user and provides a numeral basis for the computation of a Security Label (see Section 2.1).

Such scoring systems for the IoT domain are subject of current research and selected approaches are presented in the following section. The most widely used scoring system is the Common Vulnerability Scoring System (CVSS). Its enormous practical relevance makes it the de-facto standard in industry. However, it is not explicitly targeting the IoT domain. With [35], Loi et al. propose a three-level rating for selected IoT devices that may serve as a basis for a star-rating or for deriving a single numerical score value from it. However, the rating mechanism is solely based on the authors' expert knowledge and cannot be automated. A precicely specified scoring system based on CVSS but extended with additional metrics is provided by Alrawi et al. [2].

"The Common Vulnerability Scoring System (CVSS) is an open framework for communicating the characteristics and severity of software vulnerabilities." (see [22], p. 1) It is widely used worldwide for risk assessment and prioritization of vulnerabilities and serves as de-facto industry standard, most prominently recommended by the U.S. National Institute of Standards and Technology (NIST), and the global payment card industry [55]. With the Special Publication 800-115, NIST recommends the NIST National Vulnerability Database (NVD) for analysing the vulnerabilities found through VA. In the NVD, CVEs are attributed with their corresponding CVSS [39].

To assess the CVSS v3.1 base score [22], eight questions about the vulnerability have to be answered by an expert with deep knowledge about the vulnerability and its impacts. This input is then translated to a score between 0 and 10, where relative importance is reflected in ratio values [55]. The purpose of the base score is to reflect the constant, intrinsic characteristics of the vulnerability [22]. In an optional extension on top of this base score, temporal and environmental metrics can be taken into account to further refine the risk assessment. Since it is common for entities responsible of the vulnerable product to only provide the base score, which is constant over time and across user environments, this overview focusses on the base score computation.

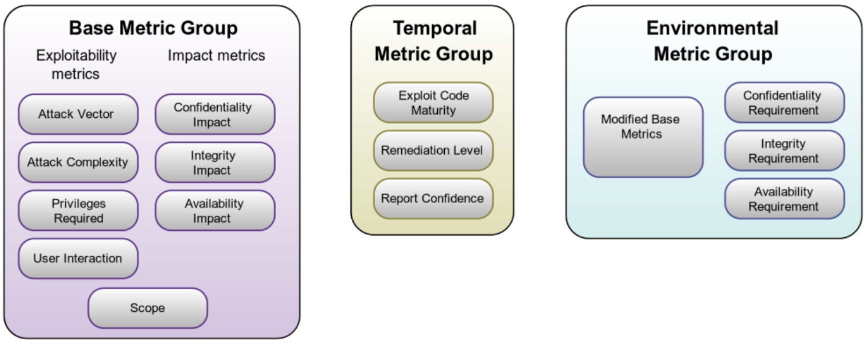


Figure 2.9: Metric groups of the Common Vulnerability Scoring System (CVSS) v3.1, taken from [22]

The base metric group consists of exploitability metrics and impact metrics [22] as presented in Figure 2.9. The temporal and environmental metric groups are provided for completeness. The exploitability metrics are the attack vector, the attack complexity, the privileges required, and the user interaction. They characterize the vulnerable component, which is typically a software component (application, module, driver, etc.), but may also be a hardware device. The exploitability metrics result in a exploitability sub-score.

- Attack Vector
  The attack vector metric value is larger the more remotely the vulnerability can be exploited. Possible metric values with their numerical values are *Network* (0.85), *Adjacent* (0.62), *Local* (0.55), and *Physical* (0.2).
- Attack Complexity
  The attack complexity metric value reflects whether specific configurations and conditions beyond the attacker's control are necessary for the exploit. Possible metric values are *Low* (0.77) and *High* (0.44).

- Privileges Required
  The privileges required metric value is larger the less privileges the attacker has to possess for executing the exploit. Possible metric values are *None* (0.85), *Low* (0.62, resp. 0.68 if Scope is *Changed*), and *High* (0.27, resp. 0.5 if Scope is *Changed*).

- User Interaction
  The user interaction metric value is larger when a successful exploit does not require a user input other than the attacker's. Possible metric values are *None* (0.85) and *Required* (0.62).

The Exploitability sub score is defined as:

$$\begin{aligned} Exploitability = 8.22 * AttackVector * AttackComplexity \\ * PrivilegesRequired * UserInteraction \end{aligned} \tag{2.1}$$

The impact metrics consist of the impact on the CIA-triade: Confidentiality, Integrity, Availability. They take the direct consequences of successful exploits of this vulnerability into account, whereby the impacted component does not necessarily have to be the vulnerable component itself, but can also be a network resource. The impact metrics result in an impact sub-score.

- Confidentiality
  The confidentiality metric value is larger when a successful exploit results in more extensive disclosure of information to unauthorized users. Possible metric values are *High* (0.56), *Low* (0.22), and *None* (0).

- Integrity
  The integrity metric value is larger when the trustworthiness and veracity of information is more extensively impaired. Possible metric values are *High* (0.56), *Low* (0.22), and *None* (0).

- Availability
  The availability metric value is larger when the availability of the impacted component is more extensively impaired. Possible metric values are *High* (0.56), *Low* (0.22), and *None* (0).

The Impact sub score (ISS) and the Impact are defined as:

$$\begin{aligned} ISS &= 1 - [(1 - Confidentiality) * (1 - Integrity) * (1 - Availability)] \\ Impact &= 6.42 * ISS, \text{ if Scope is } Unchanged \\ &= 7.52 * (ISS - 0.029) - 3.25 * (ISS - 0.02)^{15}, \text{ if Scope is } Changed \end{aligned} \tag{2.2}$$

In addition to the exploitability and impact metrics, the scope of the exploit is incorporated as eighth metric value. It reflects a change of the security scope of a vulnerable component. Possible metric values are *Changed* and *Unchanged*.

Please, refer to the specification document for further details on the metric values [22].

The Base Score is defined as:

$$\begin{aligned} BaseScore &= 0,\ \text{if } Impact <= 0, else \\ &= Roundup(Minimum[(Impact + Exploitability), 10]), \\ &\text{if Scope is } Unchanged \\ &= Roundup(Minimum[1.08 * (Impact + Exploitability), 10]), \\ &\text{if Scope is } Changed \end{aligned} \tag{2.3}$$

The resulting CVSS Base Score is then mapped to a textual representation:

- CVSS Base Score 0.0: None
- CVSS Base Score 0.1-3.9: Low
- CVSS Base Score 4.0-6.9: Medium
- CVSS Base Score 7.0-8.9: High
- CVSS Base Score 9.0-10.0: Critical

Despite the worldwide utilization of the CVSS, however, Spring et al. argue that "it is not justified, either formally or empirically". They criticise that "the CVSS v3.0 documentation offers no evidence or argument that the formula or construction method are robust. The calculation method is clear, though unjustified. " [55] Additionally, Figueroa-Lorenzo et al. criticise that the CVSS is not properly adapted to industrial IoT environments [21]. Despite all criticism, the Common Vulnerability Scoring System convinces with its practical relevance and the availability of scoring data for all disclosed CVEs through the National Vulnerability Database. The search for more justified severity assessment systems better suited for the IoT domain is subject to ongoing research and as long as no candidate has prevailed there, the CVSS is a good starting point for the risk assessment of this VAM solution (see Section 3).

However, since the CVSS is not targeted to the IoT domain, other approaches for scoring systems are considered as well.

In [35], Loi et al. propose a security test suite that supports 20 consumer IoT devices and automatically checks for attack vectors that threaten the confidentiality of data, the integrity of data, and the access control and that allow for reflective attacks launched from another IoT device. All measures concerning these dimensions are given below. Loi et al. claim that the proposed methodology may serve as basis for a star-based security rating for IoT devices.

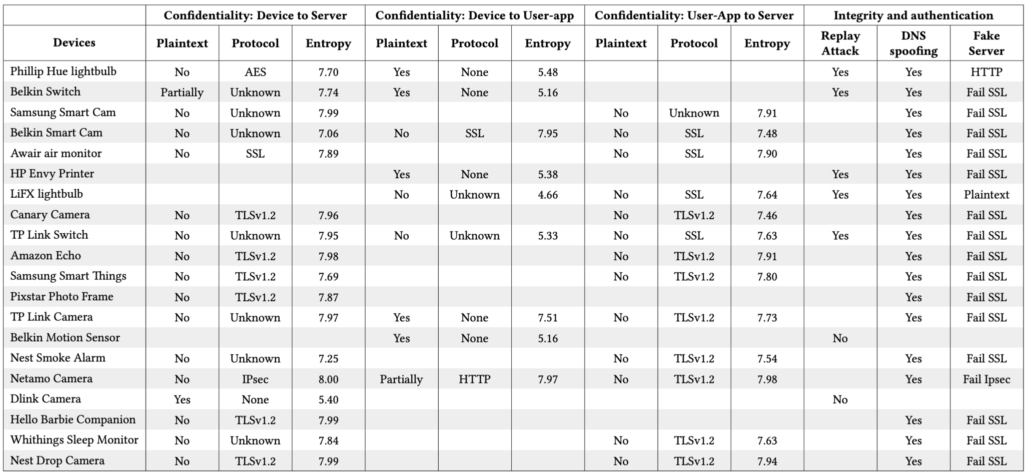
- Confidentiality of exchanged data:
  all exchanged data is either
  - plaintext,
  - encoded, or
  - encrypted
- Integrity:
  the device only performs intended functions and all exchanged messages are unmodified
  includes tests for:
  - replay attacks
    - legitimate packets from user application to device are captured and replayed using a Python script
    - for non-encrypted-packets: certain packet fields are modified and send to the device to check the device's response
  - DNS spoofing
    - check DNS queries for DNSSEC
    - perform DNS spoofing, if DNSSEC is not used
- Access Control and Availability:
  perform vulnerability assessment for attacks with low complexity and DoS attacks
  - scan for open ports using nmap
  - test for ports with known login credentials used by the Mirai botnet (Telnet, SSH, HTTP)
  - automatically perform ICMP pings and send UDP packets using hping3 and send TCP SYN packets to flood the device and stop it from operating properly

In order to achieve data confidentiality, all communication channels between device and cloud server, device and user application and user application and cloud server are assessed using ARP spoofing. To identify the utilized security protocol, then, the protocol field gets inspected. If the packet's payload is human-readable, it is considered to be plaintext. Otherwise, an automated entropy test for the presence of encryption and its strength is performed.

It is not explained in detail if and how the proposed tests can be completely automated without any user interaction, e.g. for capturing or inspecting packets. However, the proposed checks for integrity and access control and availability are not applicable to a runtime risk assessment system during normal operation of the device under test in a smart home environment, since they stop the device from functioning properly.

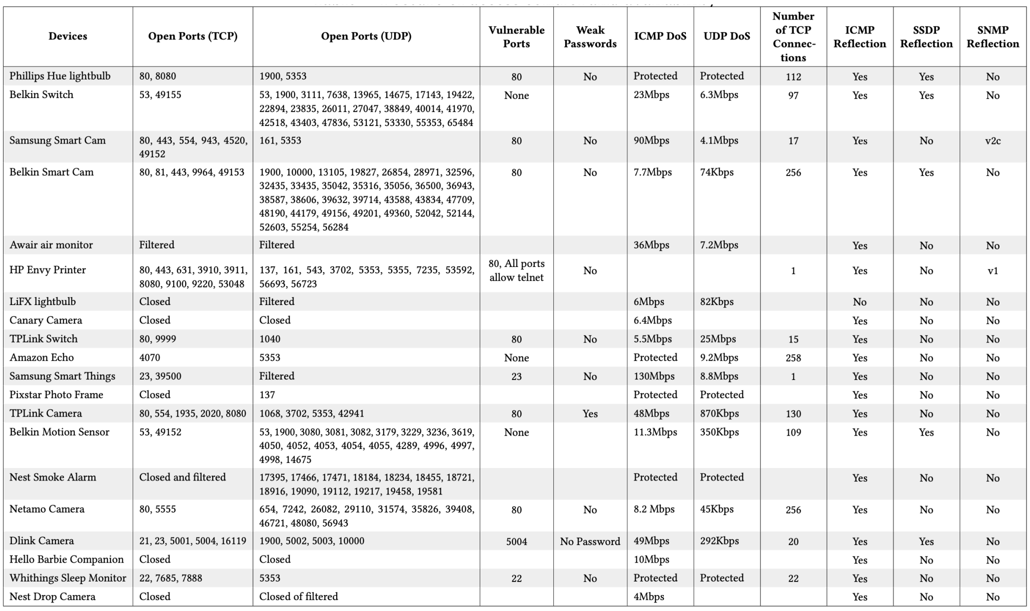
Loi et al. performed the proposed tests on 20 consumer IoT devices. The results are shown in figures 2.10 and 2.11.

In order to derive a score from the test suite's result, a three-level rating is proposed for the 20 consumer IoT devices under test (see figure 2.12), where a rating of A represents the device being secure, B being moderately secure/insecure, and C being insecure. It is not explained in further detail, how the differentiation is being made. Instead, the proposed rating is described as "subjective and given based on authors perceptions". Deriving a single score by giving weights to each dimension is described as future work.



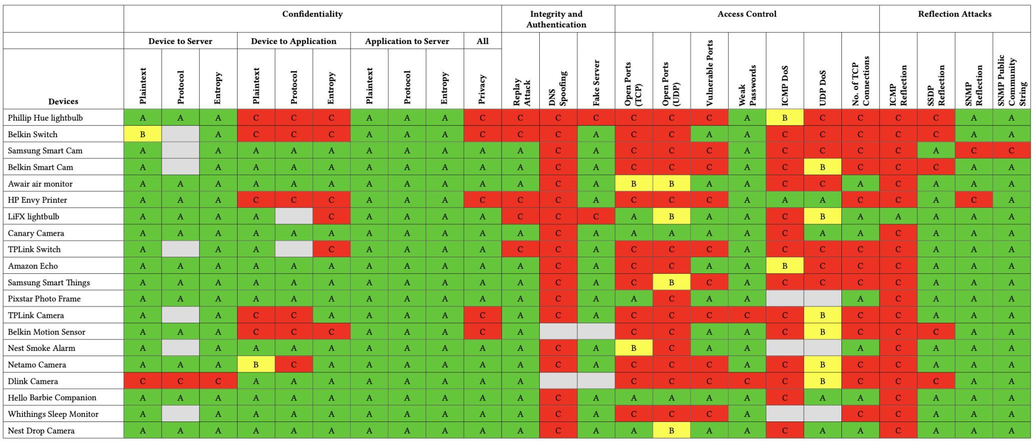
	Confidentiality: Device to Server			Confidentiality: Device to User-app			Confidentiality: User-App to Server			Integrity and authentication		
Devices	Plaintext	Protocol	Entropy	Plaintext	Protocol	Entropy	Plaintext	Protocol	Entropy	Replay Attack	DNS spoofing	Fake Server
Phillip Hue lightbulb	No	AES	7.70	Yes	None	5.48				Yes	Yes	HTTP
Belkin Switch	Partially	Unknown	7.74	Yes	None	5.16				Yes	Yes	Fail SSL
Samsung Smart Cam	No	Unknown	7.99				No	Unknown	7.91		Yes	Fail SSL
Belkin Smart Cam	No	Unknown	7.06	No	SSL	7.95	No	SSL	7.48		Yes	Fail SSL
Awair air monitor	No	SSL	7.89				No	SSL	7.90		Yes	Fail SSL
HP Envy Printer				Yes	None	5.38				Yes	Yes	Fail SSL
LiFX lightbulb				No	Unknown	4.66	No	SSL	7.64	Yes	Yes	Plaintext
Canary Camera	No	TLSv1.2	7.96				No	TLSv1.2	7.46		Yes	Fail SSL
TP Link Switch	No	Unknown	7.95	No	Unknown	5.33	No	SSL	7.63	Yes	Yes	Fail SSL
Amazon Echo	No	TLSv1.2	7.98				No	TLSv1.2	7.91		Yes	Fail SSL
Samsung Smart Things	No	TLSv1.2	7.69				No	TLSv1.2	7.80		Yes	Fail SSL
Pixstar Photo Frame	No	TLSv1.2	7.87								Yes	Fail SSL
TP Link Camera	No	Unknown	7.97	Yes	None	7.51	No	TLSv1.2	7.73		Yes	Fail SSL
Belkin Motion Sensor				Yes	None	5.16				No		
Nest Smoke Alarm	No	Unknown	7.25				No	TLSv1.2	7.54		Yes	Fail SSL
Netamo Camera	No	IPsec	8.00	Partially	HTTP	7.97	No	TLSv1.2	7.98		Yes	Fail Ipsec
Dlink Camera	Yes	None	5.40							No		
Hello Barbie Companion	No	TLSv1.2	7.99								Yes	Fail SSL
Whithings Sleep Monitor	No	Unknown	7.84				No	TLSv1.2	7.63		Yes	Fail SSL
Nest Drop Camera	No	TLSv1.2	7.99				No	TLSv1.2	7.94		Yes	Fail SSL

Figure 2.10: Results of evaluating 20 IoT devices' confidentiality and integrity using the test suite proposed by Loi et al., taken from [35]



Devices	Open Ports (TCP)	Open Ports (UDP)	Vulnerable Ports	Weak Passwords	ICMP DoS	UDP DoS	Number of TCP Connections	ICMP Reflection	SSDP Reflection	SNMP Reflection
Phillips Hue lightbulb	80, 8080	1900, 5353	80	No	Protected	Protected	112	Yes	Yes	No
Belkin Switch	53, 49155	53, 1900, 3111, 7638, 13965, 14675, 17143, 19422, 22894, 23835, 26011, 27047, 38849, 40014, 41970, 42518, 43403, 47836, 53121, 53330, 55353, 65484	None		23Mbps	6.3Mbps	97	Yes	Yes	No
Samsung Smart Cam	80, 443, 554, 943, 4520, 49152	161, 5353	80	No	90Mbps	4.1Mbps	17	Yes	No	v2c
Belkin Smart Cam	80, 81, 443, 9964, 49153	1900, 10000, 13105, 19827, 26854, 28971, 32596, 32435, 33435, 35042, 35316, 35056, 36500, 36943, 38587, 38606, 39632, 39714, 43588, 43834, 47709, 48190, 44179, 49156, 49201, 49360, 52042, 52144, 52603, 55254, 56284	80	No	7.7Mbps	74Kbps	256	Yes	Yes	No
Awair air monitor	Filtered	Filtered			36Mbps	7.2Mbps		Yes	No	No
HP Envy Printer	80, 443, 631, 3910, 3911, 8080, 9100, 9220, 53048	137, 161, 543, 3702, 5353, 5355, 7235, 53592, 56693, 56723	80, All ports allow telnet	No			1	Yes	No	v1
LiFX lightbulb	Closed	Filtered			6Mbps	82Kbps		No	No	No
Canary Camera	Closed	Closed			6.4Mbps			Yes	No	No
TPLink Switch	80, 9999	1040	80	No	5.5Mbps	25Mbps	15	Yes	No	No
Amazon Echo	4070	5353	None		Protected	9.2Mbps	258	Yes	No	No
Samsung Smart Things	23, 39500	Filtered	23	No	130Mbps	8.8Mbps	1	Yes	No	No
Pixstar Photo Frame	Closed	137			Protected	Protected		Yes	No	No
TPLink Camera	80, 554, 1935, 2020, 8080	1068, 3702, 5353, 42941	80	Yes	48Mbps	870Kbps	130	Yes	No	No
Belkin Motion Sensor	53, 49152	53, 1900, 3080, 3081, 3082, 3179, 3229, 3236, 3619, 4050, 4052, 4053, 4054, 4055, 4289, 4996, 4997, 4998, 14675	None		11.3Mbps	350Kbps	109	Yes	Yes	No
Nest Smoke Alarm	Closed and filtered	17395, 17466, 17471, 18184, 18234, 18455, 18721, 18916, 19090, 19112, 19217, 19458, 19581			Protected	Protected		Yes	No	No
Netamo Camera	80, 5555	654, 7242, 26082, 29110, 31574, 35826, 39408, 46721, 48080, 56943	80	No	8.2 Mbps	45Kbps	256	Yes	No	No
Dlink Camera	21, 23, 5001, 5004, 16119	1900, 5002, 5003, 10000	5004	No Password	49Mbps	292Kbps	20	Yes	Yes	No
Hello Barbie Companion	Closed	Closed			10Mbps			Yes	No	No
Whithings Sleep Monitor	22, 7685, 7888	5353	22	No	Protected	Protected	22	Yes	No	No
Nest Drop Camera	Closed	Closed of filtered			4Mbps			Yes	No	No

Figure 2.11: Results of evaluating 20 IoT devices' access control and availability using the test suite proposed by Loi et al., taken from [35]



	Confidentiality										Integrity and Authentication			Access Control							Reflection Attacks			
	Device to Server			Device to Application			Application to Server			All														
Devices	Plaintext	Protocol	Entropy	Plaintext	Protocol	Entropy	Plaintext	Protocol	Entropy	Privacy	Replay Attack	DNS Spoofing	Fake Server	Open Ports (TCP)	Open Ports (UDP)	Vulnerable Ports	Weak Passwords	ICMP DoS	UDP DoS	No. of TCP Connections	ICMP Reflection	SSDP Reflection	SNMP Reflection	SNMP Public Community String
Phillip Hue lightbulb	A	A	A	C	C	C	A	A	A	C	C	C	C	C	C	C	A	B	C	C	C	C	A	A
Belkin Switch	B		A	C	C	C	A	A	A	C	C	C	A	C	C	A	A	C	C	C	C	C	A	A
Samsung Smart Cam	A		A	A	A	A	A	A	A	A	A	C	A	C	C	C	A	C	C	C	C	A	C	C
Belkin Smart Cam	A		A	A	A	A	A	A	A	A	A	C	A	C	C	C	A	C	B	C	C	C	A	A
Awair air monitor	A	A	A	A	A	A	A	A	A	A	A	C	A	B	B	A	A	C	C	A	C	A	A	A
HP Envy Printer	A	A	A	C	C	C	A	A	A	C	C	C	A	C	C	C	A	A	A	C	C	A	C	A
LiFX lightbulb	A	A	A	A		C	A	A	A	A	C	C	C	A	B	A	A	C	B	A	A	A	A	A
Canary Camera	A	A	A	A	A	A	A	A	A	A	A	C	A	A	A	A	A	C	A	A	C	A	A	A
TPLink Switch	A		A	A		C	A	A	A	A	C	C	A	C	C	C	A	C	C	C	C	A	A	A
Amazon Echo	A	A	A	A	A	A	A	A	A	A	A	C	A	C	C	A	A	B	C	C	C	A	A	A
Samsung Smart Things	A	A	A	A	A	A	A	A	A	A	A	C	A	C	B	C	A	C	C	C	C	A	A	A
Pixstar Photo Frame	A	A	A	A	A	A	A	A	A	A	A	C	A	A	C	A	A			A	C	A	A	A
TPLink Camera	A		A	C	C	A	A	A	A	C	A	C	A	C	C	C	C	C	B	C	C	A	A	A
Belkin Motion Sensor	A	A	A	C	C	C	A	A	A	C	A			C	C	A	A	C	B	C	C	C	A	A
Nest Smoke Alarm	A		A	A	A	A	A	A	A	A	A	C	A	B	C	A	A			A	C	A	A	A
Netamo Camera	A	A	A	B	C	A	A	A	A	A	A	C	A	C	C	C	A	C	B	C	C	A	A	A
Dlink Camera	C	C	C	A	A	A	A	A	A	A	A			C	C	C	C	C	B	C	C	C	A	A
Hello Barbie Companion	A	A	A	A	A	A	A	A	A	A	A	C	A	A	A	A	A	C	A	A	C	A	A	A
Whithings Sleep Monitor	A		A	A	A	A	A	A	A	A	A	C	A	C	C	C	A			C	C	A	A	A
Nest Drop Camera	A	A	A	A	A	A	A	A	A	A	A	C	A	A	B	A	A	C	A	A	C	A	A	A

Figure 2.12: Security rating results of evaluating 20 IoT devices using the test suite proposed by Loi et al. with "A" being secure, "B" being moderately secure/insecure, and "C" being insecure, taken from [35]

An approach that extends a rating of CVSS-attributed database-based Vulnerability Assessment as described above, but with additional metrics, is proposed by Alrawi et al. in [2]. There, Alrawi et al. propose a methodology for security analysis of home-based IoT devices: attack vectors and mitigations can be differentiated in those concerning the device, the mobile application, the communication or the cloud endpoint. Based on this methodology, both a systematic review of literature and the evaluation of 45 home-based IoT devices using Nessus Scanner is performed. As explained in Section 2.3.1, Nessus Scanner scans devices for services and performs a vulnerability scoring based on the Common Vulnerability Scoring System explained above. An exemplary list of detected CVEs and their corresponding CVSS for one device can be found in Figure 2.13.

In addition to the proposed results, Alrawi et al. established an evaluation portal that also includes a scoring system for all evaluated devices [71], that differentiates scoring rubrics into concerning the device, the mobile application, the cloud endpoint and the network connection. Since the security index proposed here is focussed on the device, only the device-related scoring factors are given below (see [2]):

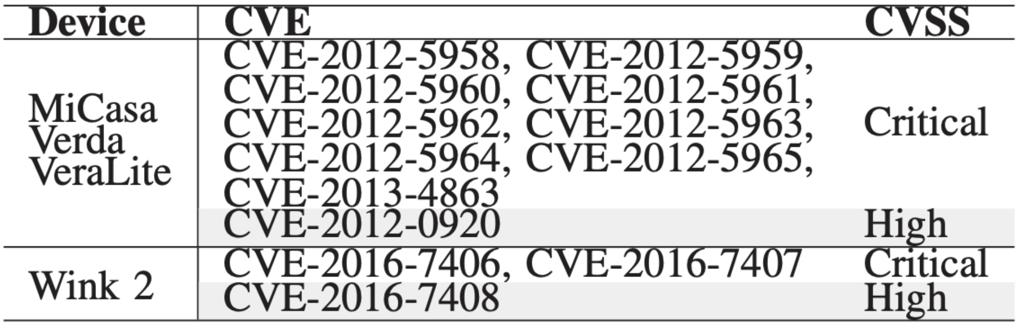
- Device (42 Points of a total of 175 Points)
  - Internet Pairing (3 Points)
    - Wifi - device broadcasts unsecured wifi to allow user to connect and configure (3 Points)
    - Low-Energy (LE) - device uses LE protocol to pair with mobile to configure the device (2 Points)
    - Wired - device uses a wired medium to directly connect to the local network (1 Point)
    - Manual - device requires user to manually input network credentials to connect and configure the device (0 Points)
  - Configuration (7 Points)
    - Default configuration (7 Points)
    - Forced configuration (0 Points)
  - Upgradeability (4 Points)
    - Manual (4 Points)
    - Consent-based (1 Point)
    - Automatic (0 Points)
  - Exposed Services (4 Points)
    - 5 or more services (4 Points)
    - 3-4 services (3 Points)
    - 1-2 services (2 Points)
    - No services (0 Points)
  - Vulnerabilities (24 Points)
    - Critical
      - 11 or more critical vulnerabilities (10 Points)
      - 6-10 (9 Points)

    - 1-5 (8 Points)
  - High
    - 11 or more highly severe vulnerabilities (7 Points)
    - 6-10 (6 Points)
    - 1-5 (5 Points)
  - Medium
    - 11 or more medium severe vulnerabilities (4 Points)
    - 6-10 (3 Points)
    - 1-5 (2 Points)
  - Low
    - 11 or more low severe vulnerabilities (3 Points)
    - 6-10 (2 Points)
    - 1-5 (1 Point)

Further justification of the weighting of the scoring points is not proposed.

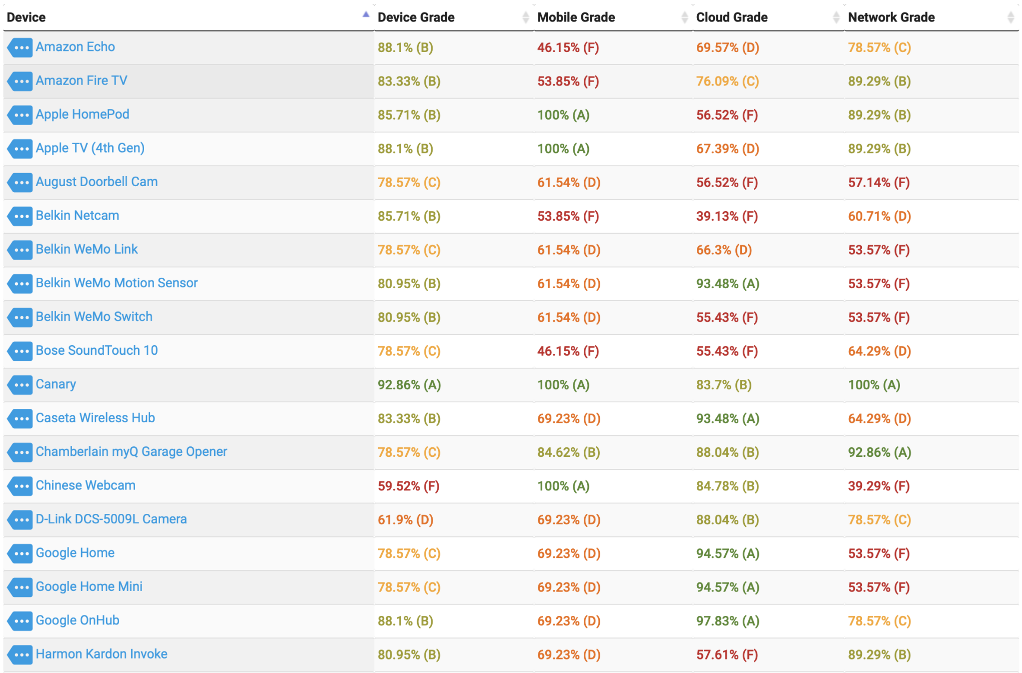
Based on those points, a simple score is calculated for each rubric: divide the sum of points by the total number of points of that rubric and subtract the result from one (therefore, a higher score means less security vulnerabilities detected). Based on general cutoffs, a grade letter can be assigned (A - 0.9+, B - 0.8+, etc.).

Scorecards for currently 45 devices are proposed as well (see Figure 2.14) [70].



Device	CVE	CVSS
MiCasa Verda VeraLite	CVE-2012-5958, CVE-2012-5959, CVE-2012-5960, CVE-2012-5961, CVE-2012-5962, CVE-2012-5963, CVE-2012-5964, CVE-2012-5965, CVE-2013-4863	Critical
	CVE-2012-0920	High
Wink 2	CVE-2016-7406, CVE-2016-7407	Critical
	CVE-2016-7408	High

Figure 2.13: Exemplary list of CVEs and their CVSS score, taken from [2]



Device	Device Grade	Mobile Grade	Cloud Grade	Network Grade
Amazon Echo	88.1% (B)	46.15% (F)	69.57% (D)	78.57% (C)
Amazon Fire TV	83.33% (B)	53.85% (F)	76.09% (C)	89.29% (B)
Apple HomePod	85.71% (B)	100% (A)	56.52% (F)	89.29% (B)
Apple TV (4th Gen)	88.1% (B)	100% (A)	67.39% (D)	89.29% (B)
August Doorbell Cam	78.57% (C)	61.54% (D)	56.52% (F)	57.14% (F)
Belkin Netcam	85.71% (B)	53.85% (F)	39.13% (F)	60.71% (D)
Belkin WeMo Link	78.57% (C)	61.54% (D)	66.3% (D)	53.57% (F)
Belkin WeMo Motion Sensor	80.95% (B)	61.54% (D)	93.48% (A)	53.57% (F)
Belkin WeMo Switch	80.95% (B)	61.54% (D)	55.43% (F)	53.57% (F)
Bose SoundTouch 10	78.57% (C)	46.15% (F)	55.43% (F)	64.29% (D)
Canary	92.86% (A)	100% (A)	83.7% (B)	100% (A)
Caseta Wireless Hub	83.33% (B)	69.23% (D)	93.48% (A)	64.29% (D)
Chamberlain myQ Garage Opener	78.57% (C)	84.62% (B)	88.04% (B)	92.86% (A)
Chinese Webcam	59.52% (F)	100% (A)	84.78% (B)	39.29% (F)
D-Link DCS-5009L Camera	61.9% (D)	69.23% (D)	88.04% (B)	78.57% (C)
Google Home	78.57% (C)	69.23% (D)	94.57% (A)	53.57% (F)
Google Home Mini	78.57% (C)	69.23% (D)	94.57% (A)	53.57% (F)
Google OnHub	88.1% (B)	69.23% (D)	97.83% (A)	78.57% (C)
Harmon Kardon Invoke	80.95% (B)	69.23% (D)	57.61% (F)	89.29% (B)

Figure 2.14: Exemplary extract of scorecards, taken from [70]

Although the scoring system proposed by Alrawi et al. does not provide justification of their score computation either, they rely on the globally used CVSS and therefore benefit from its practical relevance. But based on CVE assessment and their attribution with CVSS scores, further risk assessment is provided with additional metrics that are of special interest in the smart home IoT domain. Therefore, the device scoring system proposed by Alrawi et al. serves as a basis for the runtime security score proposed with this thesis and described in detail in Section 3.

2.5 Trusted Execution Environments through Hardware Root of Trust

In the Personal Computer (PC) domain, security by design principles are well researched and established. A main building block of security for general purpose machines is process isolation. Thereby, processes from different Software Providers can be securely separated from each other. Commonly, this is enabled through Virtual Memory enforced by the Memory Management Unit (MMU). During runtime, virtual memory regions of processes are assigned to physical memory by the MMU. The latter then also ensures that processes cannot access memory regions assigned to another process.

In contrast, the concept of processes separated from each other is not commonly supported in the embedded domain. Especially devices with a very simple and dedicated functionality do not support interleaved execution of different processes and hence do not provide isolation mechanisms. Nevertheless, more and more embedded devices are connected to the worldwide Internet and form the IoT sector. It is increasingly becoming a use case that an entity, e.g. the device Manufacturer, merges different software products (firmware, application software, third-party software, libraries) into one binary file that is flashed on the device before shipment, hence gathering several pieces of software from different providers on the device.

Therefore, there is an increasing need in the IoT domain to securely separate pieces of software from each other and to establish so called Trusted Execution Environments (TEE) supported by secure Hardware mechanisms, the so called Hardware Root of Trust (RoT). Nevertheless, isolation mechanisms known from the PC domain (such as Virtual Memory) cannot be applied easily to constrained IoT devices. Due to the limitations in computational power, power consumption and form factor, a complex MMU cannot be placed on the System on Chip (SoC). Thus, software isolation mechanisms for constrained IoT devices represent a relatively new branch of research and selected representatives are presented in the following sections.

A main advantage of the processor architectures with Hardware Root of Trust presented in this section is that the hardware-enforced isolation does not require the help of an OS to secure the separation. Hence, the OS in total does not necessarily have to be trusted. The underlying idea is to keep the Trusted Computing Base (TCB) as small as possible since smaller code size inherently reduces the likelihood of undetected code vulnerabilities, instead of incorporating the OS kernel, privileged services, and libraries in it. Therefore, providing a Trusted Execution Environment (TEE) allows for an isolated environment to execute trusted applications in. Such a TEE could then be used to securely provide services such as remote attestation (see Section 2.6) in order to determine the system security state.

The most widespread approach from industry is the ARMv8-M architecture with Trustzone, presented in Section 2.5.1. A processor architecture developed in academic context is provided with Sancus by the KU Leuven and described in Section 2.5.2. A large open-source community project is formed around the RISC-V processor architecture described in Section 2.5.3. It originated at the University of California, Berkeley, in 2010 and comprises modular, open standards for microprocessors. There exist several industry partner that offer SoCs compliant with RISC-V standards, both proprietary and open-source. Therefore, it is to be classified between industry and academia.

2.5.1 ARMv8-M with Trustzone

According to [44], with 60%, ARM processors have the greatest share in the embedded market. With ARMv8-M, ARM introduced the first microprocessor architecture with their Trustzone technology for their range of Cortex-M processors that "power the most energy-efficient embedded devices" [4]. Before, Trustzone has only been available for Cortex-A processors for "supreme performance at optimal power" [4]. Despite the same name, both Trustzone implementations differ significantly. This thesis focusses exclusively on Trustzone for ARMv8-M.

Trustzone represents a system-wide security approach for providing the two separated protection domains "secure world" and "non-secure world" [44]. Secure context switching between both worlds is not implemented using a monitor or a hypervisor, but through a set of mechanisms implemented in Hardware. Hence, Trustzone for Cortex-M provides a Hardware Root of Trust and a strong hardware-enforced separation of the secure world from the non-secure world. Within the secure world, however, there is no additional separation of software components possible per default. Without a Memory Protection Unit (see below), all software running in the secure world has the same extensive permissions.

The separation of both worlds is based on a configurable memory-map that divides all memory in secure regions, non-secure regions and non-secure callable regions. Hence, the processor state is derived directly from the position of the instruction pointer. If it points to a memory address assigned to the secure world, the processor state is secure, and vice versa. The processor modes are orthogonal to its states: There is a Thread mode and a Handler mode in both worlds. Apart from the differentiation of secure and non-secure memory, there is no further separation of code and data in memory supported by Trustzone.

Switching between both worlds utilizes dedicated entry points. Therefore, three new processor instructions are implemented in hardware: secure gateway (SG), branch with exchange to non-secure state ($BXNS$) and branch with link and exchange to non-secure state ($BLXNS$). Only the secure gateway instruction allows for switching to secure world. All SG instructions have to lie in non-secure callable memory to be treated as such in order to prevent other binary data (e.g. a lookup table) with a value that equals the SG instruction's opcode to be misused as entry function for the secure world.

An optional Memory Protection Unit (MPU) can be included in the SoC design and provides additional opportunities for assigning memory access permissions for privileged and unprivileged software. It is possible to add an MPU for only the secure world, or only the non-secure world, or both. Both the secure and the non-secure MPU can be implemented to allow different numbers of protected memory regions.

In practise, Trustzone allows for not having to trust the OS by loading a secure firmware in the secure world through a secure bootloading mechanism. This secure firmware then takes responsibility for "setting up its internal data structures, configuring the interrupt controller of the entire system, and setting up protections for secure memory regions and peripherals" [44]. Upon completion, execution can be handed over to the bootloader of the rich OS in non-secure world.

2.5.2 Sancus

An approach from academia for a processor architecture providing a Trusted Execution Environment was developed at KU Leuven and first presented in 2013 at the USENIX Conference on Security [41]. An extended version has then been published in 2017 [42]. All further discussion of the architecture refers to Sancus 2.0 from 2017.

The main proposal of the Sancus project is the Sancus security architecture. It is claimed to provide strong isolation, remote attestation, and secure communication [42]. The underlying system model is shown in Figure 2.15 and is based on the idea that the infrastructure provider IP owns and administrates the hardware, hence nodes N_1 to N_k. The Software deployed on the devices is then provided by the software providers SP_1 to SP_j in the form of software modules $SM_{1,1}$ to $SM_{j,k}$. This use case requires strong isolation in particular due to the extensibility provided by new software modules from the software providers.

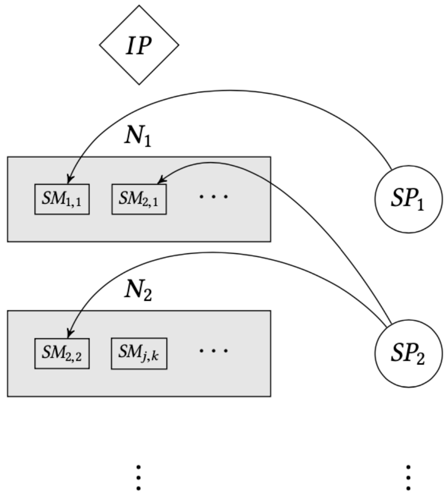


Figure 2.15: Overview of the Sancus system model, taken from [42]

The main advantage of the Sancus architecture is that only the hardware has to be trusted. There is no software Trusted Computing Base (TCB), all isolation is strongly hardware-enforced and based on a set of three cryptographic primitives for symmetric encryption. These are a cryptographic hash function, a key derivation function ”to derive a cryptographic key from a master key and some diversification data”, and an authenticated encryption with associated data, providing both encryption and decryption of data. For a hardware TCB, these cryptographic primitives have to be implemented in hardware.

Using these primitives, a hierarchy of keys is established. The underlying master key is the symmetric node master key K_N, shared between the node and the infrastructure provider. Using this secret node master key and the unique software provider ID SP as input for the key derivation function mentioned above, a symmetric key $K_{N,SP}$ for securing the communication between the node and the software provider is available.

Every software module on a node is divided into a protected and unprotected text section for code and a protected and unprotected data section for runtime data as shown in Figure 2.16. This offers software providers the possibility of keeping their software TCB as small as possible.

The cryptographic hash function is then used to compute the identity of a software module ID_{SM_n} from the content of the corresponding protected text section and the start and end addresses of the protected text and data sections. The latter is also referred to as the layout of the software module.

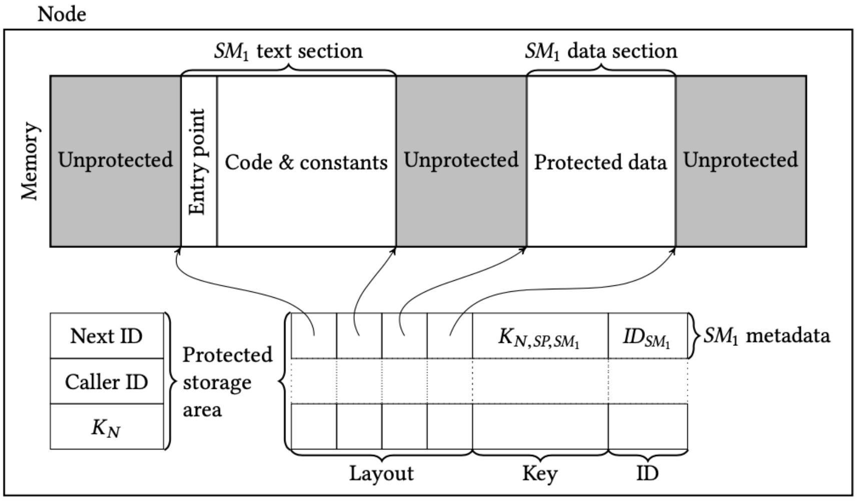


Figure 2.16: Memory layout of the Sancus architecture, taken from [42]

Memory protection in Sancus is enabled without a dedicated Memory Protection Unit, but through "program counter-based memory access control". Thereby, the data region is not executable and can only be accessed if the program counter resides in the corresponding text section of the exact same software module. Additionally, all keys described above are stored in a protected memory region and can only indirectly be accessed through a set of processor instructions. This access depends on the program counter as well and always returns the keys of the software module the program counter resides in. Entering a software module is only allowed through designated entry points. All other access is inhibited.

2.5.3 RISC-V with Physical Memory Protection (PMP)

RISC-V is an open-source Reduced Instruction Set Computer (RISC) architecture with background in academia. It started as fifth generation of RISC architectures in 2010 at the University of California, Berkeley, and stands out between other ISA designs due to its open source license. Additionally, its modular design allows SoC manufacturers to combine extensions to precisely meet an application use case.

Based on the base ISA and the combination of extensions, there exist approaches that provide Trusted Execution Environments. A lightweight representative is Multizone, a software layer that manages the hardware security blocks to provide secure isolation of TEEs. In contrast to Trustzone, Multizone provides a configurable number of hardware-enforced, software-defined isolated zones instead of only one secure and one non-secure world. Additionally, it provides a dedicated TEE in machine mode (the Multizone nanoKernel) with a secure inter-zone communication service to monitor switches to other zones. It requires a rv32i, rv32e, or rv64i RISC-V base with the U-mode extension and the Physical Memory Protection.

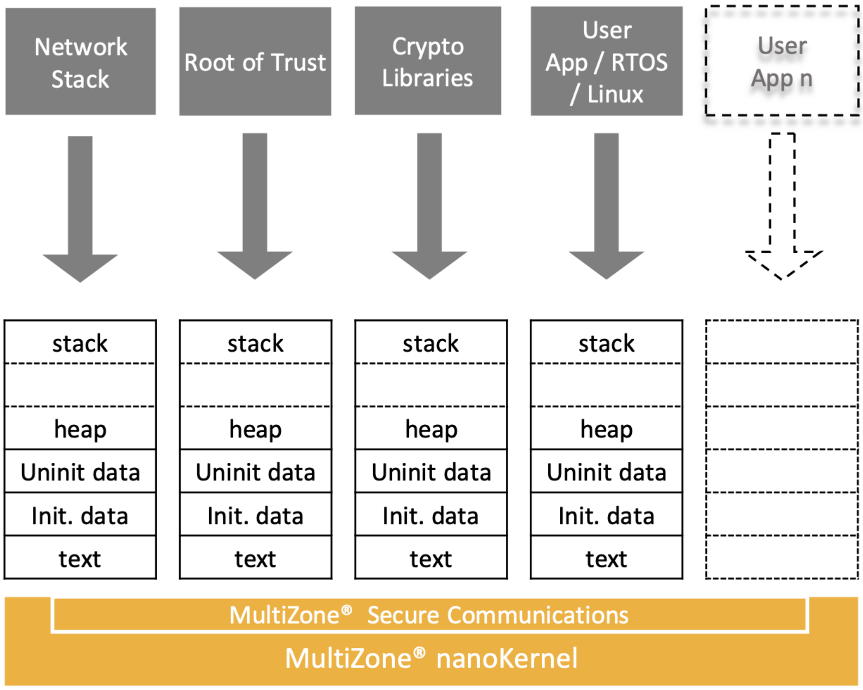


Figure 2.17: Architectural overview of Multizone Security, taken from [26]

An architectural overview of Multizone is provided with Figure 2.17. It is based on the "lightweight, formally verifiable, bare metal" MultiZone nanoKernel [26] with the Multizone Secure Communications messenger, both running in machine mode (M). The Secure Communication is based on messaging without any shared memory. The zones upon the nanoKernel are exemplary filled with a network stack, a Root of Trust (e.g. for key management), cryptographic libraries, an OS (e.g. RTOS) and optional additional user applications. All zones run in user mode (U). Utilizing the machine mode for the nanoKernel and the communication manager, however, results in the fact that the applications in the zones cannot differentiate between user mode and machine mode and hence, do not benefit from this processor extension.

The underlying idea behind Multizone Security is to provide both a software solution and a toolchain to seamlessly integrate operating systems, applications, and libraries into one firmware image that can be flashed to the device (see Figure 2.18). Hence, it follows the idea of having one entity that is responsible for the one executable deployed on the device.

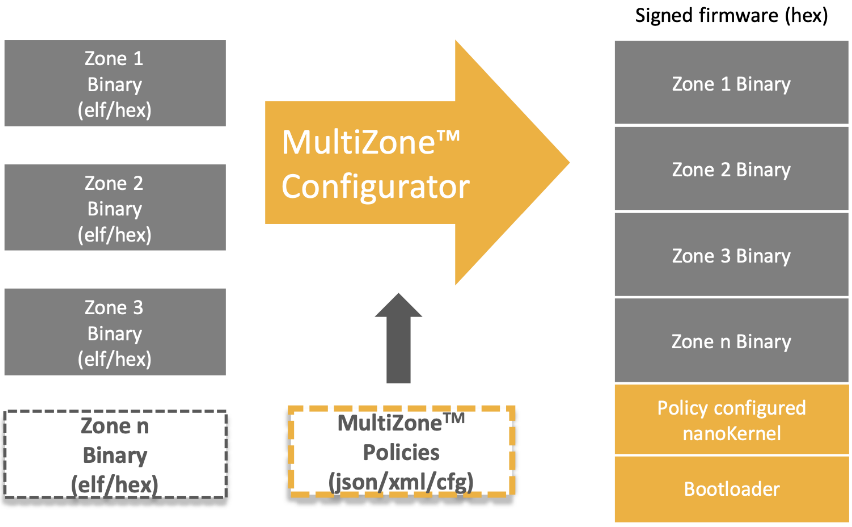


Figure 2.18: Graphical representation of the compiling process of Multizone Security, resulting in one firmware image, taken from [26]

2.5.4 Conclusion

With ARM's Trustzone, a widespread processor architecture for providing a Trusted Execution Environment from industry has been presented. In contrast, the Sancus architecture is a sophisticated approach from academia. The Multizone Security approach for RISC-V is a project for an instruction set architecture that originated in academia but has large partners in industry as well. All three architectural concepts have in common that they provide lightweight hardware-enforced isolation mechanisms for software components of constrained devices in the IoT domain. Nevertheless, they differ in many details.

First, both Trustzone and Sancus only rely on a hardware Trusted Computing Base (TCB). A software provider of a software component running in isolation on a Trustzone or Sancus-based device does not need to trust any other software, not even the operating system. Contrasting, Multizone requires a lightweight monitoring software layer for orchestration purposes and to manage the inter-zone communication.

Second, cryptographic functions implemented in hardware may be available as extensions for the manufacturers of Trustzone-based and Multizone-based SoCs, but are not provided per default for every device. In contrast, Sancus requires a set of three cryptographic primitives per design.

Third, Multizone requires the Physical Memory Protection extension of RISC-V to enable secure isolation of zones. Sancus and Trustzone, on the other hand, do not require an additional unit for memory access control (note, however, that only the optional Memory Protection Unit for Trustzone enables fine-grain differentiation of permissions within the secure world). Additionally, both Sancus and Trustzone divide the memory in secure/protected and non-secure/unprotected regions and derive the processor state directly from the instruction/program counter.

And last, Trustzone and Sancus introduce designated entry instructions for entering the secure/protected region whereas in Multizone, the nanoKernel monitors the scheduling of all zones and only allows for message-based communication between zones.

2.6 Remote Attestation (RA)

The significance of considering security by design in the IoT domain has been emphasized in the previous sections. In Section 2.5, processor security architectures with a hardware RoT that enables TEEs have been presented and discussed. Remote Attestation (RA) is a security service that builds upon and is enabled by such security architectures. It offers a remote verifier the possibility of attesting the state of a connected device [5] and thus evaluating if it can still be trusted [7].

A universal ”Remote Attestation Procedures Architecture” is provided by the IETF as Internet-Draft [7]. This draft is declared as work in progress and hence, it may be subject to change or removal. All information given here are based on the draft version from July 28, 2022.

```
  .--------------------------------.
  |                                |
  |            Verifier            |
  |                                |
  '--------------------------------'
                          ^
                          |
.-------------------------|----------.
|                         |          |
|    .----------------.   |          |
|    | Target         |   |          |
|    | Environment    |   |          |
|    |                |   | Evidence |
|    '--------------+-'   |          |
|                   |     |          |
|                   |     |          |
|           Collect |     |          |
|            Claims |     |          |
|                   |     |          |
|                   v     |          |
|                 .-------+-----.    |
|                 | Attesting   |    |
|                 | Environment |    |
|                 |             |    |
|                 '-------------'    |
|               Attester             |
'------------------------------------'
```

Figure 2.19: Relations of the Target and Attesting Environment within the Attester with the Verifier, taken from [7]

This draft attempts to provide ”a model that is neutral towards processor architectures, the content of claims, and protocols”. The proposed model includes an ”Attester” that is able to produce trustworthy information about itself, hence

"Evidence", to send it to a remote "Relying Party" that can analyse the Evidence (e.g., compare it with an expected attesting result, or consult an external "Verifier"), and decide about the trustworthiness of the Attester. However, an entity is allowed to take on multiple roles simultaneously.

Within the Attester, further differentiation into a Target Environment and an Attesting Environment is proposed as shown in Figure 2.19. Note, that the Relying Party is omitted in that example. However, this separation into two environments is not mandatory for compliance with the architecture. Depending on the implementation, both environments may be merged, or there may be multiple Attesting and Target Environments. Nevertheless, there exists an entity within the Attester that collects the targeted Claims, typically performs cryptographic functions on it, and provides it as Evidence to the Verifier. It is crucial that the Attesting Environment is designed to be robust against malicious modification.

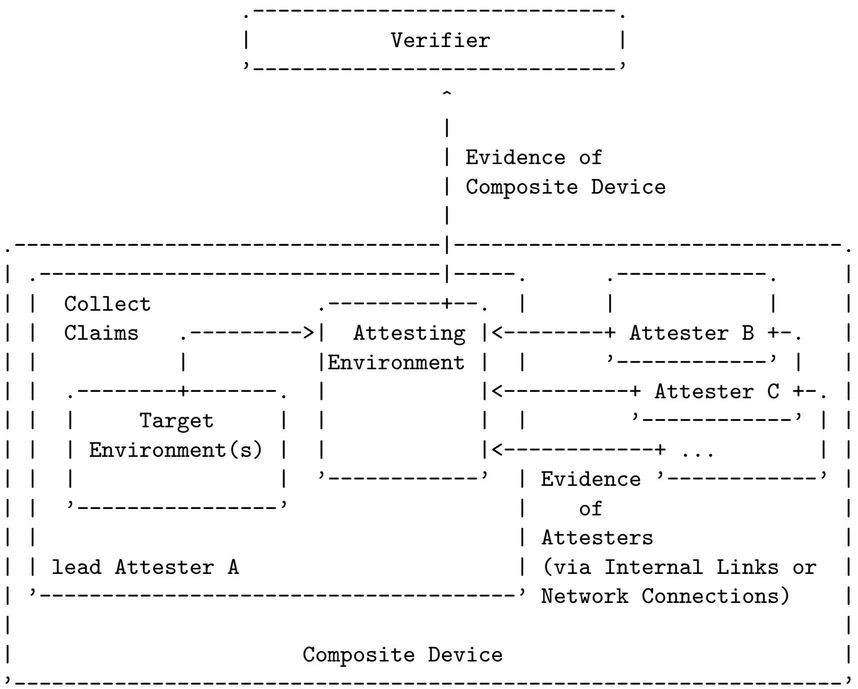


Figure 2.20: Overview of Composite Device with layered approach structure of several Attesting Environments providing Evidence for one lead Attester A, taken from [7]

In more complex scenarios, a layered or "staged" approach can be followed to collect Claims for several Target Environments and provide Evidence in a hierarchy of Attesting Environments. An example for such a "Composite Device" is shown in Figure 2.20. There, the overall trustworthiness of the device is determined by collecting the Evidence from all underlying Attesting Environments (Attester B, Attester C, etc.). In this example, only the leading Attesting Environment provides Evidence of the overall device to the Verifier. Note, that again, the Relying Party is omitted in this example.

```
     .------------.
     |            | Compare Evidence
     |  Verifier  | against appraisal policy
     |            |
     '--------+---'
         ^    |
Evidence |    | Attestation
         |    | Result
         |    v
     .---+--------.              .-------------.
     |            +------------->|             | Compare Attestation
     |  Attester  | Attestation  |   Relying   | Result against
     |            | Result       |    Party    | appraisal policy
     '------------'              '-------------'
```

Figure 2.21: Example topology called "Passport Model" with Attester, Relying Party and Verifier, taken from [7]

```
                             .-------------.
                             |             | Compare Evidence
                             |   Verifier  | against appraisal
                             |             | policy
                             '--------+----'
                                  ^   |
                         Evidence |   | Attestation
                                  |   | Result
                                  |   v
.------------.               .----|--------.
|            +-------------->|---'         | Compare Attestation
|  Attester  |   Evidence    |     Relying | Result against
|            |               |      Party  | appraisal policy
'------------'               '-------------'
```

Figure 2.22: Example topology called "Background-Check Model" with Attester, Relying Party and Verifier, taken from [7]

Two basic examples for the overall system topology are proposed as "Passport Model" and "Background-Check Model" and shown in Figures 2.21 and 2.22, respectively. Both include one entity each for the roles of Attester, Relying Party and Verifier. They differ in the positioning of the Verifier. In the Passport Model, the Attester sends its Evidence to the Verifier first, to then receive the Attestation Result from the Verifier and forward it to the Relying Party.

In contrast, in the Background-Check Model, the Verifier is connected to the Relying Party. The latter requests the Evidence from the Attester, forwards it to the Verifier and receives the Attestation Result in exchange.

The appraisal policy used to appraise Evidence and Attestation Result may consist of a comparison with a reference value.

What is not reflected in the Figures but represents a decisive countermeasure against replay-attacks is the freshness of the Evidence or Attestation Result. Only if freshness information is included in the challenge-response-schema, it is assured that the collected Claims actually reflect the latest state of the Target Environment. This can be solved by sending non-predictable nonces with the request for remote attestation. This nonce is then included in the Claims and attested as Evidence. If both the nonce sent and the nonce received are identical, the Relying Party is assured of the freshness of the attestation.

Concerning the terminology, there is a subtle difference advertised for "trust" and "trustworthiness": "Trust is a choice one makes about another system. Trustworthiness is a quality about the other system that can be used in making one's decision to trust it or not." (see [7], p. 3) Applying this to the roles described above, the Relying Party attempts to assess the trustworthiness of the Attester in order to decide whether it can still be trusted. If it relies on an external Verifier to receive the Attestation Result, is has to trust the Verifier. This trust can be established using a trust anchor, e.g. a certificate, a symmetric key, etc. Likewise, it has to trust the Root of Trust of the Attesting Environment. Concerning the processor security architectures described in Section 2.5, they provide a Hardware Root of Trust in offering Hardware-enabled Trusted Execution Environments. A more detailed discussion of the underlying trust model can be found in [7].

Remote attestation is a decisive building block for trustworthiness assessment. Based on this trustworthiness assessment, further application specific actions may be performed. Hence, Section 4.3.2 discusses the application of the principles of the model and roles described above to the Vulnerability Assessment and Management Solution proposed by this thesis.

2.7 Conclusion

Among the main contributions of this thesis is a holistic architectural concept for a Vulnerability Assessment and Management solution for IoT devices with a Hardware Root of Trust. Due to the scale and complexity of this approach, many large research areas are touched upon. This requires the extensive study of related work in the different research areas, that has been presented with this chapter and that lays the foundation for further design decisions.

Approaches for pre-purchase combined security and privacy labels exist and have been discussed. However, the proposed label designs all focus on pre-purchase information, whereas the aim of the work presented in this thesis is to assess the security state of devices in operation. Although Emami-Naeini et al. acknowledge the relevance of such post-purchase security information [16], it has not yet been the subject of research. Additionally, the use of TEEs is mentioned as best practise, but the label designs themselves do not focus on hardware security architectures that provide a TEE.

Then, assessing the vulnerability of a device requires a concept for representing the software components present on the device. Approaches for appropriate file formats of such software inventory information and for the retrieving procedure have been discussed in this chapter. It can be concluded that such a format has to enable a partitioning of software into Software Modules, it has to provide a universal identifier for each and every Software Module in that very version, and it has to provide a model and a format for its description as SBOM file. Along with the retrieval process proposed by the IETF with the Internet-Draft "Discovering and Retrieving Software Transparency and Vulnerability Information" [34], the SPDX format specified by the Linux Foundation [59] has become apparent to be the best suitable choice.

Approaches for Vulnerability Management solutions can be divided into scanning-based solutions and database-based solutions. Representatives for both classes have been presented. Since scanning-based approaches are a fairly new research area with mainly proprietary solutions available, the focus of this thesis is on database-based approaches. The CVE database is fast growing with tens of new entries per day [9]. It is widely used among many large organizations, but not yet as popular in the IoT domain. Nevertheless, it can be expected that vendors of IoT devices will incorporate CVEs in their product vulnerability management in the future, and has therefore been chosen for the VAM solution proposed by this thesis for linking SBOMs to it.

When the results of such Vulnerability Assessment are then combined with further risk assessment, a numerical security score can be derived from the gathered information. A selection of appropriate IoT security scoring systems has been discussed. Since the scoring system proposed by Alrawi et al. relies on the globally used CVSS, it serves as a basis for the runtime security score proposed with this thesis and described in detail in Section 3.

A very interesting enhancement of such security scoring systems is Trustworthiness Assessment. This requires a concept of trust and the presence of Trusted Execution Environments (TEE). However, in the IoT domain, security architectures that provide TEEs through a Hardware Root of Trust are rare. Influential representatives have been presented.

Remote attestation is a decisive building block for trustworthiness assessment. An attempt for generalizing RA procedures is proposed by the IETF and has been presented. Based on this trustworthiness assessment, further application specific actions may be performed. Hence, Section 4.3.2 discusses the application of the principles of the model and roles to the Vulnerability Assessment and Management Solution proposed by this thesis.

With this extensive literature review, the foundation is provided for the development of an enhanced runtime security score, presented in Chapter 3, and the architectural design of a comprehensive Vulnerability Management Solution for the IoT, presented in Chapter 4.

Chapter 3

Runtime Security Score

In Section 2.1, the idea of including a security star rating in the primary layer of a combined security and privacy label is mentioned. However, a precise specification for the underlying computation of such a rating is not provided.

Instead, such calculation formulae are provided by security scoring systems for the IoT domain as presented in Section 2.4. Although there exists critique on the formally not justifiable CVSS specification (see [55]), such a security scoring system that translates experts' input to a score with respect to relative importance of some aspects over others, can become a de-facto standard and serve as a very valuable basis for further risk assessment.

In extension to the pre-purchase security scoring system based on the CVSS proposed by Alrawi et al. (see Section 2.4), one of the major contributions of this thesis is the development of a runtime security score that combines the findings from Section 2.4 with the possibilities offered by Hardware Root of Trust security architectures (see Section 2.5) and trustworthiness assessment through remote attestation (see Section 2.6). This is a novel approach that enables the user of a smart home network to continuously monitor the security state of operated devices. Aspects adopted from Vulnerability Assessment provides the user with insight into disclosed vulnerabilities of operated devices. The Hardware Root of Trust of these devices provides a Trusted Computing Base in Hardware that further Trustworthiness Assessment can be build upon. And this Trustworthiness Assessment provides the user with insight into the trustworthiness of trusted software components deployed on those operated devices. This represents a holistic approach that provides the user with a profound information base in order to take management decisions. The proposed runtime security score is specified and described in detail in this chapter.

For better comparability, the specification of this security scoring system is provided in relative proportions, instead of absolute numbers. Thereby, the score can easily be adjusted to any total number of points.

The device-related scoring proposed by Alrawi et al. can be adopted as follows. Since it shows a strong asymmetry towards vulnerabilities, a large number of achieved points for this rubric will always result in a poor overall device security score. Additionally, a data basis for device characteristics such as the configuration and pairing process, the upgradeability and the number of exposed services is required to include them in the rating. In contrast to the approach proposed by Alrawi et al., the security scoring system proposed with this thesis is required to be fully automatable. Therefore, an entity responsible for providing an interface to gather this information in a standardized fashion would be necessary. In order to keep the amount of standardization, that is required to put the proposed concept into practice, minimal, the aspects Internet pairing (7% of points), configuration (16% of points), upgradeability (10% of points), and exposed services (10% of points) are excluded and left for future work.

A proposal for incorporating the hereby proposed scoring system into the Vulnerability Assessment and Management (VAM) solution proposed with this thesis is given in Section 4.

3.1 Vulnerability Assessment

Alrawi et al. suggest to perform a one-time Vulnerability Assessment (VA) to gather the CVE records that the software components deployed on the device under consideration are affected by. This VA is then included in the device score with 24 points out of 42 points for the device-related section (hence, with $\sim 57\%$) and 175 points in total and reflects a pre-purchase security rating for consumers.

Since this one-time VA falls into the class of database-based Vulnerability Assessment (see Section 2.3.2), it can be extended to a Continuous Monitoring solution with periodic Vulnerability Assessment and periodic scoring.

In accordance with the proposed scoring system of Alrawi et al., this continuous VA should receive the largest weighting in the security scoring system proposed with this thesis. Disclosed vulnerabilities always pose a risk to the impacted device and with every CVSS severity level, a higher risk can be assumed. Therefore, the weighting proposed by Alrawi et al. is adopted as follows:

Vulnerabilities (100% of points):

- Critical
  - 11 or more critical vulnerabilities (42% of points)
  - 6-10 (38% of points)
  - 1-5 (33% of points)
- High
  - 11 or more highly severe vulnerabilities (29% of points)
  - 6-10 (25% of points)
  - 1-5 (21% of points)
- Medium
  - 11 or more medium severe vulnerabilities (17% of points)
  - 6-10 (13% of points)
  - 1-5 (8% of points)
- Low
  - 11 or more low severe vulnerabilities (12% of points)
  - 6-10 (8% of points)
  - 1-5 (4% of points)

Finally, the question remains as with what frequency the Vulnerability Assessment is to be performed. In [46], Qualys, Inc. suggest to perform vulnerability scans at least on a daily basis. A discussion of this aspect is provided in Section 4.

3.2 Trustworthiness Assessment

Trustworthiness Assessment is an important enhancement of the security scoring system provided by Alrawi et al. The concept of trust is introduced with hardware security architectures that provide Trusted Execution Environments (TEE) based on a Hardware Root of Trust (see Section 2.5) and allows to continuously assess the trustworthiness of a software component with Remote Attestation (RA). As explained in Section 2.6, the RA procedure should be based on a nonce sent from the Relying Party to the Attester in order to guarantee freshness of the Attestation Result and to thereby prevent replay attacks.

Where Qualys CM claims to detect changes on the operating system of connected hosts [46], the objective here is to provide less constrained information about any software module deployed on a device - ranging from firmware over operating system to application software. Therefore, breaches in the Trustworthiness Assessment procedure have far-reaching consequences. A failed Trustworthiness Assessment of a software module always has to be interpreted as a malicious alteration of the software as it was intended by its Software Provider.

In theory, it would be conceivable to take the relevance or ”purpose” of a software component into consideration here. There may be Software Modules that fulfil more critical purposes than others. In practise, however, such considerations can only be applied to an automated scoring system, if such relevance information is available. This would require an instance at the Software Provider that is willing and able to attribute the incorporated software components accordingly. However, the proposed approach can easily be extended to take such information into account, if they are available.

Consequently, it is assumed that all Software Modules incorporated in the Trustworthiness Assessment have the same relevance and their unintended alteration has the same effect on the overall security state. Then, one or more failures in the Trustworthiness Assessment of a device always result in the maximum of all assigned points and break the score computation immediately. This can be combined with warnings or recommendations for action for the user as described in Section 4.

3.3 Relevance Level

Another relevance level that may be taken into consideration is the relevance of the device under consideration to the user. Depending on the device type or its purpose, it may be more or less relevant in its environment. E.g. the proper function of a main door lock is typically more relevant for the user experience than that of a room humidity sensor.

However, this is highly individual risk assessment and therefore, it requires user configuration. For some classes of devices, generally accepted rules for differences in their relevance to the user experience may be found, but there definitely is no universal ranking of device relevance levels. Therefore, the user is required to configure its individual risk assessment during the initial device connection process. This coincides with Qualys, Inc., who suggest to assign custom weighted values to assets to denote ”the business value of critical assets” [46].

The relevance level of the device under consideration for the entire system or for the user experience shall not directly factor into the runtime security score. An extensive and undisturbed security report with the bare results of the proposed security state assessment may provide valuable information to the user. Instead, it is proposed to make the warnings and recommendations for action depending on the individual relevance level of this device. Thereby, a device whose security state is highly relevant to the user may receive a recommendation for immediate shutdown once a pre-defined threshold is exceeded. A detailed discussion of these management aspects can be found in Section 4.

3.4 Non-Numerical Representation

The hereby proposed runtime security scoring system results in a numerical representation of the assessed security state of the IoT device. However, it is a common assumption that humans find it difficult to attach meaning to numerical values. Therefore, alternative representations are suggested.

Loi et al. suggest a three-level rating based on the letters $A - C$, similar to the EU energy labels for electrical devices [35]. However, this rating is solely based on expert opinion. Although Alrawi et al. also suggest a letter-based rating, they propose a mapping based on general cutoffs as explained in Section 2.4.

Such general cutoffs could also be represented in a star rating as suggested by Emami-Naeini [18]. However, the determination of a representation that the majority of users responds best to is a matter of research in the area of human-centered design and, hence, considered out of scope for this thesis.

Therefore, for this initial proposal, a representation of the numerical scoring result as a traffic light with the three evenly distributed levels and colors RED (for a score from 66.1% to 100%), $YELLOW$ (for a score from 33.1% to 66%), and $GREEN$ (for a score from 0% to 33%) is suggested. This scheme of colors is very well known from actual traffic lights and food nutrition facts labels, and therefore, it is assumed that the user can interpret the different colors very well. However, a detailed survey of such effects on the user is subject to future work.

3.5 Conclusion

To conclude, the runtime security scoring system proposed with this chapter is based on the scoring system proposed by Alrawi et al., but with decisive extensions. Thereby, it represents a major contribution in this research area. It focusses on the proposed Vulnerability Assessment and extends it towards Continuous Monitoring. Thereby, the user is enabled to continuously monitor the security state of all operated devices and provided with insight into disclosed vulnerabilities.

Additionally, it utilizes the concept of Trusted Execution Environments to include Trustworthiness Assessment. Since the trustworthiness of all software components on a device is assumed to be critical, a failed Trustworthiness Assessment of one Software Module always results in the maximum point value in order to represent the poorest security state. This provides the user with insight into the trustworthiness of trusted software components deployed on operated devices.

This assessment of the security state lays the foundation and can be combined with individual risk assessment in order to provide the most useful warnings or recommendations for actions to the user.

Lastly, the resulting numerical security score is represented as a three-level traffic light. This is a reasonable choice for the non-numerical representation, since traffic lights are commonly known from everyday life and can be interpreted easily by the user.

Chapter 4

Vulnerability Management Solution for IoT

In addition to the Runtime Security Index for IoT devices described in Chapter 3, a process for retrieving software inventory information, a format for SBOM files, and an architectural concept for a complete Vulnerability Management Solution for the IoT are further main contributions of this thesis. The following Section 4.1 provides a system model definition which forms the basis for further work. The Device and Software Module Management is then described in Section 4.2.

Continuous Monitoring functionality provides the information necessary for the computation of the Runtime Security Index and is described in Section 4.3. Its main building blocks are the Vulnerability Assessment procedure described in Section 4.3.1 and the Trustworthiness Assessment procedure described in Section 4.3.2.

The actual computation procedure of the Runtime Security Index for IoT devices at runtime is then described in Section 4.4 and followed by the description of further risk assessment in Section 4.5 on Vulnerability Management.

4.1 System Model Definition

The System Model presented here is an extension of the System Model proposed by Noorman et al. in [42] in 2017 and serves as the basic architectural concept for the Vulnerability Management solution presented in this thesis. Figure 4.1 shows a high-level overview of the overall system architecture and the components involved.

As outlined in this high-level overview, most consumer smarthome systems come with some kind of dedicated control unit. Depending on the vendor, this component may be called controller (e.g. Bosch's Smart Home Controller), gateway (e.g. IKEA's TRÅDFRI gateway), bridge, hub, base (e.g. Telekom's Magenta SmartHome router or SmartHome Home Base), or base station (e.g. IETF RFC 7228 [10]). Other cloud-based solutions exist (e.g. Amazon's Alexa, Samsung's Smart Things), but are not the focus of this work. Nevertheless, the additional functionality proposed by this work could be adapted to such smarthome systems as well.

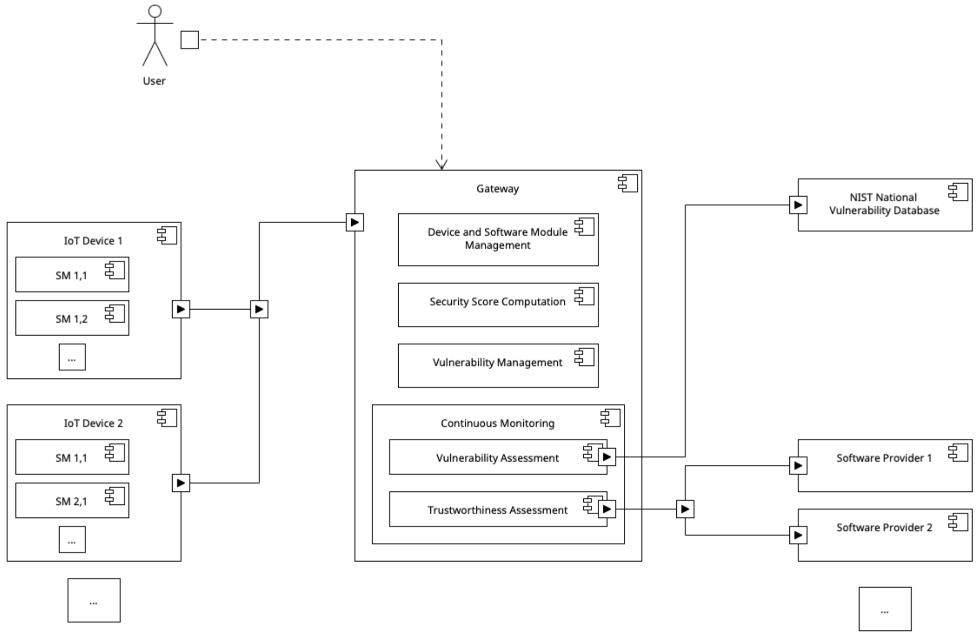


Figure 4.1: High-level overview of the overall system architecture

For generalization, this dedicated control unit will be called gateway in the following. Typically, it is far less resource-constrained than the IoT devices connected to it, as described by the IETF in [10]. Hence, it comes with more computational power, more memory, more interfaces (most notably, the full TCP/IP-protocol stack and often a range of wireless communication protocols, e.g. Wi-Fi, Bluetooth Low Energy (BLE), Zigbee) and a larger battery or direct connection to the domestic power supply. This makes it a vital link between the connected IoT devices and the user and opens up a wide range of possibilities. Especially for constrained IoT devices, it is common that they only support very lightweight wireless communication protocols such as BLE or Zigbee. Hence, any access from the Internet has to be realized via the gateway. This is actually beneficial from a security perspective as security measures can be put in place at the gateway more easily due to the availability of more powerful resources.

Apart from automated control functions, some kind of graphical user interface is typically provided to enable the user to access the network and comfortably perform management functions. Whether this access is enabled through a smartphone application, a browser interface, or voice control is considered irrelevant for this work. All details concerning user-centered design decisions make up a whole branch of research and are out of scope here.

Software Providers can deploy Software Modules ($SM1,1$, etc. in Figure 4.1) on the connected IoT devices. During the initial connection setup, these software modules are registered at the gateway. Subsequently, the gateway is responsible for further device and software module management and tracks available updates for all deployed software modules using the depicted interface to the Software Providers as described in detail in Section 4.2.

It is a reasonable assumption that a commercial IoT device is shipped with an initial set of Software Modules and not as general purpose machine that can run different firmware and/or software chosen by the user. Additionally, in the IoT domain, all software on the device is usually provided by one entity that merges firmware, application software, third-party software, and libraries into one binary that is flashed on the device before shipment [54]. In the context of this thesis, this entity is called "Manufacturer" as proposed in [33]. This may indeed be the hardware manufacturer of the retail customer IoT device. In case of a less constrained device with a built in Linux stack, this could also be the entity integrating the device. However, it is a reasonable assumption that one such entity exists for every device.

To enable the management of Software Modules, a standardized description for them that is complied with by all components is a fundamental requirement. Such descriptions have been discussed in detail in Section 2.2. The specific underlying information model in SPDX format is presented in the upcoming Section 4.2.1. The necessary device registration process with SBOM retrieval is then presented in the upcoming Section 4.2.2.

To summarize this system model definition, the results of this work can be adapted to any smart home system with the following features:

1. It includes exactly one gateway that provides the following components as specified in the upcoming sections: Device and Software Module Management, Continuous Monitoring, Security Score Computation, Vulnerability Management. It can be assumed that a future extension towards systems with two or more interconnected and cooperating gateways is possible with moderate effort.
2. It provides any kind of graphical user interface to the user.

3. It supports one or more constrained IoT devices connected to the gateway via standard wired or wireless connection protocols.

4. There exists one standardized description for Software Modules that is complied with by all components and provides fields equivalent to the VAM SBOM format described in Section 4.2.1.

5. There exists an entity called Manufacturer that provides such an SBOM file in a way that is compliant with the procedure specified in Section 4.2.2.

6. There exists a specified interface to a Vulnerability Database, e.g. the NIST National Vulnerability Database (NVD) for CVE records.

4.2 Device and Software Module Management

Device and Software Module Management lays the foundation for Vulnerability Assessment and Vulnerability Management. For the following considerations, the focus is dedicated to the main Software Module Management tasks: the check for newly disclosed vulnerabilities and the check for trustworthiness of that Software Module. All other common tasks of operating a smart home IoT device, e.g. the pairing of the device with the gateway on host-to-network-layer of the TCP/IP layer model, are not considered here as they are irrelevant to the Vulnerability Management Solution that this thesis focusses on.

As described in Chapter 2.2, a standardized description of the software inventory is a fundamental requirement for the Vulnerability Assessment and Management System proposed by this thesis. Hence in the following, it is assumed that a description in accordance with the proposal described in Section 4.2.1 exists and all components agree upon it. Since the Manufacturer is the entity with complete insight into all Software Modules deployed on the device, it is the component that provides this software inventory information. The Manufacturer could do so by collecting SWID tags provided by the SWID tag producers, e.g. the Software Providers, but details of this process are not the focus of this thesis.

During the initial connection of a new IoT device to a Gateway (the device registration) this software inventory information needs to be transmitted to the Gateway. Only afterwards, the Software Modules can be appropriately managed: they can be linked against a vulnerability database, the manufacturer can be queried for more recent software updates or patches, and the IoT device can be updated to that new software version. The proposed procedure for device registration focussing on these Software Module Management tasks is described in detail in Section 4.2.2.

4.2.1 Software Inventory Information

Since the Vulnerability Management solution proposed in this thesis is supposed to be suitable for constrained IoT devices, the software inventory information file is not supposed to be stored directly on the device. Therefore, its file size is not crucial. Due to its compatibility with the SBOM retrieval mechanism proposed by the IETF and its straightforward implementation, the SPDX format proposed by the Linux Foundation is a well-suited choice as SBOM file format in the context of this thesis.

Lots of terms in this research area exist. This ambiguity has already been discussed in chapter 2.2. For the remainder of this thesis, the term ”Software Bill of Material (SBOM)” describes the Software Inventory Information needed to reliably discriminate a certain Software Module with designated version number, components, and licenses. The latter is mentioned for the sake of completeness, but is not the focus of this thesis.

In order to meet the requirements of the Vulnerability Management solution proposed in this thesis, a new SBOM file format is specified based on the SPDX format. In the following, this format is called ”VAM SBOM”. Table 4.1 shows all fields that are mandatory in this format. For full compatibility with the SPDX format, the $tag : value$ notation and the tag-names have been adopted. Note, however, that all indicated fields are mandatory in the VAM SBOM, whereas some of them are optional in the original SPDX format. The VAM SBOM file must be a flat text file with $*.spdx$-extension.

The $DocumentNamespace$ described in Table 4.1 must be composed as follows: `https://[CreatorWebsite]/[pathToSpdx]/[PID]-[UUID]-[version].spdx`, where:

- $CreatorWebsite$ refers to the domain of the entity providing the SPDX file;
- $pathToSPDX$ contains the path to the SPDX file;
- PID contains the product ID;
- $UUID$ contains a Universally Unique Identifier (UUID) that is unique for this specific SPDX file version;
- $version$ contains the software binary version of the Software Module.

SPDX field name [59]	**Description**
Document Creation Information	
SPDXVersion	Provides a reference number for parsing the SPDX file. Must be set to ”SPDX-2.2”.
SPDXID	Provides an identifier for the current SPDX file. The data format for this field is $SPDXRef-DOCUMENT$.
DocumentName	Provides an additional identifier for the current SPDX file that is easier to refer to than the $SPDXID$.
DocumentNamespace	Provides a unique absolute URI in https-scheme that serves as document specific namespace for the specific SPDX in that very version. Since in the context of this thesis, the SPDX file is accessible on the Internet, this is a URL. The format for composing this URI is given above.
Creator	Identifies the person or entity creating the SPDX file.
Created	Provides date and timestamp for the SPDX file creation. The data format for this field is $YYYY-MM-DDThh:mm:ssZ$.
Package Information	
PackageName	Provides a textual identifier for the Software Module in the SPDX file.
SPDXID	Provides a unique identifier for the Software Module in the SPDX file. Since in the context of this thesis, the SPDX file is accessible on the Internet, the format of this field is $[DocumentNamespace]\#SPDXRef-[idstring]$, where *idstring* is a unique string containing letters, numbers, ’.’ and ’-’.
PackageVersion	Identifies the specific version of the Software Module.
PackageDownloadLocation	Provides the URL for downloading the Software Module in the specified version from the Manufacturer webserver.
FilesAnalyzed	Provides a boolean indicator of whether the Software Module has been analzed. Must be set to ”true”.

File Information	
FileName	Provides the full path (relative to the root of the Package) and the full file name (including the file type) for every file belonging to the Package.
SPDXID	Provides a unique identifier for the file. Since in the context of this thesis, the SPDX file is accessible on the Internet, the format of this field is $[DocumentNamespace]\#SPDXRef-[idstring]$, where *idstring* is a unique string containing letters, numbers, '.' and '-'.
FileChecksum	Provides a checksum for every file belonging to the Package. Providing a SHA1-checksum is mandatory, other optional algorithms may be: SHA224, SHA256, SHA384, SHA512, MD2, MD4, MD5, MD6.

Table 4.1: Overview of all fields of the VAM SBOM format, adopted from the SPDX 2.2 specification, and their description [59]. For better readability, fields related with licensing are omitted, even if they are mandatory for SPDX 2.2 compliance. Fields which are mandatory according to the SPDX 2.2 specification are written in bold letters. Note, however, that for compliance with the VAM SBOM specification presented in this thesis, all fields are mandatory.

In the initial version of the VAM SBOM format described here, the *PackageDownloadLocation* must be the URL to the Manufacturer webserver. An extension towards version control systems is provided by the underlying SPDX format and planned as independent future extension for the VAM SBOM format.

An example for a VAM SBOM file as specified above is provided in the following listing:

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-RISC-V
DocumentName: risc-V
DocumentNamespace: https://SoftwareProvider.org/spdxdocs/devices/
    risc-v/PhysicalMemoryProtection/sifive-hifive1-revB-01-2.11.1
Creator: Organization: Software Provider A
Created: 2022-01-01T18:30:22Z

PackageName: freeRTOS
SPDXID: https://SoftwareProvider.org/spdxdocs/devices/risc-v/
    PhysicalMemoryProtection/
    sifive-hifive1-revB-01#SPDXRef-freeRTOS-2.11.1
PackageVersion: 2.11.1
```

```
PackageDownloadLocation: https://SoftwareProvider.org/downloads/
    devices/risc-v/PhysicalMemoryProtection/sifive-hifive1-revB-01.tar
FilesAnalyzed: true
PackageLicenseConcluded: LGPL-2.0
PackageLicenseDeclared: LGPL-2.0
PackageCopyrightText: <text>
                        Copyright 2020-2022 Jane Roe
                        </text>

FileName: ./bin/foo.
SPDXID: https://SoftwareProvider.org/spdxdocs/devices/risc-v/
    PhysicalMemoryProtection/
    sifive-hifive1-revB-01#SPDXRef-freeRTOS-2.11.1-foo
FileChecksum: SHA1: d6a770ba38583ed4bb4525bd96e50461655d2758
```

Concerning this initial version of the VAM SBOM format described here, the CoSWID-format was considered irrelevant due to the fact that the SBOM file is not directly received from the device. Nevertheless, this could be a beneficial extension for more capable devices. If the device is able to transmit its SBOM file directly, the Manufacturer would not need to host a dedicated webserver. As demonstrated in Section 2.2.1, most SPDX format fields have equivalents in the SWID and CoSWID formats, respectively. However, there are mandatory fields in the SWID/CoSWID-formats that do not have obvious equivalents in the SPDX format. Hence, an extension of the VAM SBOM format towards SWID/CoSWID-compliance requires an appropriate mapping. The support of yet another SBOM format is not crucial for proving the general feasibility of the architectural concept, however, and is therefore planned for future work.

To conclude, the presented VAM SBOM format is suitable for constrained IoT devices that are not capable of sending their SBOM file themselves. Putting additional licensing information aside, it requires only 14 mandatory fields to precisely and unambiguously describe and identify Software Modules. In the initial version presented here, it only requires a flat text file, which enables a simple parsing process. It is compliant with the SPDX 2.2 specification and with the SBOM retrieval mechanism proposed by the IETF. The complete proposed SBOM retrieval mechanism during the device registration is presented in the following section.

4.2.2 Device Registration

The device registration at the gateway with the transmission of complete software inventory information is an important building block for all further Software Module Management tasks. When a new IoT device enters the smart home network, it has to be set up at the gateway. For the following considerations, the focus is dedicated to the main Software Module Management tasks: the check for new vulnerabilities and the check for trustworthiness. As explained above, all other common tasks of operating a smart home IoT device, e.g. the pairing of the device with the gateway on host-to-network-layer, are not considered here.

In summary, the requirements for the proposed device registration procedure are as follows:

1. There exists a communication interface between the gateway and the constrained IoT device. Details of the communication protocol do not matter. It may be a wired serial interface, a wifi connection or one of the in the IoT domain widely used Zigbee, Zwave or Bluetooth Low Energy (BLE) protocols. Since System-on-Chips (SoC) with Hardware Root of Trust functionality that enables Trusted Execution Environments are a fairly new development, they have not yet found their way into retail customer smart home devices. Therefore, hardware development boards that are available for research purposes most often do not provide a large variety of communication interfaces. The modularity of the concept and of the reference implementation proposed in this thesis allows the network connection to be expanded with the release of corresponding hardware.

2. It is assumed that the IoT device has been connected to the same local network that the gateway operates in. This is reasonable, since the pairing of the device with the gateway is considered out of scope for this work as described above.

3. There exists an entity called Manufacturer that provides a VAM SBOM file for the IoT device in accordance with the procedure specified in the following.

4. The IoT device is associated with a product ID, a serial number, an URL to the Manufacturer webservice that provides the software inventory information, and a version of the binary by the Manufacturer before shipment.

Based on these requirements, the proposed procedure for the device registration is specified in the following. Figure 4.2 shows the exchanged messages for the error-free case accordingly.

Please remark that the following specification for discovering and retrieving software inventory information is in accordance with the IETF Draft "Discovering and Retrieving Software Transparency and Vulnerability Information" and falls into the second category described thereby: "objects may be found in one of three ways: [...], on a web site (e.g., via URI), [...]" [34].

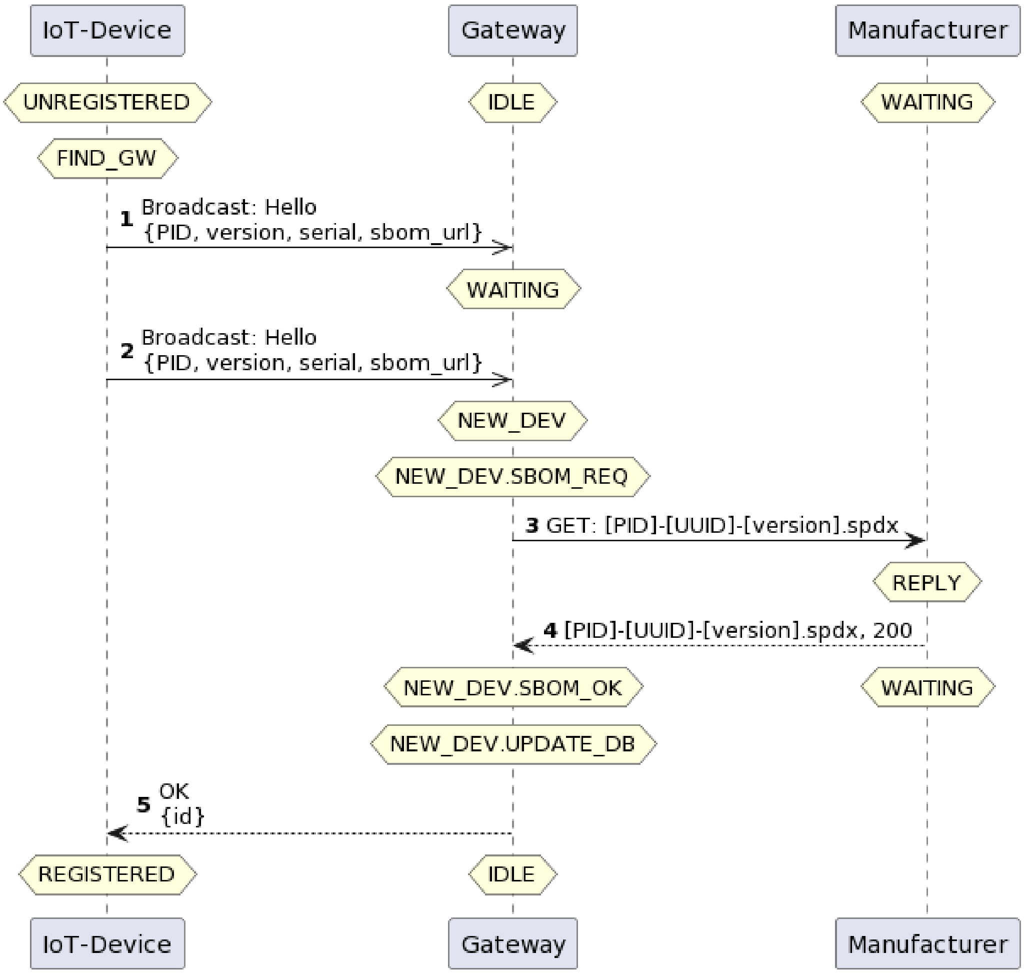


Figure 4.2: Message sequence chart showing the device registration procedure

The new IoT device initially is in the state *unregistered*.

1. Then, it starts searching for a reachable gateway in the network, e.g. due to a user input, by sending a broadcast *hello*-message. This *hello*-message includes the **device identification information**, namely the product ID PID, the software binary version $version$, a serial number $serial$ and the sbom_url to the Manufacturer's webserver that provides the software inventory information $CreatorWebsite$. The PID, $version$, and $serial$ are all assigned by the Manufacturer and used to differentiate products and software versions at the webserver to provide accurate software inventory information.

2. Initially, the gateway is in *idle* mode and becomes ready for pairing with a new device, e.g. due to a user input. Then, it is in the state *waiting*. As soon as the gateway receives the broadcast *hello*-message in the *waiting* state, it

changes to *new_dev*. The device identification information is then stored in a local database and used to compose the request for the Manufacturer's webserver. In accordance with the VAM SBOM format specified in Section 4.2.1, the webserver's address is derived from the *DocumentNamespace* field and has the following format:
`https://[CreatorWebsite]/[pathToSpdx]/[PID]-[UUID]-[version].spdx`.

3. The gateway then changes to state *new_dev.sbom_req* and sends the request as https-*GET* message to the composed address. From the *PID* and *version* in the url, the webserver can differentiate different products and different software binary versions and selects the appropriate VAM SBOM file.

4. Initially, the Manufacturer webserver is in state *waiting*, waiting for new requests. Upon receiving the *GET* message from the gateway, it changes to *reply* state, answers the request with the appropriate VAM SBOM file and http-*OK*-state and changes back to *waiting* state again.

   The gateway receives the VAM SBOM file, changes to *new_dev.sbom_ok* and analyses the received VAM SBOM file. At this point, additional security measures such as certificates and encryption of the VAM SBOM file may be implemented to guarantee authenticity and integrity but are considered out of scope for this thesis.

5. If the analysis succeeds, the gateway changes to *new_dev.update_db* state and updates its local database with the information parsed from the VAM SBOM file. Subsequently, an *OK*-message with a device ID assigned by the gateway is sent back to the IoT device. The gateway changes back to *idle* state.

   Finally, the IoT device stores the received device ID locally and changes to *registered* state.

Note, that the gateway's *idle* state could be any state where the gateway performs tasks that are not related to the procedure described above. Analogously, the IoT device can leave the *registered* state and start performing tasks that are related to its purpose as smart home IoT device.

The specification above strictly defines the format of the url to the Manufacturer webserver. In order to achieve broad acceptance and standardization across many manufacturers, it is always preferable to give as few restrictions as possible.

Hence, the given specification can easily be expanded to support other interfaces between the gateway and the manufacturer. As proposed by [34] and described in Chapter 2.2, this interface may not only support http, but there could be a more sophisticated API established. And if the device was able to directly transmit its SBOM from its local storage itself, a Manufacturer webserver would even be dispensable. All of this is a valuable but independent future extension.

After the device is properly set up at the gateway, it can take up its dedicated function as smart home IoT device. All further Software Module Management tasks are taken care of by the gateway as described in Section 4.3.

4.3 Continuous Monitoring

In addition to the basic Software Module Management functionality described above, the gateway performs several background checks to implement continuous monitoring and to lay the foundation for the subsequent security score computation. These include the Vulnerability Assessment and Trustworthiness Assessment as main building blocks, which are described in detail in the following paragraphs.

4.3.1 Vulnerability Assessment

Vulnerability Management (VM) is one of the gateway's main features the proposed software architecture provides. As discussed in Section 2.3, VM requires Vulnerability Assessment (VA) in the first place. Apart from more sophisticated approaches that can hardly be automated, such as reverse engineering, a widely used starting point for automated VA is the Common Vulnerabilities and Exposures (CVE) database for publicly disclosed cybersecurity vulnerabilities.

During operation, the gateway is responsible of finding new CVE records that are related to the software modules deployed on any connected device. To realize this, the gateway has to provide the following functionality.

Searching new CVE IDs

First, the gateway periodically performs a search for new CVE IDs related to any of the deployed software modules. Since an automated classification and assignment of CVE records to a class of IoT devices or even specific device models has been proven difficult by Khoury et al. in [31] and Blinowski et al. in [9] (see Section 2.4), an entity that is responsible for manually assigning CVE records to specific IoT device models and versions is required. As specified in Section 4.1, it is assumed that there exists an entity that merges firmware, application software, third-party software, and libraries into one binary that is flashed on the device before shipment: the Manufacturer. Therefore, the Manufacturer is also the entity responsible for assigning CVE records to its Software Modules and providing the corresponding Vulnerability Information.

Thus, a well-defined interface to the Manufacturer is introduced here. This requires standardization, but comes with the great advantage of a significantly reduced complexity. Additionally, there already exist standardization efforts concerning Software Bills of Material (see Section 2.2). Hence, it is a feasible assumption that such a standard can be instantiated or extended accordingly.

This interface to the Manufacturer can be defined in two different ways.

First case: If compliance with the SPDX format is not mandatory, the VAM SBOM format specified in 4.2.1 can be extended with a $VulnerabilityInformationDownloadLocation$ field in the Package Information-area. This field provides the URL for downloading the Vulnerability Information for this Software Module in the specified version from the Manufacturer webserver. The corresponding message sequence chart is based on the Device Registration procedure described above and shown in Figure 4.3.

0) Message 0) represents the last message from the Manufacturer to the Gateway during the device registration process described in Section 4.2.2 and shown in Figure 4.2. Note, that all previous messages of the device registration process are omitted here for better readability.

1) Initially, the Gateway is in *idle* mode and starts searching for new Vulnerability Information based on the configurable frequency $freq_{newCVE}$ described below. Based on the $VulnerabilityInformationDownloadLocation$ received with the VAM SBOM during device registration, the Gateway changes to state $vulninf\_req$ and sends a https-GET message to the corresponding address.

2) Initially, the Manufacturer webserver is in state *waiting*. Upon receiving the GET message from the gateway, it changes to *reply* state, answers the request with the appropriate Vulnerability Information file and http-OK-state and changes back to *waiting* state again.
Upon receiving the Vulnerability Information from the Manufacturer webserver, the gateway changes to $vulninf\_rec$ state, analyses the file and updates its local database with the information parsed from the Vulnerability Information file. At this point, additional security measures such as certificates and encryption of the Vulnerability Information file may be implemented to guarantee authenticity and integrity, but are considered out of scope for this thesis.

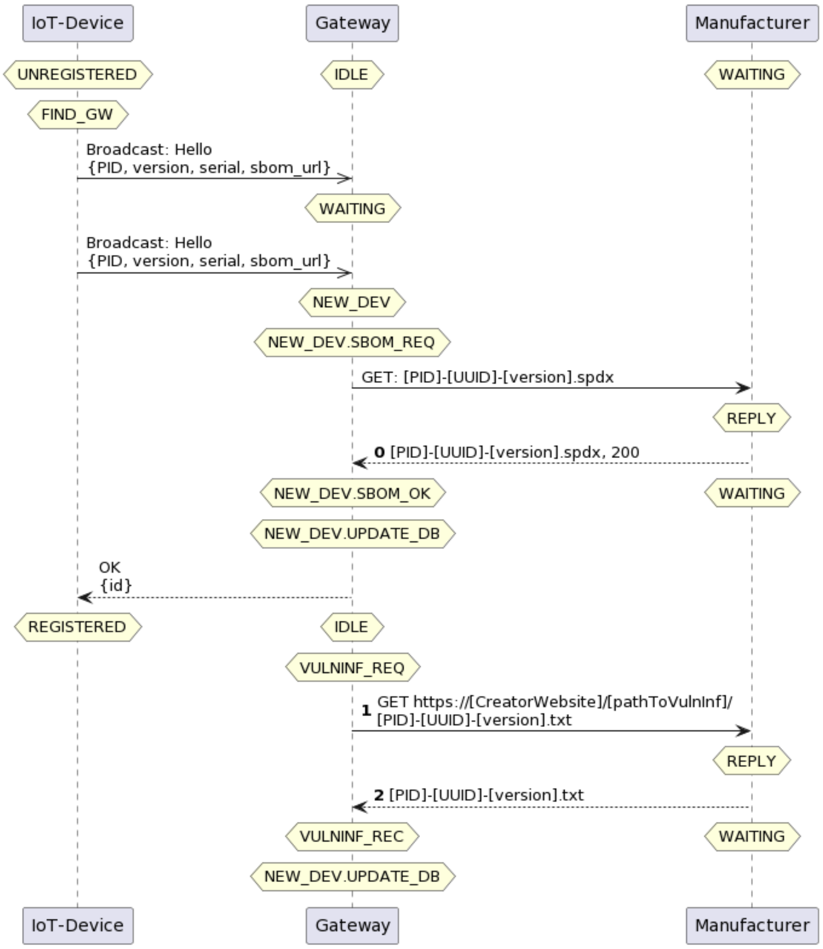


Figure 4.3: Message sequence chart showing the download procedure of the Vulnerability Information for case 1: The VAM SBOM includes the $VulnerabilityInformationDownloadLocation$ field for reaching the Manufacturer

> Afterwards, the gateway proceeds with retrieving the CVE records corresponding to the received Vulnerability Information as described below.

Note, that the message 0) only has to be exchanged once during the Device Registration process described in Section 4.2.2. The sequence of messages 1) and 2) is issued periodically during the continuous monitoring process with frequency $freq_{newCVE}$ for every connected IoT device.

Second case: If compliance with the SPDX format is necessary, the IoT device has to provide a dedicated interface for retrieving the $VulnerabilityInformationDownloadLocation$. The corresponding message sequence chart is shown in Figure 4.4.

Note, that the messages 0.1) and 0.2) only have to be exchanged once. Hence, they extend the device registration process described in Section 4.2.2.

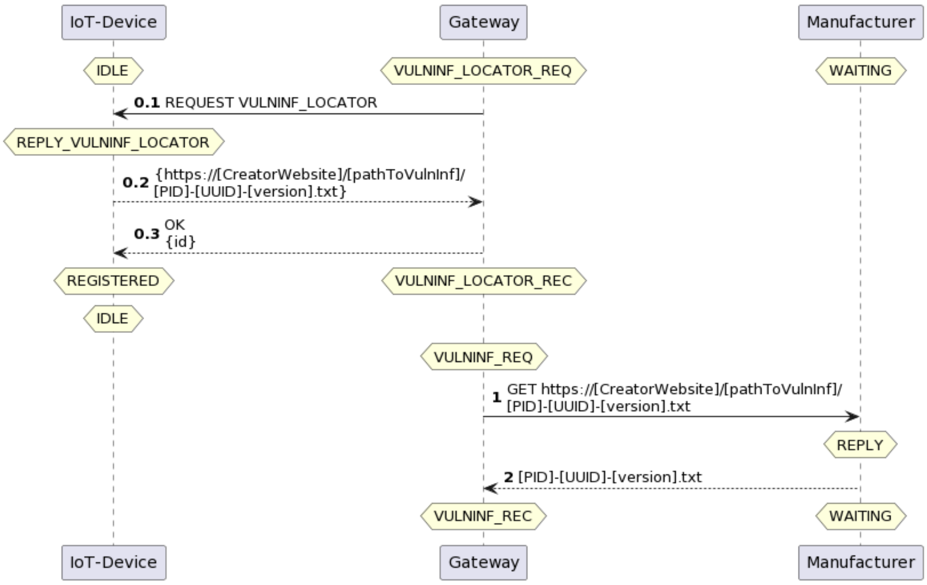

IoT-Device
Gateway
Manufacturer
IDLE
VULNINF_LOCATOR_REQ
WAITING
0.1 REQUEST VULNINF_LOCATOR
REPLY_VULNINF_LOCATOR
0.2 {https://[CreatorWebsite]/[pathToVulnInf]/
[PID]-[UUID]-[version].txt}
0.3 OK
{id}
REGISTERED
VULNINF_LOCATOR_REC
IDLE
VULNINF_REQ
1 GET https://[CreatorWebsite]/[pathToVulnInf]/
[PID]-[UUID]-[version].txt
REPLY
2 [PID]-[UUID]-[version].txt
VULNINF_REC
WAITING
IoT-Device
Gateway
Manufacturer

Figure 4.4: Message sequence chart showing the download procedure of the Vulnerability Information for case 2: The VAM SBOM does not provide the $VulnerabilityInformationDownloadLocation$. Instead, the IoT device provides a dedicated interface to answer requests from the Gateway with the Manufacturer's webserver address

0.1) Message 0.1) replaces the last OK-message of the device registration process shown in Figure 4.2. Thereby, the Gateway changes to $vulninf\_locator\_req$ state and sends a $REQUEST\_VULNINF\_LOCATOR$-message to the IoT device. Note, that all other messages of the device registration process are omitted here for better readability.

0.2) Upon receiving the $REQUEST\_VULNINF\_LOCATOR$-message from the Gateway, the IoT device changes to state $reply\_vulninf\_locator$ and answers with the $VulnerabilityInformationDownloadLocation$.

0.3) Upon receiving the $VulnerabilityInformationDownloadLocation$, the Gateway responses with the original OK-message of the device registration process shown in Figure 4.2, changes to state $vulninf\_locator\_rec$, and updates its local database accordingly.

1)-2) The messages 1) and 2) are equivalent to the ones in Figure 4.3.

Note, that the IoT device's and the Gateway's *idle* states could be any states where both entities perform tasks that are not related to the procedure described above.

In both cases, the Manufacturer webserver's address is derived from the *DocumentNamespace* field in the VAM SBOM and has the following format: `https://[CreatorWebsite]/[pathToVulnerabilityInformation]/[PID]-[UUID]-[version]`.

For the initial version described in this thesis, the Vulnerability Information is a plain string including all CVE IDs related to the Software Module separated with semicolon. For expandability in the future, this can be replaced with a more sophisticated data format, such as json, supporting other vulnerability databases apart from the CVE database, e.g. the Open Source Vulnerability Database [43].

For both cases, the well-defined interface to the Manufacturer is thereby specified and the Gateway's local database is updated with all CVE IDs that are currently assigned to the Software Module deployed on the IoT device under consideration.

Retrieving CVE records

The CVE records are then retrieved from the API provided for the National Vulnerability Database by the U.S. National Institute of Standards and Technology [38] with mention of the CVE IDs.The corresponding message sequence chart is shown in Figure 4.5.

1) The Gateway starts in *cve_req* state and sends a https-*GET* message to the webserver provided by NIST at the address `https://services.nvd.nist.gov/rest/json/cve/1.0/[CVE_ID]`.

2) The NIST webserver answers the request with the corresponding CVE record in json format. Upon receiving, the Gateway changes to *cve_rec* state, analyses the file and updates its local databases accordingly. Afterwards, it changes to *idle* state.

Frequency-based Vulnerability Assessment

Let the time interval at which this search for new CVE IDs is performed be configurable by the choice of frequency $freq_{newCVE}$. According to [46], vulnerability scans should be performed at least on a daily basis, which therefore serves as default value. Note, that the procedure for searching new CVE IDs as described in Section 4.3.1 is necessary for every connected IoT device. Accordingly, retrieving

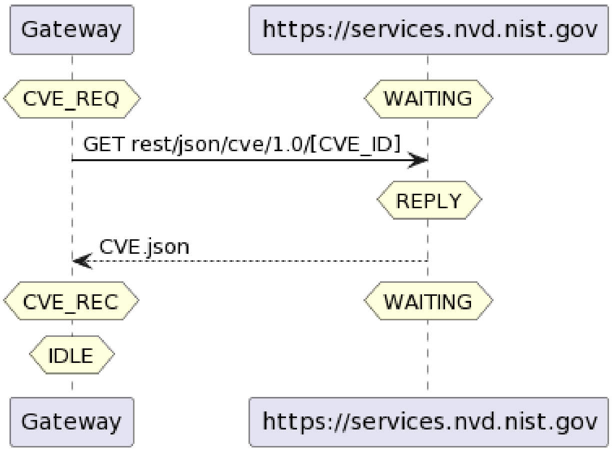

Gateway
https://services.nvd.nist.gov
CVE_REQ
WAITING
GET rest/json/cve/1.0/[CVE_ID]
REPLY
CVE.json
CVE_REC
WAITING
IDLE
Gateway
https://services.nvd.nist.gov

Figure 4.5: Message sequence chart showing the download procedure of the CVE record in json format

the corresponding CVE records as described in Section 4.3.1 is necessary for every such CVE ID. Introducing this kind of periodical search for new vulnerabilities makes the proposed system a solution providing Continuous Monitoring as described in Section 2.3.

Finally, whenever the internal database of all deployed Software Modules is changed (e.g. because a new IoT device is registered at the gateway), an exceptional search for all CVE records concerning this module is performed. Afterwards, this module is covered by the periodical routine described above.

It can be argued that a very sophisticated user may want to have additional functionality. Hence, the user can be provided with the possibility of manually triggering an immediate search for new CVEs. Additionally, the user can be enabled to enter CVEs and manually assign it to a certain type of device or a Software Module to overcome the issue of a not properly working manual assignment. Another additional feature could be an ignore list for CVE records the user specifically wants to exclude from the vulnerability check. Certainly, this option is clearly not recommended and poses the risk of severe failure of the VA process for inexperienced users. All these extensions and their analysis are considered future work.

The proposed Vulnerability Assessment procedure enables Continuous Monitoring by continuously performing vulnerability scans with a configurable frequency. It supports both connected IoT devices with an extending VAM SBOM format, and connected IoT devices with a dedicated interface to transmit the Manufacturer's webserver address. It supports CVE records in json format, but can easily be extended with further Vulnerability Information file formats. Thereby, this VA procedure allows for providing the user with detailed Vulnerability Information concerning the operated IoT devices and lays the foundation for further Vulnerability Scoring as described in Section 3.

4.3.2 Trustworthiness Assessment (TA)

Trustworthiness Assessment (TA) is the next big building block for the Continuous Monitoring performed by this VAM solution and serves as important basis for the Runtime IoT Score described in Section 3.

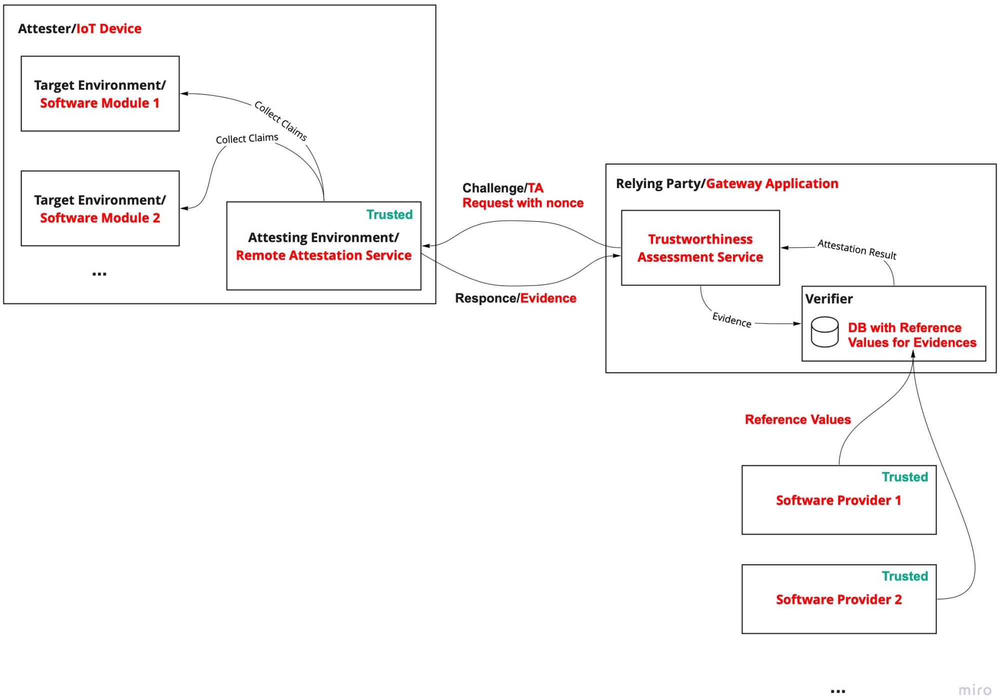


Figure 4.6: Underlying Trustworthiness Assessment model of the VAM solution proposed by this thesis. The roles according to the IETF RATS Internet-Draft [7] are written in black. The terminology used throughout this thesis is written in red. The entities that represent the trust anchor are marked as "Trusted".

A well-established method for assessing the trustworthiness of a device is Remote Attestation as described in Section 2.6. Hence, it is chosen for the VAM solution proposed by this thesis for assessing the trustworthiness of the Software Module deployed on devices connected to the gateway. As reasoned above, it is assumed here that there is only one Software Module deployed on every device.

An extension towards larger numbers of deployed Software Modules is planned for future work.

The Gateway Application of the VAM solution proposed by this thesis is comparable to the ”Trusted Application Manager (TAM)” described in the use case of ”Trusted Execution Environment Provisioning” in the IETF Internet-Draft for ”Remote Attestation Procedures Architectures” (RATS) [7]. There, the TAM is responsible for managing software components deployed in a TEE and for assessing their trustworthiness. Therefore, it performs a Remote Attestation as described in Section 2.6 for software components in the TEE.

Based on the Background-Check Model described in Section 2.6, the System Model presented in Section 4.1 has been extended. It is shown in Figure 4.6 and assigns the roles of Attester, Relying Party, and Verifier.

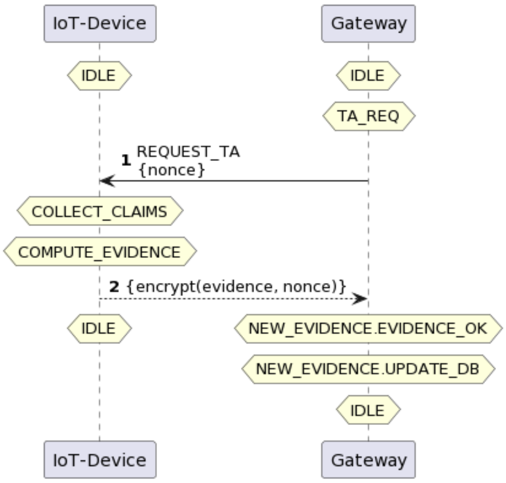


Figure 4.7: Message sequence chart showing the download procedure of the Evidence in the Remote Attestation procedure

The IoT device serves as Attester and contains one or more Target Environments with the isolated Software Modules. They may reside in one or more separated TEEs, or in unprotected memory areas. In addition to these Target Environments, an Attesting Environment provides the actual Remote Attestation service to the Gateway. This service has to be trusted, hence, it is required to reside in a TEE. Ideally, its Trusted Computing Base should be as small as possible. When using one of the security architectures presented in Section 2.5 for the IoT device, the TEE is implemented with a Hardware Root of Trust. In this case, no software has to be trusted to guarantee the TEE's trustworthiness (with exception of Multizone, which is hardware-enforced, but requires a small, verifiable software layer).

When challenged with a TA request, the RA service collects the Claims of all challenged Target Environments, hence, Software Modules. Details of what these Claims may contain are described below.

The Relying Party in the context of this VAM solution is the Gateway Application. It runs the hereby described Trustworthiness Assessment service. Therein, it sends its TA request and a nonce as Challenge to the RA service of connected devices as shown in the message sequence chart in Figure 4.7 with message 1). As described in Section 2.6, the nonce is incorporated into processing the Claims in order to guarantee freshness of the provided Evidence. If the process of calculating the next nonce is a secret, a man-in-the-middle-attacker cannot guess it properly and use it to proclaim an old or maliciously modified Evidence. The RA service on the IoT device processes the nonce and the collected Claims and answers the TA request with the Evidence as response in message 2).

This Evidence is then verified by the Gateway Application. Therefore, an additional component within the Gateway Application takes the role of the Verifier and has sufficient information for appraising the Evidence. For this VAM solution, this is solved by managing a database with reference values for all Evidences of all Software Modules deployed on all devices. Thereby, there is no need for an external Verifier that has to be trusted, to whom interfaces have to be specified and with which messages have to be exchanged. This reduces complexity, network traffic, and the number of necessary trust anchors.

In order to enable this, the Device Registration procedure described in Section 4.2.2 has to be extended as demonstrated in Figure 4.8. In addition to the VAM SBOM file, the reference value for this Software Module in form of a hash of the Claims has to be sent to the Gateway by the Manufacturer as shown with message 1). Upon receiving, the Gateway Application updates its database accordingly.

Since the SPDX format chosen to represent software inventory information for this VAM solution does not provide an appropriate field for such reference evidence, it would have to be extended in order to incorporate this information. Thereby, the proposed VAM SBOM format would not be SPDX-compliant anymore. Therefore, it is encouraged to find a universal representation of such reference evidence and include a corresponding field in a future version of the SPDX format. Until then, it is proposed to send the reference Evidence as message additional to the SPDX-compliant VAM SBOM file as described above.

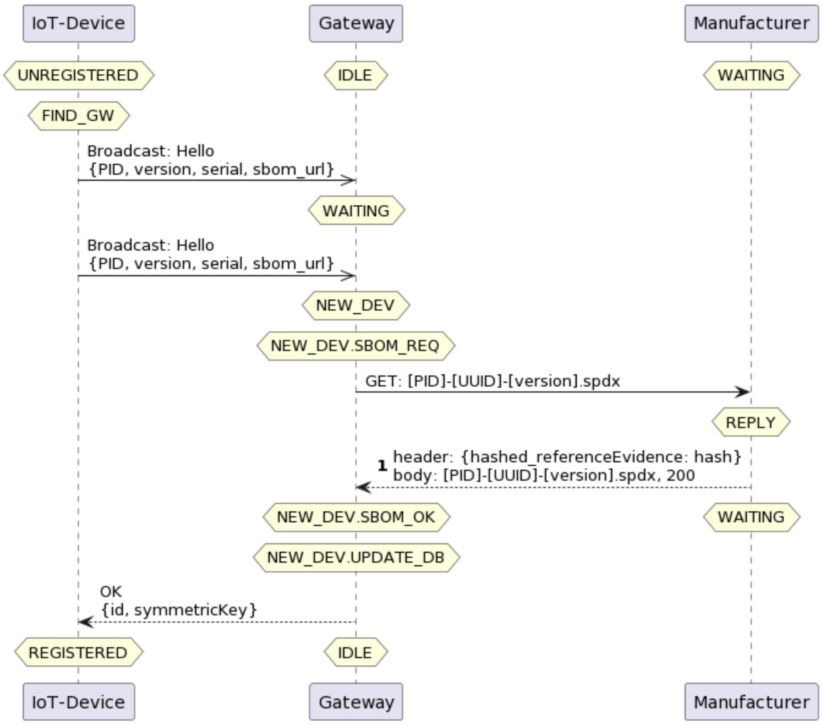


Figure 4.8: Message sequence chart showing the download procedure of the reference Evidence during the device registration procedure

In order to enable symmetric encryption of the Evidence between the IoT device and the Gateway during the remote attestation, a symmetric key is necessary. In general, this key could be generated by:

1. the Software Provider and transmitted with the VAM SBOM as described above, or
2. the Gateway during the Device Registration procedure.

The latter has the clear advantage, that identical IoT devices can receive different symmetric keys in order to guarantee authenticity of the encrypted message. Therefore, this option is chosen for the VAM solution proposed with this thesis. In both cases, the symmetric key has to be securely stored in a TEE.

An appropriate specification for the appearance of the reference Evidence is proposed below.

Since the reference Evidence is crucial for appraising the Evidence received from the IoT device, its creation and receipt has to be trusted as well.

In order to verify the attestation result, the Evidence is compared to its corresponding reference value for every Software Module. If they are equal, the Remote

Attestation is considered successful and the corresponding Software Module trustworthy. If they differ, the Trustworthiness Assessment failed for this Software Module on this very IoT device. There is no further gradation in between success and failure. As indicated in Figure 4.1 and described in Section 3, this TA result is included in the calculation of the Runtime IoT Score proposed above. Based on this score, the user can be provided with further recommendations for action as described below.

When discussing trust and trust anchors, the role of the Gateway has to be taken into consideration as well. As described in Section 4.1, the hardware properties fulfilled by the Gateway are not as precisely specified as the ones of the constrained IoT devices. It is generally assumed that it is less resource-constrained than the IoT devices. In order to keep the solution proposed by this thesis as generic as possible, further restrictions should be avoided at this point. However, if the Gateway is nearly as restricted as the IoT devices, a security architecture comparable to those described in Section 2.5 may be used to enable a TEE that contains all components of the proposed VAM solution as they are all crucial for the Vulnerability Assessment and Management tasks of the Gateway.

If the Gateway has more powerful hardware, there exist security architectures such as Intel SGX [13] or ARM Trustzone for Cortex-A [44]. However, the focus of this thesis is placed on assessing the overall security of constrained IoT devices in a smart home environment and hence, such architectures are omitted here.

In the upcoming section, a specification for the reference Evidence is provided. Based on this specification, the TA model described above is applied to the three already discussed security architectures. Then, this Section is concluded with a discussion of the frequency with which this TA procedure shall be executed.

Reference Evidence

A universal specification for the reference Evidence expected by the Verifier is necessary to provide support of different security architectures. For Sancus [42], one of the security architectures described in Section 2.5, such an identity of a Software Module has been proposed and serves as a basis here.

For better reference, the representation of the Memory layout of the Sancus architecture is included here again in Figure 4.9. The Software Module in memory is separated into its text section and its data section. The *layout* of the SM consists of the start and end addresses of both the text section and the data section. The *identity* of the SM consists of a hash of the text section and the *layout*. Thereby, the resulting *identity* of two identical Software Modules running in parallel on the device will have different *identities*.

In addition to this *identity*, there exists a private key $K_{N,SP,SM}$ for this Software Module of this Software Provider for this very device. This key is stored in a protected TEE and can be used for symmetric encryption of the Evidence.

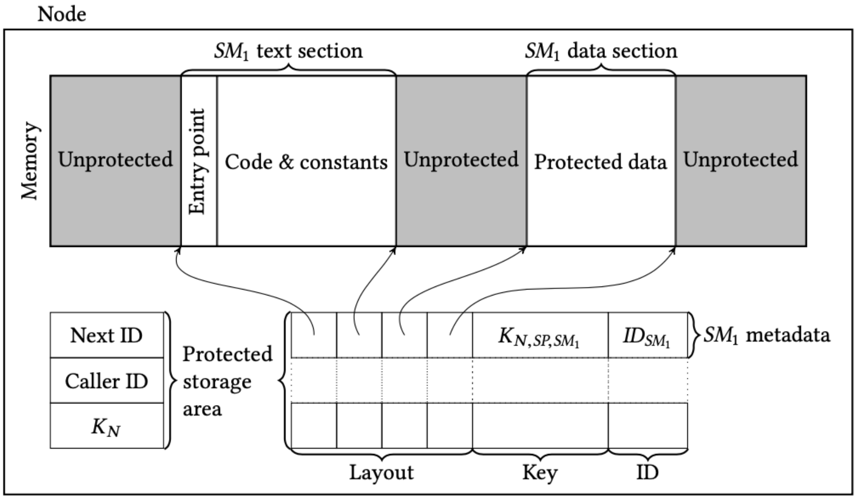


Figure 4.9: Memory layout of the Sancus architecture, taken from [42]

Based on these universal parameters, RA procedures for different security architectures can be implemented. The following sections propose RA procedures for the security architectures described in Section 2.5.

Remote Attestation for ARM's Trustzone

System on Chips with ARMv8-M architecture with Trustzone provide a secure and a non-secure world. The secure world is well-isolated from the non-secure world, but without a Memory Protection Unit (MPU), no further differentiation of privileged and unprivileged software within the secure world is possible. Since the MPU is an optional extension, its presence is not assumed here.

To keep the TCB within the secure world small, it shall only contain a Secure Boot process, the RA service described hereby, and the necessary metadata. Accordingly, it is assumed that the actual Software Module resides in the non-secure world. A Secure Boot is necessary to start up the IoT Device in a state where the secure world can be set up securely before handing control over to the non-secure world (e.g. to the bootloader of a rich OS in non-secure world [44]). To reduce complexity, the focus of this work is set to the RA service and details of the Secure Boot process are omitted here.

For the Software Module deployed on the IoT Device, the RA service has to be informed about the SM's memory layout. It is suggested that this is done during the Linking of the SM image by the Manufacturer, since the IoT Device's memory layout is determined there, and following the assumption that the development process at the Manufacturer is not compromised.

Later changes on this memory layout are only legal, if the TCB is extended with a Secure Update process in the secure world. During every update of the Software Module, the RA service has to be updated as well.

In addition to the SM's layout, the private key for encryption is required. As part of the Device Registration procedure, the Gateway chooses a private key and transmits it to the RA service in the secure world. Mechanisms for securing the private key during transmission are subject of future work.

Thereby, the secure RA service is able to keep track of the start and end addresses of the text and data sections of all Software Modules on the device. Whenever an RA is requested for a Software Module, the RA service computes a hash of its text section and its layout.

The resulting hash and the nonce are then encrypted using the SM's private key and sent back to the Gateway by the RA service.

Remote Attestation for RISC-V Multizone

Implementing Remote Attestation for Multizone requires an extension of the Multizone nanoKernel. This nanoKernel is responsible for assigning and protecting the zones for different Software Modules and provides a secure communication channel (see Figure 2.17). The nanoKernel is the only software component in Multizone that runs in machine mode and has full access to memory.

During compiling (see Figure 2.18), the zone binaries are arranged and assigned to memory addresses. Hence, at this point, the layouts of all Software Modules can be determined and have to be made available to the nanoKernel.

Again, the private key for encryption is required and is supplied by the Gateway during Device Registration as described above for ARM's Trustzone. A secure Bootloader is already included in the nanoKernel.

In addition to the metadata, the extension of the nanoKernel has to include the RA service itself. This secure RA procedure is similar to the one described above for ARM's Trustzone. Whenever an RA is requested for a Software Module, the RA service within the nanoKernel computes a hash of the text section of the corresponding zone and its layout. The resulting hash and the nonce are then encrypted using the corresponding private key and sent to the Gateway.

Except for the nanoKernel, no other zone has the permission to access the other zones' memory for computing the hash of the text section. In fact, this is the purpose behind the memory protection. Therefore, it is inevitable to extend the nanoKernel. Doing so shall not break its qualities of being "lightweight, formally verifiable, bare metal" [26].

Remote Attestation for Sancus

With Sancus [42], Noorman et al. propose a security architecture that inherently provides an RA service. It is based on the *identity* and $K_{N,SP,SM}$ described in Section 4.3.2, but since RA is considered in the processor architecture by design, it differs significantly from typical implementations. The RA procedure provided with the Sancus processor architecture is described as follows.

Both the *identity* and the $K_{N,SP,SM}$ are stored in a protected memory area that is inaccessible directly from software, and can only implicitly be used by dedicated processor instructions. Additionally, $K_{N,SP,SM}$ is derived from the Software Module's *identity* and since it is stored in protected memory and this protection is trusted, it only has to be computed once. This key is then used program counter-dependent: If the program counter resides in the text section of Software Module SM_n, cryptographic processor instructions will automatically use the corresponding K_{N,SP,SM_n} for encryption and decryption. Additionally, these instructions can only be invoked if the memory protection of this Software Module has been enabled. "It follows that only a well-isolated SM installed on behalf of SP on [node] N can compute cryptographic primitives with $K_{N,SP,SM}$, and this is the basis for implementing both remote attestation and secure communication." [42]

To provide Remote Attestation, the Software Module only has to encrypt any message (e.g. the nonce) implicitly with its $K_{N,SP,SM}$ and the Receiving Party "will have high assurance that it has been produced by SM since, as mentioned above, only SM is able to use this key." [42] Since after the memory protection has been enabled, this SM is considered to be well-isolated, and since the encryption key can only be used by a SM with enabled memory protection, only the encryption key of the received Evidence has to be verified.

For this RA procedure, the isolation and memory protection mechanisms of the underlying security architecture have to be completely trusted. If an additional security layer on top of this hardware Root of Trust is requested, an additional layer with extended permission similar to the nanoKernel in Multizone is required.

This would require extensive changes to the Sancus security architecture and is considered out of scope here.

Frequency-based Trustworthiness Assessment

Let the time interval at which the proposed Trustworthiness Assessment procedure is performed be configurable by the choice of frequency $freq_{TA}$. To the best of our knowledge, a structured study of a reasonable choice for this frequency has not yet been conducted. It shall not be selected lightheadedly, because it occupies resources on the constrained IoT Device: it requires computation; potentially including expensive cryptographic functions, and potentially including switches between TEEs (depending on the underlying security architecture and the implementation).

On the other hand, if an attacker has been able to gain access to the memory assigned to a Software Module, this may have severe consequences. Hence, it is desirable to become aware of this security breach as soon as possible.

It might be argued that a Software Module residing in a TEE does not require Remote Attestation at all. This can only be true, if it can be unmistakenly guaranteed that the underlying security architecture providing this TEE is error free and works completely as intended. However, these are developed by humans and errors occur. Therefore, performing Trustworthiness Assessment on a regular basis serves as an additional layer of security.

However, the sweet spot between occupying the constrained IoT Device too much, and not being able to react to successful attacks fast enough, still has to be studied.

Depending on the so configured frequency, the TA procedure described in Section 4.3.2 is performed accordingly for every connected IoT Device. Thereby, this TA procedure enables Continuous Monitoring, and provides the necessary foundation for the computation of the Runtime IoT Score described in Section 3 and its representation as Security Label to the user as described in Section 4.4.

4.4 Security Score Computation

As described in detail in Section 3, the Vulnerability Assessment (VA) and Trustworthiness Assessment (TA) procedures defined above lay the foundation for the computation of the proposed runtime security score. It is assumed that their latest results per Software Module per IoT Device are stored in a database.

For this VAM solution, the total number of points is set to 100.

As mentioned in Section 3, a failed TA for one or more Software Modules deployed on the device under consideration always results in the maximum possible security score immediately, hence, 100 in the case of this VAM solution. Further aspects of the score computation can be skipped in that case.

Since the NIST NVD is used for retrieving the CVE record during VA (see Section 4.3.1), the CVSS base score and the base severity can be directly extracted from the query result at the fields "*baseScore*" and "*baseSeverity*". Therefore, the vulnerabilities present in the device under consideration can be counted and classified into the four severity levels: *LOW*, *MEDIUM*, *HIGH*, and *CRITICAL*. The points are then assigned with respect to the weighting suggested in Section 3.

In order to keep the security score up-to-date, its computation has to be triggered with the following events:

- if the CVE entries in the device database change
- if the latest TA result changes

As discussed in Section 3.4, this score is then mapped to a traffic light color and displayed to the user based on the following mapping:

- *GREEN*: for a score from 0% to 33%
- *YELLOW*: for a score from 33.1% to 66%
- *RED*: for a score from 66.1% to 100%

4.5 Vulnerability Management

As mentioned before, the mere Vulnerability Assessment is extended with management functionality in order to provide a holistic Vulnerability Assessment and Management (VAM) solution. This includes an individual risk assessment on a per-device-basis and warnings, resp. recommendations for action for the user. The latter is based on the security score computation described above and the individually configured relevance level of the device under consideration (see Section 3.3).

In order to enable this consideration of a relevance level, it has to be configured during the device registration procedure. Hence, the procedure presented in Section 4.2.2 has to be extended accordingly. Therefore, at the beginning of the state *new_dev.update_db*, a user input is provided for choosing a relevance level of either *LOW* or *HIGH* for this IoT device. This configuration is then stored in the database as well.

Relevance Level Scoring Result	*LOW*	*HIGH*
GREEN	no action	no action
YELLOW	no action	send informative warning
RED	send informative warning	send recommendation for shutting down this IoT Device

Table 4.2: Decision matrix for the warning/recommended action in dependence of the scoring result and the IoT device's relevance level

The decision matrix for sending warnings, resp. recommendations for action to the user is provided with Table 4.2. If the security scoring result is *GREEN*, hence, the device is less likely to possess severe vulnerabilities and all deployed Software Modules are considered to be trustworthy, no action is required independent of the configured relevance level.

If the security scoring result is *YELLOW*, hence, the device is more likely to possess a medium number of vulnerabilities with a medium level of severity on average, and all deployed Software Modules are considered to be trustworthy, the system's reaction depends on the configured relevance level. If it is set to *LOW*, no action is required. If it is set to *HIGH*, an informative warning is issued.

If the security scoring result is *RED*, hence, the device is very likely to possess a larger number of more severe vulnerabilities, and/or at least one of the deployed Software Modules is not considered to be trustworthy anymore, the system's reaction depends on the configured relevance level again. If it is set to *LOW*, an informative warning is issued. If it is set to *HIGH*, a recommendation for shutting down this IoT Device immediately is issued.

In both cases, the informative warning includes the security report that is also provided with the Gateway Application. The method of transfer depends on the configuration of the VAM solution. It could be a push message on a mobile device or an email with normal priority to the user's email address.

The method of transfer for the recommendation for shutting down this IoT Device immediately depends on the configuration of the VAM solution as well. It could be a push message with high priority on a mobile device or an email with high priority to the user's email address.

Since this proposal is the first attempt for combining a security score with a level of individual relevance, it is kept fairly simple. Effects of a alternative differentiations of relevance levels have to be investigated in future work.

4.6 Conclusion

To conclude, the Runtime Security Index for IoT devices described in Chapter 3 has been extended with a process for retrieving software inventory information, a precise specification for a standardized SBOM file format that can be used in a network with heterogeneous IoT devices, and an architectural concept for a complete Vulnerability Management Solution for the IoT. Those are further main contributions of this thesis and have been presented in detail, starting with the system model definition and the Device and Software Module Management.

Continuous Monitoring functionality then provides the information necessary for the computation of the Runtime Security Index and has been described in detail as well. Its main building blocks are the Vulnerability Assessment procedure and the Trustworthiness Assessment procedure. Thereby, a concept for the effective interaction of the proposed Vulnerability Assessment and Management solution with the constrained IoT device, its Manufacturer, and a CVE database that includes all relevant specifications and descriptions of interfaces has been proposed.

In order to provide the user with the security state of the smart home system at runtime, a precise specification for the calculation of a novel online IoT security score has been presented in detail, that represents a significant enhancement of existing pre-purchase IoT security indices and brings together several branches of research. The actual computation procedure of the Runtime Security Index for IoT devices at runtime has been described and followed by the description of further risk assessment on Vulnerability Management.

The contributions of this chapter led to a concept of a holistic Vulnerability Assessment and Management (VAM) solution for the smart home IoT domain. It includes a numerical runtime security score that provides insight into the security state of connected constrained devices with a Trusted Execution Environment. This runtime security score computation has been embedded in a comprehensive system architecture that specifies all necessary interfaces between the system components. The effective and functional interaction of all relevant system components is demonstrated in a Proof of Concept implementation described in Chapter 5.

Chapter 5

Proof of Concept Implementation

Based on the related work discussed in Chapter 2, a system architecture for a Vulnerability Assessment and Management solution has been presented in Chapter 4. In order to show the general feasibility and functionality of the proposed approach, a Proof of Concept (PoC) implementation has been realized and is presented in the following sections.

As explained above, System-on-Chips (SoC) with Hardware Root of Trust functionality that enables Trusted Execution Environments are a relatively new development. Therefore, there are not many development boards available for research purposes. For the processor architectures discussed in Section 2.5, e.g. the following development boards are published (the following list may not be exhaustive):

- ARMv8-M with Trustzone:
  - STMicroelectronics NUCLEO board L552ZE-Q
  - Microchip SAM L11 Cortex-M23
- RISC-V with Physical Memory Protection (PMP):
  - SiFive HiFive1 Rev B board
  - Digilent Arty A7 FPGA
- Sancus:
  - Xula 2 Stickit! Board V4.0 (unavailable)

Depending on the number of available platforms and on the size of the research community, the amount of open source software for these development boards is rather limited and most often, only provides basic support and limited documentation. Starting points can be the FreeRTOS port [1], or the STM32CubeL5 MCU Firmware Package [56] for the ARMv8-M architecture with Trustzone, the Multizone SDK for RISC-V architectures [25] or the Sancus TEE Project for the Sancus architecture [53]. Due to improvable documentation, dependence on certain IDEs (e.g. STM32CubeIDE for STMicroelectronics boards, Keil Microcontroller Development Kit (MDK) for Microchip boards), constraints on the availability of development boards, and other unexpected challenges, the entry barrier for producing high quality software is quite high.

Hence, a Proof of Concept (PoC) implementation employing mock-ups to cover secondary parts of the system shows all relevant aspects and is sufficient to show the feasibility and functional capability of the architectural concept. The focus of this PoC implementation is to show the correct interaction of the connected components that collect all information necessary to provide a proper assessment of the overall security state and present it to the user. These essential features remain the same with the actual implementation of firmware for actual development platforms. Since this is rather an engineering task, where technical problems of compilers, undocumented platforms, fast changes of hardware, etc. have to be solved, it does not provide a fundamental gain in scientific knowledge. It can be expected that this situation relaxes in the future when a common ground is found for such hardware architectures, so that these platforms overcome the state of "bleeding edge", gain stability, and receive a proper documentation that is tested by large communities developing firmware, operating systems and applications for them.

In the following sections, the relevant aspects of this PoC implementation are described, both for the gateway application and for the constrained device mock-up. In Section 5.1, the general setup is described. Thereupon, Section 5.2 describes the implementation of the Device and Software Module Management procedure as it has been defined in Section 4.2.

The implementation of the Continuous Monitoring procedures that have been defined in Section 4.3 and that can be divided into Vulnerability Assessment and Trustworthiness Assessment is described in Section 5.3.

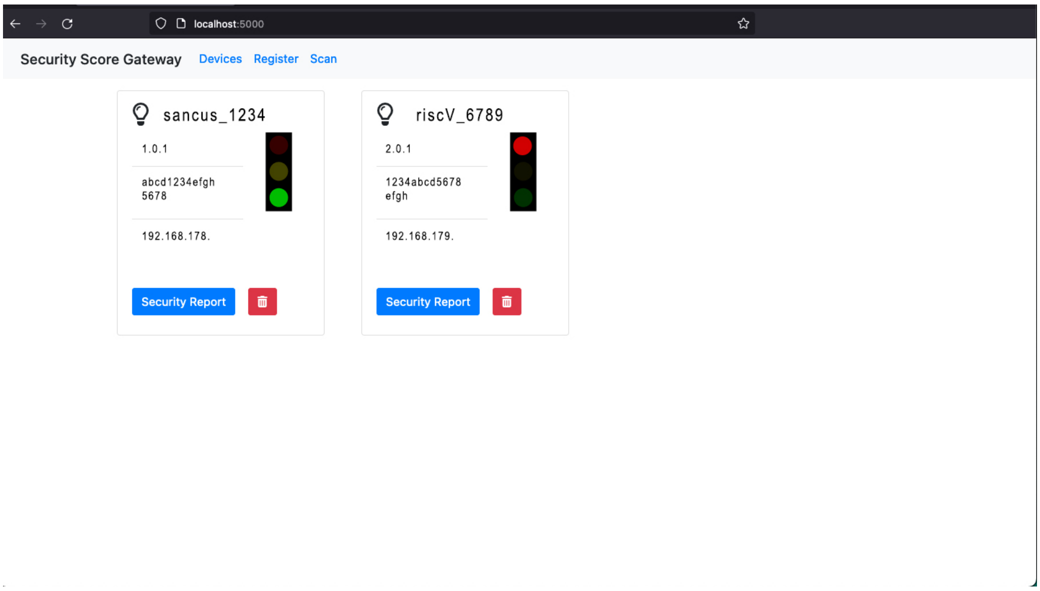
The collected security-relevant information is then merged into the Runtime Security Score that has been defined in Chapter 3 and whose computation procedure has been described in Section 4.4. Its PoC implementation is shown below in Section 5.4.

Last, the implementation of the Vulnerability Management features that have been defined in Section 4.5 is described in Section 5.5.

5.1 General Setup

5.1.1 Gateway Application

As described in the System Model Definition in Section 4.1, the gateway is the dedicated control unit in the underlying system model of this PoC implementation. It provides a simple Graphical User Interface (GUI), performs the device and Software Module management described in Section 4.2, computes the security score proposed in Chapter 3, performs the continuous monitoring discussed in Section 4.3, and the Vulnerability Management described in Section 4.5.



sancus_1234
1.0.1
abcd1234efgh5678
192.168.178.

riscV_6789
2.0.1
1234abcd5678efgh
192.168.179.

Figure 5.1: Example for the overview of all connected devices in the gateway application

The gateway application for this PoC implementation is realized as web-based application using the Python Flask web application framework in version 2.1 [64] running on a general purpose machine. Flask is a very suitable choice for this use case of a PoC implementation as it enables a quick and easy prototype implementation. It is based on the Web Server Gateway Interface (WSGI) [69].

In development mode, the Flask framework runs a webserver at *localhost* at port 5000. Hence, it provides a GUI that is accessible with any browser application and shows an overview of all connected devices (see Figure 5.1). This overview includes a device identifier, its Software Module version, its serial number, and its IP address. A more detailed security report can be accessed and the result of the last security score is represented as a traffic light icon. As described in Section 4.1, all details concerning user-centered design decisions are considered out of scope for this thesis and the traffic light icon proposed in this PoC implementation may be replaced with another user-friendly representation of the underlying security score with neglectable effort.

5.1.2 Constrained Device Mock-Up

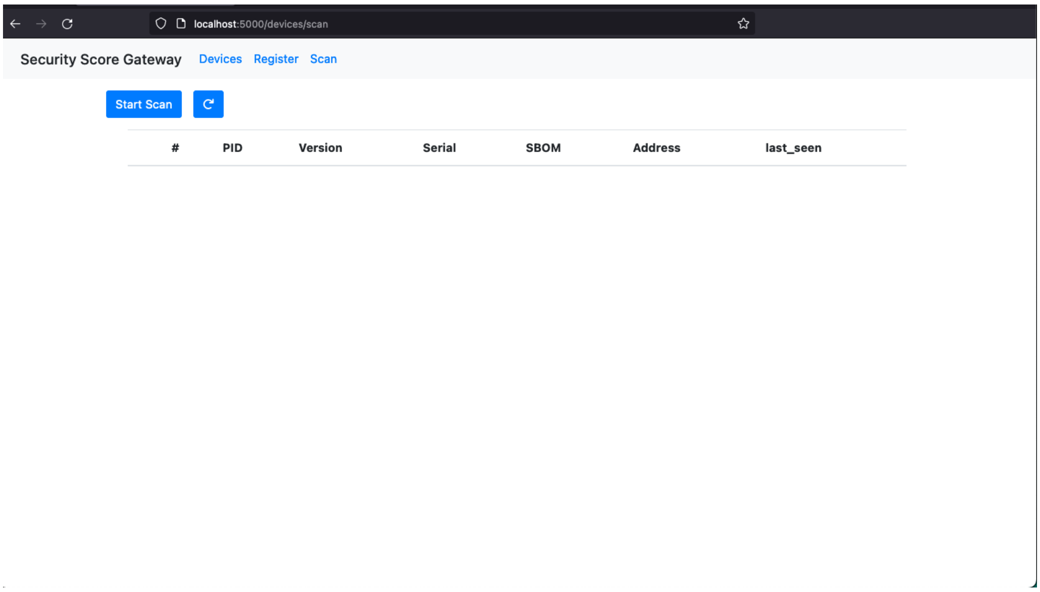
In order to show the general feasibility and functionality of the proposed approach, a mock-up for constrained IoT devices with Hardware Root of Trust functionality that enables Trusted Execution Environments has been implemented. Thereby, the proposed PoC implementation abstracts away the network connection. As explained in the requirements for the proposed device registration procedure in Section 4.2.2, actual hardware development boards that are available for research purposes most often do not provide a large variety of communication interfaces anyway. Therefore, the network connection is neglectable at this time. Hence, the device mock-up runs on the same general purpose machine as the gateway application.

The proposed device mock-up is realized as a plain Python script, independent of Flask or any other web application framework. The device identification information, that would normally be assigned by the Manufacturer and stored in a secure memory region on the IoT device, and the well-defined *gateway_port* and *device_port* are set using a configuration (config) file for now.

5.2 Device and Software Module Management

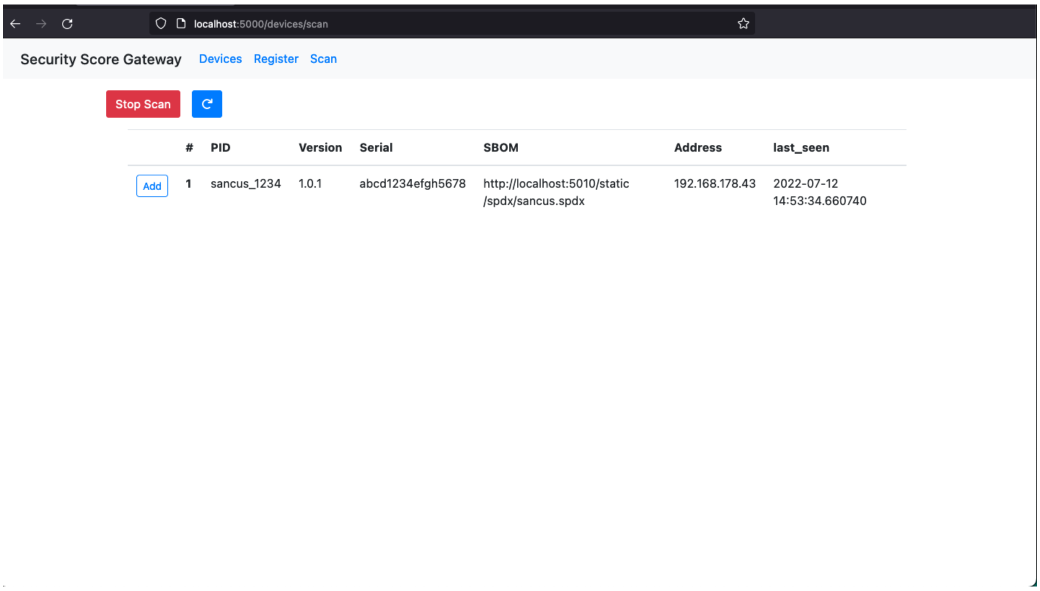
5.2.1 Gateway Application

Concerning the device and Software Module management, a scan for new devices can be initiated as user interaction by clicking on the corresponding link and on "Start Scan" (see Figure 5.2), which puts the gateway from *idle* mode to *waiting*. With starting the scan, a background task is started that handles the device registration procedure described in Section 4.2.2 and proceeds as follows: In the task, a UDP-socket is opened and bound to a well-defined gateway port, e.g. 12345. All incoming messages are then subjected to a pattern matching for the regular expression `r"^HELLO;.*;.*;.*;.*$"`. If such a message is received, the gateway changes its state to *new_dev* and the message is parsed and the product ID PID, the software binary version *version*, the serial number *serial* and the url to the Manufacturer's webserver that provides the software inventory information $CreatorWebsite$ are temporarily stored in a database with all devices found during the scanning process. All scanned devices are then displayed to the user and available for adding (see Figure 5.3).



#	PID	Version	Serial	SBOM	Address	last_seen

Figure 5.2: Initial appearance of the route */devices/scan* before starting the device registration process



	#	PID	Version	Serial	SBOM	Address	last_seen
Add	1	sancus_1234	1.0.1	abcd1234efgh5678	http://localhost:5010/static /spdx/sancus.spdx	192.168.178.43	2022-07-12 14:53:34.660740

Figure 5.3: List of all devices found in the scanning process during the device registration process

After the user has chosen a device for adding it, the gateway changes to *new_dev.sbom_req* state and a background task is started that sends a http-*GET*-request to the *CreatorWebsite*-URL. Upon receiving the VAM SBOM file from the Manufacturer, the gateway changes to *new_dev.sbom_ok* state and performs basic sanity-checks. At this point, the PoC implementation is prepared for further security measures to be implemented in the future, e.g. encryption and/or certificates. When the file analysis succeeds, the gateway changes its state to *new_dev.update_db*, parses the file, and stores it into a persistent device database.

Finally, the gateway sends a `OK;{id}`-message with a device ID assigned by the gateway back to the device using another well-defined device port, e.g. 5020. Thereby, the device registration procedure at the gateway is complete.

5.2.2 Constrained Device Mock-Up

Since the device mock-up keeps sending its broadcast message until it receives the OK from the gateway, two separate threads are utilized for the different tasks. Initially, the device is in *unregistered* state. Due to a console-based user input, the device changes to *find_gw* state and starts broadcasting. For the actual IoT device, the search for a reachable gateway could e.g. be initiated due to the user pressing a dedicated pairing button on the IoT device.
The `HELLO;{pid};{version};{serial};{sbom_url}`-message is sent over a UDP-broadcast-socket to the well-defined gateway port of the gateway application on the same localhost every 5 seconds.

In parallel, the second thread listens on another UDP-socket on the well-defined device port. All received messages are then subjected to a pattern matching for the regular expression `r"^OK;\d+$"`. If such a message is received, the device changes to *registered* state, the thread is terminated, and it joins the main application thread. This event is used to stop the broadcasting thread as well. As explained in Section 4.2.2, the actual IoT device could now start performing tasks that are related to its purpose as smart home IoT device.

5.3 Continuous Monitoring

The continuous monitoring functionality includes both Vulnerability Assessment and Trustworthiness Assessment as main building blocks, whose PoC implementations are described in detail in the following paragraphs.

5.3.1 Vulnerability Assessment

Gateway Application

As presented in Section 4.3.1, the Vulnerability Assessment process is divided into the search for new CVE IDs that are associated with any of the Software Modules deployed on connected constrained devices and the retrieval of the corresponding CVE records from a CVE database. Thereby, the search for new CVE IDs depends on the configurable frequency $freq_{newCVE}$ that is valid for all connected constrained devices.

The default value for the configurable frequency $freq_{newCVE}$ is daily.

For the first case described in Section 4.3.1, the VAM SBOM format is extended with a $VulnerabilityInformationDownloadLocation$ field in the Package Information-area. Hence, the file parsing during the device registration procedure described in Section 5.2.1 is extended accordingly and the URL is stored in the persistent local device database of the gateway.

For the second case, the IoT device has to provide a dedicated interface for retrieving the $VulnerabilityInformationDownloadLocation$ during the device registration procedure. Therefore, the device registration procedure described in Section 5.2.1 remains unchanged until the gateway would normally send the `OK;{id}`-message back to the device. Instead, the gateway changes to $vulninf\_locator\_req$ state and sends a $REQUEST\_VULNINF\_LOCATOR$-message with the payload `REQ_VULNINF_LOCATOR` to the IoT device using the UDP-socket described above. Upon receiving the device's response, the gateway responses with the OK-message of the original device registration process as described above, changes to $vulninf\_locator\_rec$ state and stores the URL in its persistent local device database.

The PoC implementation of the gateway application proposed with this thesis supports both cases.

A background task is in charge of periodically requesting the Vulnerability Information for all connected devices in a loop. Therefore, the gateway changes to $vulninf\_req$ state every $\frac{1}{freq_{newCVE}}s$, iterates through all local device database entries, and sends http-GET-requests to the corresponding $VulnerabilityInformationDownloadLocation$-URLs . Upon receiving the Vulnerability Information, the gateway changes to $vulninf\_rec$ state and performs basic sanity-checks. At this point, the PoC implementation is prepared for further security measures to be implemented in the future, e.g. encryption and/or certificates. When the file analysis succeeds, the gateway changes its state to $new\_dev.update\_db$, parses the received string with a pattern matching for the regular expression `r"^(CVE-\d{4}-\d{4,};)+$"`, stores it into the persistent device database, and continues with the next device. Thereby, the search for new CVE IDs at the gateway is complete.

For retrieving the CVE records for all CVE IDs stored in the gateway's device database, another background task is used that periodically sends requests to the

API provided for the National Vulnerability Database by the U.S. National Institute of Standards and Technology [38]. Therefore, the gateway changes to *cve_req* state and sends http-*GET*-requests to the webserver provided by NIST at the address `https://services.nvd.nist.gov/rest/json/cve/1.0/[CVE_ID]` for every CVE ID stored in the database. Upon receiving the corresponding CVE record in json format, the Gateway changes to *cve_rec* state, parses the file, and updates its local databases accordingly. Afterwards, it changes to *idle* state.

In addition to the periodical procedure described above, the search and retrieval process is also triggered exclusively for the affected Software Modules whenever the internal database of all deployed Software Modules is changed (e.g. because a new IoT device is registered at the gateway). Combining this exception with the periodical procedure, both the search for new CVE IDs and the retrieval of the corresponding CVE records from a CVE database at the gateway are complete. This results in a device database with vulnerability information that is as actual as the frequency $freq_{newCVE}$ allows it.

Constrained Device Mock-Up

The first case for the search for new CVE IDs does not require any changes on the device implementation. Both the Gateway Application and the Manufacturer webserver commit to the extension of the VAM SBOM format and exchange the relevant URL without involving the constrained device whatsoever.

The second case requires the constrained device to provide an interface for requesting the $VulnerabilityInformationDownloadLocation$ during the device registration procedure. As described in Section 5.2.2, a dedicated thread is used to listen on another UDP-socket on the well-defined device port. All received messages are then subjected to a pattern matching for the regular expression `r"^REQ_VULNINF_LOCATOR$"`. If such a message is received, the device changes to *reply_vulninf_locator* state, the thread is terminated, and it joins the main application thread. This event is used to stop the broadcasting thread as well.

In the main application thread, the `http://[CreatorWebsite]/[pathToVulnerabilityInformation]/[PID]-[UUID]-[version]`-message is sent over a newly opened UDP-socket to the well-defined gateway port of the gateway application. The handling of the OK-message as last message of the device registration process remains unchanged.

The PoC implementation of the constrained device mock-up proposed with this thesis supports both cases.

The retrieval process of the corresponding CVE records from a CVE database does not involve the constrained device and therefore does not require any changes whatsoever.

5.3.2 Trustworthiness Assessment

Gateway Application

As presented in Section 4.3.2, Remote Attestation is chosen for assessing the trustworthiness of the Software Modules deployed on connected constrained devices. The Trustworthiness Assessment procedure depends on the configurable frequency $freq_{TA}$ that is valid for all connected constrained devices.

The default value for the configurable frequency $freq_{TA}$ is daily.

In order to provide the Gateway with the reference Evidence that is necessary to verify the Evidence received from the constrained device as described in Section 4.3.2, it is necessary to extend the device registration procedure accordingly. As described in Section 5.2.1, a background task is responsible for the communication with the Manufacturer webservice. With this extension, the Gateway application does not only expect to receive the VAM SBOM file, but also the reference Evidence in form of a hash of the Claims. Upon receiving both, the Gateway changes to *new_dev.sbom_ok* state. When the VAM SBOM file handling succeeds, the Gateway changes its state to *new_dev.update_db*, and stores both the VAM SBOM file and the reference Evidence into a persistent device database.

Then, the Gateway sends an extended `OK;{id};{symmetricKey}`-message with the device ID and the symmetric key assigned by the gateway back to the device using the well-defined device port. Thereby, the device registration procedure with the extension towards Trustworthiness Assessment at the gateway is complete.

A background task is in charge of periodically requesting the current Evidence for all connected devices in a loop. Therefore, the Gateway changes to ta_req state every $\frac{1}{freq_{TA}}s$, iterates through all local device database entries, establishes a TCP-connection with the corresponding device, and sends a $REQUEST\_TA$-message with the payload `REQ_TA;{nonce}` to the IoT device. Upon receiving the encrypted Evidence and nonce, the gateway changes to *new_evidence.evidence_ok* state, decrypts the message using the symmetric key of this device, and compares the received Evidence to the reference Evidence and the received nonce with the one sent. When this verification succeeds, the Gateway changes its state to *new_evidence.update_db*, and updates the database entry for the last TA for the corresponding device with a timestamp. Then, it continues with the next device. Thereby, the periodical Trustworthiness Assessment procedure at the gateway is complete. This results in a device database with trustworthiness information that is as actual as the frequency $freq_{TA}$ allows it.

Constrained Device Mock-Up

The extension of the device registration procedure concerning the retrieval of the reference Evidence does not require any changes on the device implementation. Both the Gateway Application and the Manufacturer webserver commit to the extension of the VAM SBOM format and exchange the relevant reference Evidence in form of a hash of the Claims without involving the constrained device whatsoever.

Concerning the OK-message that the device receives in the final step of the device registration procedure, the pattern matching for the regular expression is changed to `r"^OK;\d+;.*$"` and the symmetric key for the encryption of future Evidences is stored in persistent memory.

For the periodical Trustworthiness Assessment procedure, the constrained device receives the $REQUEST\_TA$-message with the nonce via the TCP-connection and changes its state to *collect_claims*. For the purpose of this PoC implementation, both a trustworthy and a not trustworthy constrained device mock-up are implemented.

For the trustworthy constrained device mock-up, the reference Evidence is manually deployed on the constrained device. For the not trustworthy constrained device mock-up, a deviating Evidence is deployed.

For composing the response message to the Gateway, the device changes to *compute_evidence* state, the Evidence stored on the device is encrypted together with the received nonce and sent back to the Gateway via the TCP-connection. Afterwards, the device changes to *idle* state.

5.4 Security Score Computation

5.4.1 Gateway Application

As presented in Section 4.4, the Vulnerability Assessment (VA) and Trustworthiness Assessment (TA) procedures implemented above lay the foundation for the computation of the proposed runtime security score. In order to keep the security score up-to-date, its computation has to be triggered with the following events:

- if the CVE entries in the device database change
- if the latest TA result changes

During the security score computation in another background task, the Gateway application iterates through all local device database entries, looks up the latest results of the Vulnerability Assessment and the Trustworthiness Assessment, and computes the score as follows. If the latest result of the Trustworthiness Assessment is "*false*", the security score is immediately set to 100 and the computation procedure breaks.

Otherwise, the stored CVE records are iterated and the occurrences of the CVSS "*baseSeverity*" levels *LOW*, *MEDIUM*, *HIGH*, and *CRITICAL* are counted. Based on the number of vulnerabilities found, the security score is incremented with the indicated points as described in Section 3.1:

- *Critical*
  - 11 or more critical vulnerabilities (42 points)
  - 6-10 (38 points)
  - 1-5 (33 points)
- *High*
  - 11 or more highly severe vulnerabilities (29 points)
  - 6-10 (25 points)
  - 1-5 (21 points)
- *Medium*
  - 11 or more medium severe vulnerabilities (17 points)
  - 6-10 (13 points)
  - 1-5 (8 points)

- *Low*
    - 11 or more low severe vulnerabilities (12 points)
    - 6-10 (8 points)
    - 1-5 (4 points)

As described in Section 3.4, the resulting score is then stored in the device database, mapped to a traffic light color, and displayed to the user based on the following mapping:

- *GREEN*: for a score from 0 to 33
- *YELLOW*: for a score from 33.1 to 66
- *RED*: for a score from 66.1 to 100

Thereby, the computation of the security score at the gateway is complete. This results in a device database with up-to-date security information.

5.4.2 Constrained Device Mock-Up

The computation of the security score does not require any changes on the device implementation, since it exclusively takes place at the Gateway.

5.5 Vulnerability Management

5.5.1 Gateway Application

As described in Section 4.5, the mere Vulnerability Assessment described above is extended with the Vulnerability Management aspects of individual risk assessment on a per-device basis and warnings, resp. recommendations for action for the user. The former is based on a relevance level that is individually configured during the device registration procedure of that very device. Therefore, the device registration procedure implemented above has to be extended as follows.

At the beginning of the state *new_dev.update_db* at the Gateway, a selection form is displayed to the user. Depending on the user's choice, the relevance level of that device is either set to *LOW* or *HIGH* and stored in the local device database.

Then, the background task that performs the security score computation described in Section 5.4.1 is extended as follows. After the computation of the security score is complete and has been mapped to the traffic light color, the decision matrix presented in Table 4.2 in Section 4.5 is implemented. The ”informative warning” is displayed as highlighted text box in the device overview view shown in Figure 5.1 with the warning text that ”The security state of this device is $\{YELLOW|RED\}$! More details on possible vulnerabilities can be found in the security report.”

The ”recommendation for shutting down this device immediately” is displayed to the user as pop-up window and includes the following warning text: ”The security state of the highly relevant device with device ID $\{id\}$ is RED. It is recommended to shut this device down immediately, until the possibility for severe security breaches has been investigated. More details on possible vulnerabilities can be found in the security report. Contact the device Manufacturer for information on software updates that solve the security issues.”

Thereby, the Vulnerability Management of the VAM solution at the gateway is complete.

5.5.2 Constrained Device Mock-Up

The computation of the security score does not require any changes on the device implementation, since it exclusively takes place at the Gateway.

5.6 Conclusion

Based on the related work discussed in Chapter 2 and the system architecture for a Vulnerability Assessment and Management solution presented in Chapter 4, a Proof of Concept (PoC) implementation, that shows the general feasibility and functionality of the proposed approach, has been realized and presented. It consists of the actual implementation of a Gateway application and a mock-up of a constrained IoT device that provides the following features:

- the initial device registration with transmission of an SBOM in accordance with the proposed SBOM file specification

- an online Vulnerability Assessment for all registered Software Modules with periodic checks for new disclosed vulnerabilities and updated vulnerability information

- an online Trustworthiness Assessment of all registered Software Modules in order to assess the modules’ integrity

- an online computation of the aforementioned security score for constrained IoT devices

- an online Vulnerability Management for all registered Software Modules with individual risk assessment on a per-device level and warnings and recommended user actions derived therefrom
- a basic webbrowser-based Graphical User Interface (GUI) that presents an overview of all connected devices, their security scores, a more detailed security report, and allows access to individual per-device risk assessment

With the contributions of this chapter, the effective and functional interaction of all relevant system components has been shown.

Chapter 6

Evaluation

The contributions of this thesis led to a concept of a holistic Vulnerability Assessment and Management (VAM) solution for the smart home IoT domain. It includes a numerical runtime security score that provides insight into the security state of connected constrained devices with a Trusted Execution Environment. This runtime security score computation was embedded in a comprehensive system architecture that specifies all necessary interfaces between the system components. The effective and functional interaction of all relevant system components has been shown in a Proof of Concept implementation.

With these contributions, this thesis represents the first step into a new research direction by combining findings from several research areas to a novel approach. This required the thorough analysis of the related research fields in Chapter 2 that laid the foundation for following design decisions.

Before the numerical runtime security score was proposed in Chapter 3, there existed scoring systems for the IoT domain based on the Common Vulnerability Scoring System and on experts' manual assessments of device characteristics, e.g. its update process, open ports, its susceptibility for replay attacks, etc. However, the identification of these device characteristics can hardly be automated and does not prove suitable for a computation at runtime. Most often, this manual assessment requires to disrupt the device's normal operation. Additionally, to the best of our knowledge, there exists no approach for an IoT scoring system that takes Trustworthiness Assessment into account at all. Therefore, the runtime security score proposed with this thesis provides a concise overview of the overall system health at a glance during operation. It incorporates both database-based Vulnerability Assessment and Trustworthiness Assessment based on remote attestation and targets constrained devices in the IoT domain. This represents a novel approach and an important scientific advance in this field of research.

One of the main building blocks for Vulnerability Assessment and Trustworthiness Assessment is a concept for a standardized description of the software inventory in order to reliably discriminate a certain Software Module with designated version number and components. Based on the proposed VAM SBOM format, a standardized device registration procedure with SBOM retrieval is proposed. It only requires an existing communication interface between the gateway and the IoT device, the existence of a Manufacturer as entity responsible for providing the VAM SBOM file and the existence of a product ID and a serial number, an URL to the Manufacturer webservice assigned by the Manufacturer and stored on the device. For the IoT device to implement the specified interface to the gateway, it only has to be capable of sending a broadcast message with the device identification information and the SBOM URL to the gateway at a well-defined gateway port and of receiving a response at a well-defined device port.

Depending on the rate of the broadcasting messages, the communication protocol for the basic device registration procedure requires at least 2 messages between the Gateway and the constrained IoT device. Hence, this thesis proposes a feasible approach with preliminaries that can be met by any constrained IoT device.

Besides being suitable for constrained IoT devices, another advantageous feature of the proposed VAM solution is that it is fully automated. Opposed to approaches where the SBOM URL is e.g. in the device documentation or within a QR code on the packaging, the proposed SBOM retrieval mechanism does not require any such user interaction.

The scalability of the proposed device registration procedure solely depends on the scalability of the Gateway, following the assumption that the Manufacturer webservice can handle an amount of requests that exceeds the number of smart home IoT devices in a local network by far. It is not probable that the user performs the device registration procedure in parallel for a large number of constrained IoT devices. But even if so, a larger number of constrained IoT devices in the system does not affect the individual device at all, except for the additional time the Gateway may need to process all device registrations. This may result in a larger time span in which the device keeps sending its broadcasting messages. However, this should not be significant over the total battery life.

The main building block for Vulnerability Assessment and Trustworthiness Assessment that links software to a vulnerability database is a model for partitioning all software on the device into Software Modules. Existing specifications for such software inventory information have been analysed and a precise specification based on the SPDX format for a standardized SBOM format that can be used in a network with heterogeneous IoT devices was proposed. This Vulnerability Assessment and Management Software Bill of Material (VAM SBOM) is suitable for constrained IoT devices that are not capable of sending their SBOM file themselves. Excluding licensing information, it requires only 14 mandatory fields to precisely and unambiguously describe and identify Software Modules. In the initial version presented here, it only requires a flat text file, which enables a simple parsing process. It is compliant with the SPDX specification in version 2.2 and with the SBOM retrieval mechanism proposed by the IETF.

The Vulnerability Assessment and Management (VAM) solution proposed in this thesis does not implement an interface for retrieving the SBOM file directly from the device. Hence, the SBOM file size is not crucial and a more concise representation not absolutely necessary. Instead, it complies with the scope of the IETF Internet-Draft for SBOM retrieval in so far as the VAM solution serves as network-layer management system retrieving an SBOM from an IoT device that may not be capable of transmitting the SBOM file directly.

Without the software inventory information being included in a Gateway application for a smart home IoT network, the information what software in which version is running on which constrained IoT device is not easily available for an inexperienced user. Such information may be available in the device manual or online, but most often requires an amount of research that is unreasonable for an inexperienced user. Therefore, including it in the security report as proposed with this thesis makes it far more easily accessible and easier to survey.

Periodical Vulnerability Assessment based on a CVE database is a major part of the Continuous Monitoring functionality proposed with the VAM solution. Without such database-based Vulnerability Assessment, neither undisclosed nor disclosed vulnerabilities can be identified systematically. With the Vulnerability Assessment approach proposed in this thesis, disclosed vulnerabilities that possess a corresponding CVE record can be assessed easily.

The number of messages between the Gateway and the constrained IoT device that the communication protocol for the Vulnerability Assessment procedure requires, depends on the implementation of the interface for retrieving the download location for the Vulnerability Information as specified in Section 4.3.1. If the $VulnerabilityInformationDownloadLocation$ can be found as additional field in the VAM SBOM, the whole Vulnerability Assessment does not require any additional messages between the Gateway and the constrained IoT device: the VAM SBOM is retrieved during the device registration, the $VulnerabilityInformation$ $DownloadLocation$ is stored in the local Gateway database and the further search for new CVE IDs and the retrieval of the corresponding CVE records only involves the Manufacturer webserver and the NIST webserver.

If on the other hand, the IoT device has to provide a dedicated interface to transmit the $VulnerabilityInformationDownloadLocation$ to the Gateway, the device registration process has to be adjusted accordingly, introducing 3 additional messages between the Gateway and the constrained IoT device. As argued above, the system's scalability depends on the scalability of the Gateway.

Periodical Trustworthiness Assessment based on remote attestation is another major part of the Continuous Monitoring functionality proposed with the VAM solution. Without such Trustworthiness Assessment, unintended, hence, malicious modifications of software running in a Trusted Execution Environment cannot be identified systematically. Moreover, including it into a holistic approach as proposed by this thesis enables the user to assess such information easily.

The communication protocol for the proposed Trustworthiness Assessment procedure requires only 2 messages between the Gateway and the constrained IoT device. Hence, this thesis proposes a feasible approach with preliminaries that can be met by any constrained IoT device. As argued above, the system's scalability solely depends on the scalability of the Gateway: The individual TA requests do not affect those of other devices.

The specification and periodical computation of the Runtime Security Score as proposed with this thesis represents a novel combination of Vulnerability Assessment and Trustworthiness Assessment. Thereby, an inexperienced user has easy access to important security information in a concise fashion: the user is not only able to identify which software on what device possesses which disclosed vulnerability, but also to ensure that all software running in Trusted Execution Environments on the constrained IoT device has not been compromised.

As stated by the Google Open Source Security Team in [36], "generating an SBOM is only one half of the story. Once an SBOM is available for a given piece of software, it needs to be mapped onto a list of known vulnerabilities to know which components could pose a threat. By connecting these two sources of information, consumers will know not just what's in their software, but also its risks and whether they need to remediate any issues." This linking of software to a vulnerability database is extended with Trustworthiness Assessment, so that consumers will know at runtime whether their software is still trustworthy and uncompromised. This is a very valuable scientific advance.

Additionally, the numerical representation of the security score allows for an adjustment to any total number of points. Thereby, it can be adapted to different use cases and requirements and is prepared for future surveys on the impact of different numerical representations to the user. As proposed with the general cutoffs, a representation with letters, that is comparable to the EU energy labels for electrical devices, can also be implemented easily, as well as a mapping to traffic lights, that is comparable to food Nutrition facts labels. This provides versatile possibilities for customization to meet user needs.

The security score computation solely takes place at the Gateway and does not require any message exchange with the constrained IoT device. Therefore, the system's scalability only depends on the scalability of the Gateway.

The Vulnerability Management procedure proposed with this thesis includes individual risk assessment that is based on a user-driven per-device relevance level. Without such individual risk assessment, it would not be possible to differentiate more important constrained IoT devices and adjust their management accordingly. By introducing different relevance levels, the user can receive more fine-grained warnings and recommendations.

The individual risk assessment solely takes place at the Gateway and does not require any message exchange with the constrained IoT device. Therefore, the system's scalability only depends on the scalability of the Gateway.

In the IoT domain, heterogeneous devices from different manufacturers are often operated in the same network and require interoperability. This can only be achieved when all components agree upon standardized interfaces. The modular and extensible Vulnerability Assessment and Management solution proposed in this thesis requires standardization for a shared SBOM format and for the communication interfaces between the IoT device and the gateway, and the gateway and the manufacturer webserver. Complying to these standards can set incentives for device manufacturers when the thereby achieved insight into the security state of the connected devices during runtime positively affects the consumers' buying decision. An analysis of the effects on the consumer is planned for future work.

Among the contributions of this thesis is a Proof of Concept implementation of the proposed system architecture that shows the effective and functional interaction of all relevant system components. The modularity and extendability of the proposed concept is also reflected in the PoC implementation, where all interfaces of relevant components are well-defined. Hence, this PoC implementation serves very well as reference implementation for future research developments.

This PoC implementation employs mock-ups to cover secondary parts of the system and is strictly oriented to the interfaces and communication protocols specified in Section 4. Thereby, it proves the underlying concept and system architecture. Hence, the availability of actual implementations for development platforms does not provide a fundamental gain in scientific knowledge. Nevertheless, the modularity and well-defined interfaces allow for exchanging the mock-up implementation with actual firmware implementation as soon as it becomes available.

Providing a Graphical User Interface in form of a web application for the Gateway application as proposed with this thesis is a convenient choice. It enables a browser-based representation that is platform-independent and provides extensive design possibilities.

To conclude, the contributions proposed with this thesis represent an important scientific advance in the research fields of Vulnerability Assessment, Vulnerability Management, and IoT Security Scoring Systems. This includes a novel approach for a Runtime Security Score for the IoT domain that combines Vulnerability Assessment with Trustworthiness Assessment that is embedded in an architectural concept of a comprehensive Vulnerability Assessment and Management solution with well-defined interfaces. The correct and effective interoperability of all relevant components has been shown in a modular and extensible PoC implementation.

Chapter 7

Conclusion and Outlook

With the contributions of this thesis, I propose the concept of a software system architecture for networked constrained IoT devices, that can provide the user with a profound insight into the security state of the connected devices during runtime. I achieve this by combining findings from several areas of ongoing research.

Chapter 2 presented a thorough investigation and detailed presentation of these research areas and laid the necessary foundation for all further conceptual decisions. The initial ideas for this thesis have been motivated by the pre-purchase consumer security labels discussed in Section 2.1 and refined by the Vulnerability Assessment and Vulnerability Management approaches presented in Section 2.3 and the IoT security scores discussed in Section 2.4. However, to conclude the findings: pre-purchase consumer security labels do not provide security information during operation of devices. Solutions for Vulnerability Assessment and Vulnerability Management do not cover constrained IoT devices. And IoT security scores most often lack automation capabilities and do not take Trustworthiness Assessment into account. Therefore, the novel combination of these research areas has been identified as very valuable contribution to the state of current research.

Database-based Vulnerability Assessment requires a common understanding of the partitioning of software into Software Modules and an unambiguous identification of them. Hence, the current state of research in the field of Software Inventory Information was presented in Section 2.2.

Trustworthiness Assessment requires a concept of trust that can be provided by Trusted Execution Environments based on Hardware Root of Trust architectures, which have therefore been presented in Section 2.5. The actual Trustworthiness Assessment can then be realized by Remote Attestation, that has been discussed in Section 2.6.

Chapter 3 provided the specification of a novel Runtime Security Score that combines the findings from available IoT security scores with the possibilities offered by Hardware Root of Trust security architectures and Trustworthiness Assessment through Remote Attestation. Therefore, it serves as the basis for the concept of a holistic Vulnerability Assessment and Management solution that is another major contribution of this thesis.

This Runtime Security Score is a novel approach that enables the ongoing assessment of the security state of a smart home network. However, this thesis is the first to set a foot into this new research direction. Therefore, it opens up a new field of research opportunities that can be covered in the upcoming years. In Figure 7.1, a concise presentation of the identified research opportunities is given.

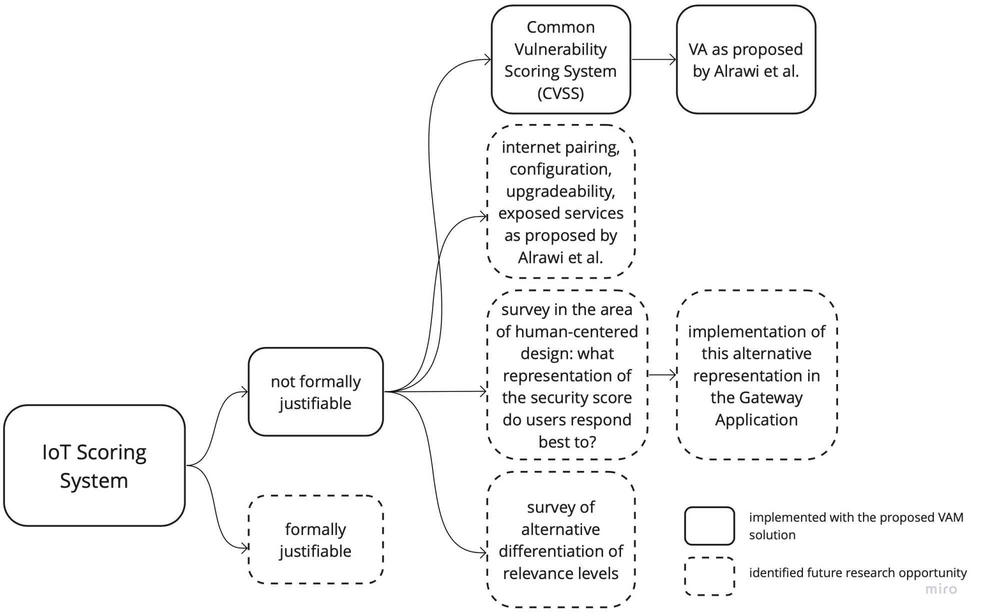

IoT Scoring System
not formally justifiable
formally justifiable
Common Vulnerability Scoring System (CVSS)
VA as proposed by Alrawi et al.
internet pairing, configuration, upgradeability, exposed services as proposed by Alrawi et al.
survey in the area of human-centered design: what representation of the security score do users respond best to?
implementation of this alternative representation in the Gateway Application
survey of alternative differentiation of relevance levels
implemented with the proposed VAM solution
identified future research opportunity
miro

Figure 7.1: Graphical representation of the identified research opportunities concerning the underlying IoT Scoring System

As outlined in Section 2.4, there exist critics of the Common Vulnerability Scoring System, that is the most widely used scoring system for vulnerabilities with enormous practical relevance and therefore served as the underlying vulnerability scoring system for the proposed Runtime Security Score. Despite its practical relevance, it is not formally justifiable. However, there exist only preliminary approaches for formally justifiable Scoring Systems targeting the IoT domain. As soon as a candidate has prevailed there, the proposed Runtime Security Score can be extended accordingly.

The existing, though formally not justifiable Runtime Security Score incorporates the CVSS as proposed by Alrawi et al. Further research can be done towards the incorporation of device-specific characteristics apart from disclosed vulnerabilities, that also contribute to its vulnerability: how does it implement the internet pairing process? Does it force the user to perform an individual device configuration? Does it provide an automatic upgrade process? It shall be investigated how such characteristics can be included in an automated score computation that requires as little standardization as possible.

Subsequent surveys shall be conducted concerning the identification of the most preferable representation of such a security score to the consumer and concerning alternative differentiations of relevance levels during the individual risk assessment.

In Chapter 4, the ideas of this Runtime Security Score have been transferred to an actual software system architecture for a holistic Vulnerability Management Solution for constrained IoT devices. The underlying system model definition that incorporates the modular components and entities of the concept and whose well-defined interfaces provide interchangeability and extensibility of the components was presented in Section 4.1. Figure 7.2 presents the research opportunities identified for this system model.

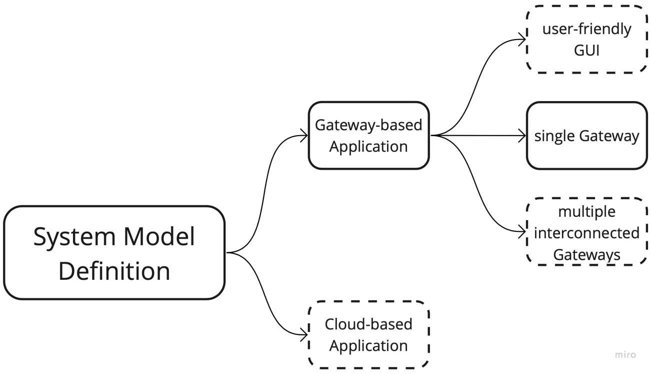


Figure 7.2: Graphical representation of the identified research opportunities concerning the System Model Definition

In the context of this thesis, the underlying system model requires exactly one Gateway and implements the Vulnerability Assessment and Management solution as application for this Gateway. With moderate effort, this can be extended towards systems with multiple interconnected Gateways and towards a cloud-based application. Identifying the most preferable user-friendly Graphical User Interface is another identified research opportunity in the field of human-centered computing.

Both Vulnerability Assessment and Trustworthiness Assessment require a concept for software inventory information and a corresponding format. The SPDX-compliant VAM SBOM format has been specified in Section 4.2.1. Figure 7.3 presents the research opportunities identified for the proposed software inventory information format.

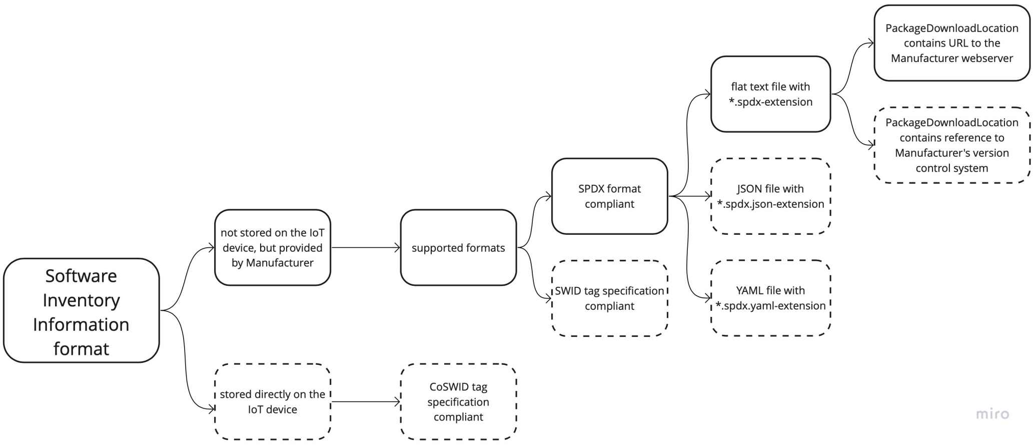

Software Inventory Information format
not stored on the IoT device, but provided by Manufacturer
supported formats
SPDX format compliant
flat text file with *.spdx-extension
PackageDownloadLocation contains URL to the Manufacturer webserver
PackageDownloadLocation contains reference to Manufacturer's version control system
JSON file with *.spdx.json-extension
YAML file with *.spdx.yaml-extension
SWID tag specification compliant
stored directly on the IoT device
CoSWID tag specification compliant
miro

Figure 7.3: Graphical representation of the identified research opportunities concerning the Software Inventory Information Format

In the context of this thesis, the software inventory information is not stored directly on the constrained IoT device, but is provided by the Manufacturer instead. Among the candidates, the SPDX format has been identified as the most suitable and has therefore been incorporated in the concept with its representation as flat text file. The *PackageDownloadLocation*-field has been chosen to contain the URL to the Manufacturer webserver for SBOM retrieval. With moderate effort, the architecture can be extended towards support of version control systems for SBOM retrieval and alternative file format such as JSON and YAML. In addition to the SPDX-compliant VAM SBOM format, support of alternative SBOM specifications such as SWID and CoSWID can be incorporated with moderate effort as well. The latter is characterized by a very lightweight footprint and would therefore support the extension of the SBOM retrieval process towards SBOM information that are stored directly on the IoT device.

This SBOM retrieval process is covered in the Device and Software Module Management process that has been described in Section 4.2.2. Figure 7.4 presents the corresponding identified research opportunities.

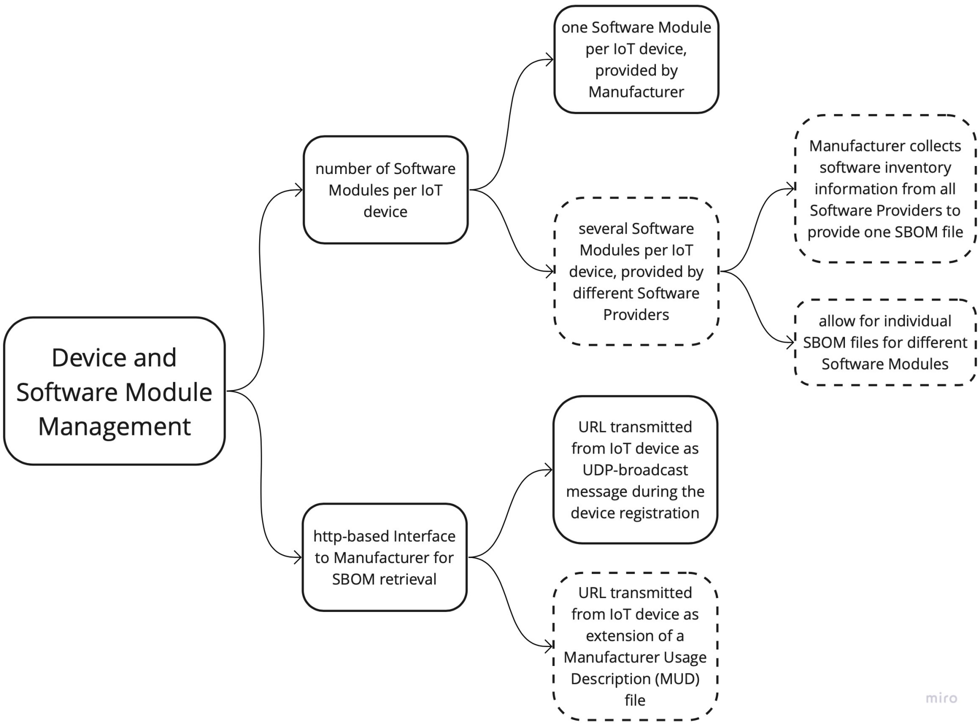

Device and Software Module Management
number of Software Modules per IoT device
one Software Module per IoT device, provided by Manufacturer
several Software Modules per IoT device, provided by different Software Providers
Manufacturer collects software inventory information from all Software Providers to provide one SBOM file
allow for individual SBOM files for different Software Modules
http-based Interface to Manufacturer for SBOM retrieval
URL transmitted from IoT device as UDP-broadcast message during the device registration
URL transmitted from IoT device as extension of a Manufacturer Usage Description (MUD) file
miro

Figure 7.4: Graphical representation of the identified research opportunities concerning the Device and Software Module Management

In the context of this thesis, the number of supported Software Modules per IoT device was limited to one Software Module that is bundled up by the Manufacturer. This is a common case in the IoT domain. However, the proposed concept will gain generality when being extended towards the support of several Software Modules per IoT device that can be provided by independent Software Providers.

In order to retrieve the VAM SBOM file, the corresponding webserver URL has to be transmitted to the Gateway. In the context of this thesis, this is effectively solved by broadcasting the URL from the IoT device during the device registration process. However, the proposed concept will gain generality when being extended towards the support of the URL being transmitted within a Manufacturer Usage Description file.

The actual Vulnerability Assessment process incorporated in the VAM solution has been described in detail in Section 4.3.1. Figure 7.5 presents the corresponding identified research opportunities.

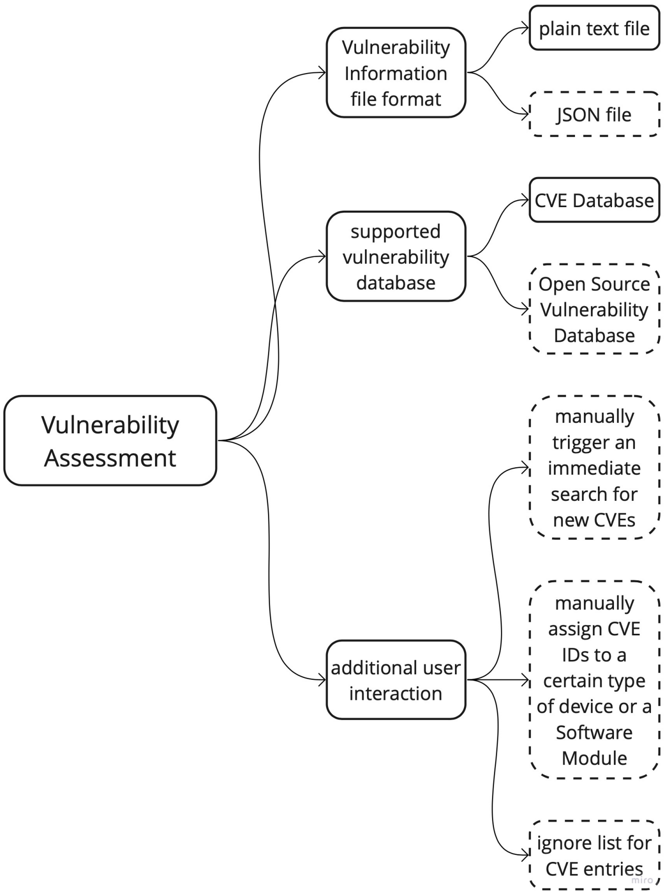

Vulnerability Assessment
Vulnerability Information file format
plain text file
JSON file
supported vulnerability database
CVE Database
Open Source Vulnerability Database
additional user interaction
manually trigger an immediate search for new CVEs
manually assign CVE IDs to a certain type of device or a Software Module
ignore list for CVE entries

Figure 7.5: Graphical representation of the identified research opportunities concerning the Vulnerability Assessment procedure

In addition to software inventory information, a suitable vulnerability information file format is required as well in order to specify the interface between Manufacturer and Gateway application. In the context of this thesis, the file format for the transmission of such vulnerability information is supported as plain text file and can be extended towards alternative file formats such as JSON.

In the context of this thesis, the Vulnerability Assessment process supports databases for CVE records. Since the CVE system serves as de-facto standard for the description and identification of disclosed vulnerabilities, this is a reasonable limitation. However, the proposed concept will gain generality when being extended towards the support of alternative vulnerability databases such as the Open Source Vulnerability Database.

As identified in Section 4.3.2, additional opportunities for user interaction can serve as valuable extension of the proposed concept in order to better cover the needs of more experienced users.

The actual Trustworthiness Assessment process incorporated in the VAM solution has been described in detail in Section 4.3.2. Figure 7.6 presents the corresponding identified research opportunities.

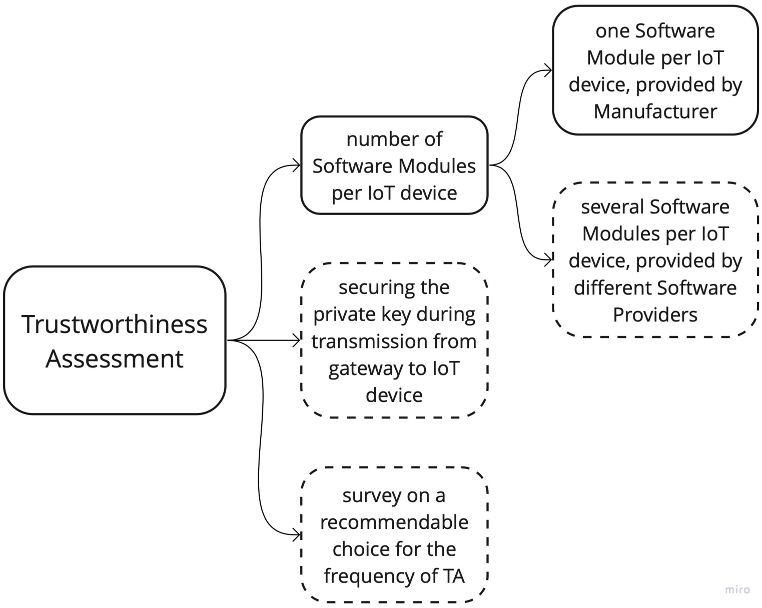

Trustworthiness Assessment
number of Software Modules per IoT device
one Software Module per IoT device, provided by Manufacturer
several Software Modules per IoT device, provided by different Software Providers
securing the private key during transmission from gateway to IoT device
survey on a recommendable choice for the frequency of TA
miro

Figure 7.6: Graphical representation of the identified research opportunities concerning the Trustworthiness Assessment procedure

In the context of this thesis, the Trustworthiness Assessment process supports one Software Module per IoT device. An extension towards larger number of Software Modules per device requires an extension of the underlying Trustworthiness Assessment process as well: the Remote Attestation Service will have to collect and transmit the Claims of several Software Modules and the Verifier will have to collect the Reference Evidences of multiple Software Modules from different independent Software Providers, just to name a few. This is not a trivial extension of the concept, but feasible.

In the context of this thesis, the symmetric key is exchanged during the device registration process using a potentially insecure channel. It may be more beneficial to e.g. ship the IoT device with a private key. Further research shall be conducted here.

Concerning the frequency with which Trustworthiness Assessment is to be performed, a survey on the most appropriate choice shall be conducted as well.

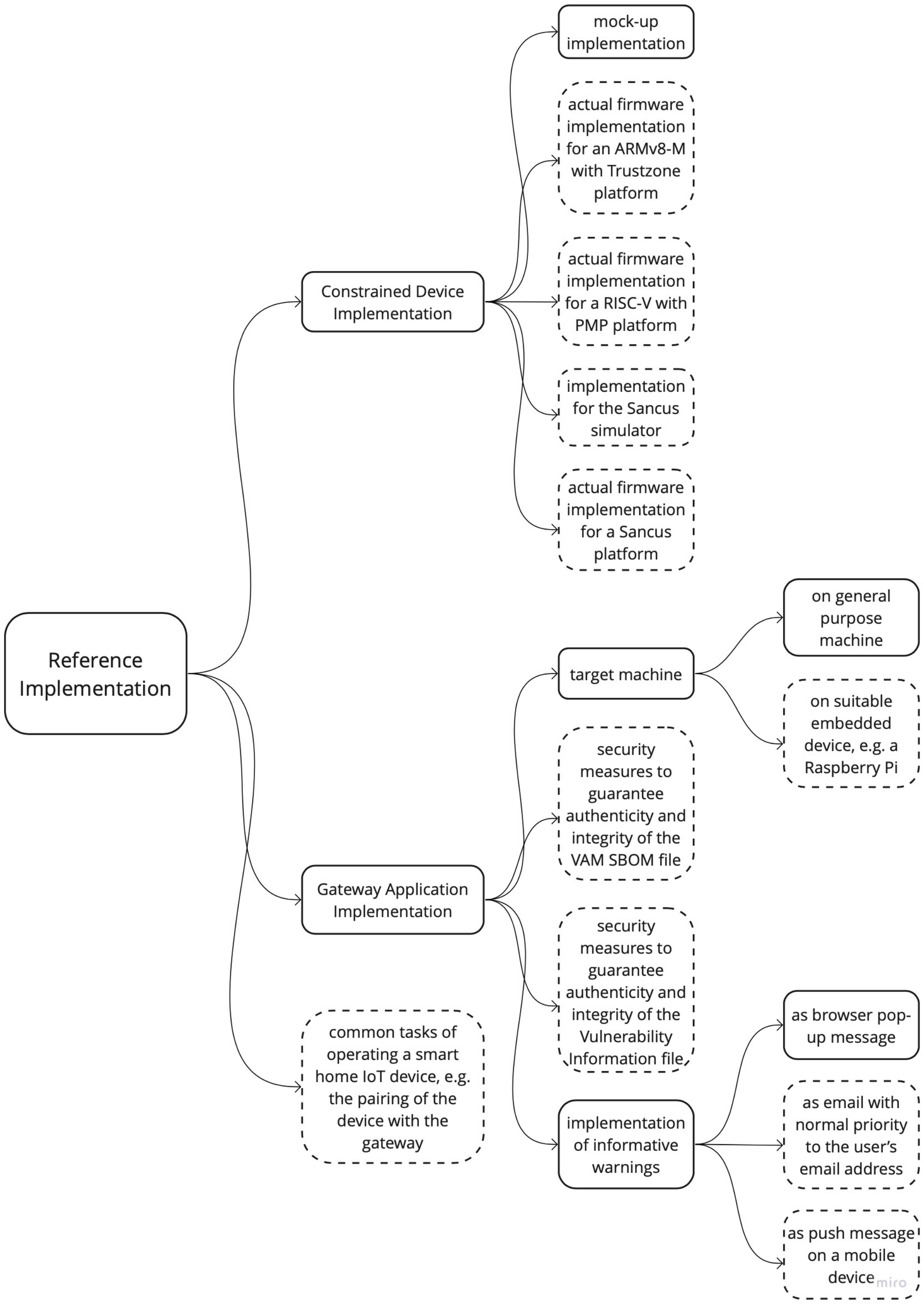

Reference Implementation
Constrained Device Implementation
mock-up implementation
actual firmware implementation for an ARMv8-M with Trustzone platform
actual firmware implementation for a RISC-V with PMP platform
implementation for the Sancus simulator
actual firmware implementation for a Sancus platform
Gateway Application Implementation
target machine
on general purpose machine
on suitable embedded device, e.g. a Raspberry Pi
security measures to guarantee authenticity and integrity of the VAM SBOM file
security measures to guarantee authenticity and integrity of the Vulnerability Information file
implementation of informative warnings
as browser pop-up message
as email with normal priority to the user's email address
as push message on a mobile device
common tasks of operating a smart home IoT device, e.g. the pairing of the device with the gateway
miro

Figure 7.7: Graphical representation of the identified research opportunities concerning the Reference Implementation

In addition to the specification of the system architecture, Chapter 5 provided a detailed description of a Proof of Concept (PoC) implementation that showed the feasibility and functionality of the proposed approach and the cooperating components. Figure 7.7 presents the corresponding identified engineering opportunities.

In the context of this thesis, a mock-up implementation of the constrained IoT device is provided. In order to prove the feasibility and correct cooperation of all relevant components, this is a feasible limitation. As outlined in Section 5, the essential characteristics of the concept remain the same with the actual implementation of firmware for actual development platforms. However, this is rather an engineering task, where technical problems of compilers, undocumented platforms, fast changes of hardware, etc. have to be solved. Therefore, it does not provide a fundamental gain in scientific knowledge. For the near future, this situation is expected to relax. As soon as architectures such as the discussed platforms become standard hardware, actual firmware implementations shall be incorporated in the reference implementation.

Concerning the implementation of the Gateway application, the target machine in the context of this thesis has been a general purpose Linux machine. This is a feasible limitation, since Gateways in smart home systems often provide the computational power and memory of a general purpose machine. However, the proposed concept will gain generality when being extended towards the support of more constrained target machines here, e.g. a Raspberry Pi.

In order to further improve the reference implementation, security measures to guarantee the authenticity and integrity of both the VAM SBOM file and the Vulnerability Information file can be put in place with moderate effort.

In the context of this thesis, the reference implementation includes informative user warnings as browser pop-up messages, depending on the individual risk assessment and per-device relevance level assigned by the user. In order to further improve the reference implementation, further channels for the transmission of such messages, e.g. e-mails or push messages, can be put in place with moderate effort.

In the context of this thesis, common tasks of operating a smart home IoT device, e.g. the pairing of the device with the gateway or the performance of actual tasks that are related to the device's purpose as smart home IoT device, are excluded, as they do not provide a contribution to the state of research. However, they may improve the suitability of the reference implementation for demonstration or teaching purposes and can be put in place with moderate effort.

In Chapter 6, this thesis was concluded with a qualitative evaluation of the proposed software system architecture and the reference implementation. The added value of this concept with regard to the resulting overhead has been discussed here.

To conclude, with the contributions of this thesis, the concept of a software system architecture for networked constrained IoT devices was proposed, that can provide the user with a profound insight into the security state of the connected IoT devices during runtime.

References

[1] G. Aggarwal. Using FreeRTOS on ARMv8-M Microcontrollers. Online, https://www.freertos.org/2020/04/using-freertos-on-armv8-m-microcontrollers.html, Accessed: 2022-07-15.

[2] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. Sok: Security evaluation of home-based iot deployments. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1362–1380, 2019.

[3] A. Amro. *IoT Vulnerability Scanning: A State of the Art*, pages 84–99. 12 2020.

[4] ARM Limited. ARM Products Processors. Online, https://www.arm.com/products/silicon-ip-cpu, Accessed: 2022-07-30.

[5] A. Banks, M. Kisiel, and P. Korsholm. Remote attestation: A literature review. Preprint, Online, https://www.researchgate.net/publication/351369060_Remote_Attestation_A_Literature_Review, Accessed: 2022-07-15, 05 2021.

[6] H. Birkholz, J. Fitzgerald-McKay, C. Schmidt, and D. Waltermire. Concise Software Identification Tags. Internet-Draft draft-ietf-sacm-coswid-21, Internet Engineering Task Force, Mar. 2022. Work in Progress.

[7] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan. Remote Attestation Procedures Architecture. Internet-Draft draft-ietf-rats-architecture-20, Internet Engineering Task Force, July 2022. Work in Progress.

[8] M. Björklund. The YANG 1.1 Data Modeling Language. RFC 7950, Aug. 2016.

[9] G. Blinowski and P. Piotrowski. *CVE Based Classification of Vulnerable IoT Systems*, pages 82–93. 05 2020.

[10] C. Bormann, M. Ersue, and A. Keränen. Terminology for Constrained-Node Networks. RFC 7228, May 2014.

[11] C. Bormann and P. E. Hoffman. Concise Binary Object Representation (CBOR). RFC 8949, Dec. 2020.

[12] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, Dec. 2017.

[13] V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

[14] Cybersecurity Insiders. Application security report. Online, https://www.tenable.com/whitepapers/cybersecurity-insiders-2018-application-security-report, Accessed: 2022-01-03, 2018.

[15] H. Djuitcheu, M. Debes, M. Aumüller, and J. Seitz. Recent review of distributed denial of service attacks in the internet of things. In *2022 5th Conference on Cloud and Internet of Things (CIoT)*, pages 32–39, 2022.

[16] P. Emami Naeini, Y. Agarwal, and L. Cranor. Specification for cmu iot security and privacy label (cispl 1.0). Online, https://www.iotsecurityprivacy.org/downloads/Privacy_and_Security_Specifications.pdf, Accessed: 2021-11-18, 01 2021.

[17] P. Emami Naeini, Y. Agarwal, L. Cranor, and H. Hibshi. Ask the experts: What should be on an iot privacy and security label? pages 447–464, 05 2020.

[18] P. Emami-Naeini, H. Dixon, Y. Agarwal, and L. F. Cranor. *Exploring How Privacy and Security Factor into IoT Device Purchase Behavior*, page 1–12. Association for Computing Machinery, New York, NY, USA, 2019.

[19] European Commission. Energy efficient products. Online, https://ec.europa.eu/info/energy-climate-change-environment/standards-tools-and-labels/products-labelling-rules-and-requirements/energy-label-and-ecodesign/energy-efficient-products_en, Accessed: 2021-09-14.

[20] R. T. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, June 2014.

[21] S. Figueroa-Lorenzo, J. Añorga, and S. Arrizabalaga. A survey of iiot protocols: A measure of vulnerability risk analysis based on cvss. *ACM Computing Surveys*, 53:1–53, 04 2020.

[22] FIRST.Org, Inc. (FIRST). Common vulnerability scoring system version 3.1, specification document, revision 1. https://www.first.org/cvss/specification-document, Accessed: 2022-07-21.

[23] Gartner. Gartner says 8.4 billion connected ”things” will be in use in 2017, up 31 percent from 2016. https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016, Accessed: 2021-07-15.

[24] Gartner. Market guide for vulnerability assessment. Online, https://www.gartner.com/doc/reprints?id=1-27GRFVF6&ct=210917&st=sb, Accessed: 2022-01-03, 06 2021.

[25] Hex Five Security, Inc. Multizone SDK. Online, https://github.com/hex-five/multizone-sdk/, Accessed: 2022-07-15.

[26] Hex Five Security, Inc. Multizone security datasheet v2020.01.09. Online, https://hex-five.com/wp-content/uploads/2020/01/multizone-datasheet-20200109.pdf, Accessed: 2022-07-30, 2020.

[27] IEEE. Standard for local and metropolitan area networks - secure device identity. *IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009)*.

[28] ISO/IEC 19770-2. Information technology - software asset management - part 2: Software identification tag, Feb. 2017.

[29] P. G. Kelley, J. Bresee, L. F. Cranor, and R. W. Reeder. A ”nutrition label” for privacy. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS ’09, New York, NY, USA, 2009. Association for Computing Machinery.

[30] P. G. Kelley, L. Cesca, J. Bresee, and L. F. Cranor. *Standardizing Privacy Notices: An Online Study of the Nutrition Label Approach*, page 1573–1582. Association for Computing Machinery, New York, NY, USA, 2010.

[31] R. Khoury, B. Vignau, S. Hallé, A. Hamou-Lhadj, and A. Razgallah. An analysis of the use of cves by iot malware. 12 2020.

[32] Knud Lasse Lueth (IoT Analytics GmbH). Iot 2019 in review: The 10 most relevant iot developments of the year. Online, https://iot-analytics.com/iot-2019-in-review/, Accessed: 2021-07-15.

[33] E. Lear, R. Droms, and D. Romascanu. Manufacturer Usage Description Specification. RFC 8520, Mar. 2019.

[34] E. Lear and S. Rose. Discovering and Retrieving Software Transparency and Vulnerability Information. Internet-Draft draft-ietf-opsawg-sbom-access-05, Internet Engineering Task Force, Mar. 2022. Work in Progress.

[35] F. Loi, A. Sivanathan, H. Habibi Gharakheili, A. Radford, and V. Sivaraman. Systematically evaluating security and privacy for consumer iot devices. pages 1–6, 11 2017.

[36] Lum, Brandon and Chang, Oliver (Google Open Source Security Team). Sbom in action: finding vulnerabilities with a software bill of materials. Online, https://security.googleblog.com/2022/06/sbom-in-action-finding-vulnerabilities.html, Accessed: 2022-06-18, June 2022.

[37] P. E. Naeini, J. Dheenadhayalan, Y. Agarwal, and L. F. Cranor. Which privacy and security attributes most impact consumers' risk perception and willingness to purchase iot devices? *2021 IEEE Symposium on Security and Privacy (SP)*, pages 519–536, 2021.

[38] National Institute of Standards and Technology, U.S. Department of Commerce. National vulnerability database. Online, https://nvd.nist.gov/developers/vulnerabilities, Accessed: 2022-07-29.

[39] NIST National Institute of Standards and Technology. Special publication 800-115, technical guide to information security testing and assessment. Online, https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf, Accessed: 2022-07-21.

[40] J. Noorman. *Sancus: A Low-Cost Security Architecture for Distributed IoT Applications on a Shared Infrastructure.* PhD thesis, 2017.

[41] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *Proceedings of the 22nd USENIX Conference on Security*, SEC'13, page 479–494, USA, 2013. USENIX Association.

[42] J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling. Sancus 2.0: A low-cost security architecture for IoT devices. *ACM Transactions on Privacy and Security (TOPS)*, 20(3):7:1–7:33, September 2017.

[43] OSV. A distributed vulnerability database for open source. Online, https://osv.dev, Accessed: 2022-07-29.

[44] S. Pinto and N. Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys*, 51:1–36, 01 2019.

[45] Principled Technologies. Comparing vulnerability and security configuration assessment coverage of leading vm vendors. Whitepaper Online, https://static.tenable.com/marketing/whitepapers/Whitepaper-Comparing_Vulnerability_and_Security_Configuration_Assessment_Coverage_of_Leading_VM_Vendors_Principled_Technologies.pdf, Accessed: 2022-01-03, 09 2019.

[46] Qualys, Inc. Continuous Monitoring: A New Blueprint for Achieving Continuous Security and Compliance. Online, https://www.qualys.com/forms/whitepapers/continuous-monitoring-blueprint-achieving-continuous-security-compliance, Accessed: 2022-01-04.

[47] Qualys, Inc. Qualys VMDR – All-in-One Vulnerability Management, Detection, and Response. Online, https://www.qualys.com/apps/vulnerability-management-detection-response/, Accessed: 2022-01-04.

[48] Qualys, Inc. Vulnerability Management. Online, https://www.qualys.com/apps/vulnerability-management/, Accessed: 2022-01-04.

[49] Rapid7. InsightVM. Online, https://www.rapid7.com/products/insightvm/, Accessed: 2022-01-05.

[50] Rapid7. Network vulnerability assessment programs and solutions. Online, https://www.rapid7.com/solutions/vulnerability-assessment/, Accessed: 2022-01-05.

[51] RISC-V International. Risc-v. Online, https://riscv.org, Accessed: 2022-06-14, 2021.

[52] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn. Iot goes nuclear: Creating a zigbee chain reaction. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 195–212, 2017.

[53] Sancus. Sancus TEE. Online, https://github.com/sancus-tee, Accessed: 2022-07-15.

[54] Y. Shen and P.-A. Vervier. *IoT Security and Privacy Labels*, pages 136–147. 06 2019.

[55] J. Spring, E. Hatleback, A. Householder, A. Manion, and D. Shick. Time to change the cvss? *IEEE Security and Privacy*, 19:74–78, 03 2021.

[56] STMicroelectronics. STM32CubeL5 MCU Firmware Package. Online, https://github.com/Poohl/STM32CubeL5/tree/linux, Accessed: 2022-07-15.

[57] tenable. Compare Nessus with industry vulnerability assessment solutions. Online, https://www.tenable.com/products/nessus/nessus-professional/competitive-comparison, Accessed: 2022-01-03.

[58] tenable. Nessus. Online, https://www.tenable.com/products/nessus, Accessed: 2022-01-03.

[59] The Linux Foundation. Spdx specification 2.2, 2020.

[60] The MITRE Corporation. About the cve progam. Online, https://www.cve.org/About/Overview, Accessed: 2022-01-06.

[61] The MITRE Corporation. Cve list downloads. Online, https://www.cve.org/Downloads, Accessed: 2022-01-28.

[62] The MITRE Corporation. Cve numbering authority (cna) rules. Online, https://www.cve.org/ResourcesSupport/AllResources/CNARules, Accessed: 2022-01-18.

[63] The MITRE Corporation. Process. Online, https://www.cve.org/About/Process, Accessed: 2022-01-18.

[64] The Pallets Projects. Flask. Online, https://palletsprojects.com/p/flask/, Accessed: 2022-07-18.

[65] U.S. Environmental Protection Agency. Learn about the fuel economy label. Online, https://www.epa.gov/greenvehicles/learn-about-fuel-economy-label, Accessed: 2021-09-14.

[66] U.S. Food and Drug Administration. Nutrition facts label better informs your food choices. Online, https://www.fda.gov/consumers/consumer-updates/nutrition-facts-label-better-informs-your-food-choices, Accessed: 2021-09-14, 12 2017.

[67] Wikipedia. Nessus (software). Online, https://de.wikipedia.org/wiki/Nessus_(Software), Accessed: 2022-07-26.

[68] WIRED Magazine. The botnet that broke the internet isn't going away. Online, https://www.wired.com/2016/12/botnet-broke-internet-isnt-going-away/, Accessed: 2021-05-03, 09 2016.

[69] WSGI.org. WSGI. Online, https://wsgi.readthedocs.io/en/latest/index.html, Accessed: 2022-07-18.

[70] YourThings. YourThings Scorecards. Online, https://yourthings.info/scorecards/, Accessed: 2021-12-1.

[71] YourThings. YourThings Scoring Rubric. Online, https://yourthings.info/method/, Accessed: 2021-11-30.

[72] K. Zandberg, K. Schleiser, F. J. Acosta Padilla, H. Tschofenig, and E. Baccelli. Secure firmware updates for constrained iot devices using open standards: A reality check. *IEEE Access*, PP:1–1, 05 2019.

Appendices

Appendix A

Summary in German (Zusammenfassung)

Die Popularität und Verbreitung von Geräten des Internets der Dinge (engl. Internet of Things, IoT) nimmt ständig zu. Sie haben Einzug in unser tägliches Leben gehalten und verwandeln unsere Wohnumgebung zunehmend in ein intelligentes Zuhause. Die meisten dieser eingeschränkten Geräte verfügen jedoch nicht über genügend Rechenleistung, Speicher und Akkulaufzeit, um Sicherheitsfunktionen zu implementieren, die für allgemeine Personal Computer üblich sind. Mit der zunehmenden Zahl der vernetzten IoT-Geräte für Verbraucher steigen daher auch deren Angriffsfläche und Schwachstellen.

Die vorliegende Arbeit widmet sich diesem Sicherheitsproblem, indem sie einen neuartigen Ansatz für einen Runtime IoT Security Score vorstellt, der dem unerfahrenen Benutzer eines Smart-Home-Systems einen tiefen Einblick in den Sicherheitszustand der angeschlossenen IoT-Geräte zur Laufzeit gibt. Dies wird durch die Kombination von Vulnerability Assessment mit einer Bewertung der Vertrauenswürdigkeit der angeschlossenen Geräte erreicht. Dies stellt einen neuartigen Ansatz darf und leistet damit einen sehr wertvollen Beitrag zum aktuellen Stand der Forschung.

Neben dem Runtime Security Score wird als weiterer wichtiger Beitrag dieser Arbeit ein ganzheitliches Konzept für eine Vulnerability Assessment and Management (VAM) Lösung vorgeschlagen. Die effektive und funktionale Interoperabilität aller relevanten Komponenten, die in diesem Konzept spezifiziert sind, wird mit einer Proof of Concept Implementierung gezeigt.

Appendix B

Listing 1: XML Schema Design for the CVE List available for download[61]

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://cve.mitre.org/cve/downloads/1.0"
    targetNamespace="http://cve.mitre.org/cve/downloads/1.0"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
        ↪ version="1.0">

    <!-- *********************************************************** -->
    <!-- Changelog:
                1.0 - Initial version
    -->
    <!-- *********************************************************** -->
    <xsd:annotation>
        <xsd:documentation xml:lang="en"> Simple schema that defines the
    ↪ format of the CVE List
            provided by MITRE </xsd:documentation>
    </xsd:annotation>

    <!-- *********************************************************** -->
    <!-- Start Item Element Definition -->
    <!-- *********************************************************** -->
    <xsd:element name="cve">
        <xsd:annotation>
            <xsd:documentation xml:lang="en"> cve is the top level element
    ↪ of the CVE List provided
                by MITRE. It represents holds all CVE Items. </xsd:
                    ↪ documentation>
        </xsd:annotation>
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="item" type="ItemType" minOccurs="1"
    ↪ maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="schemaVersion" type="xsd:token" use="
                ↪ optional"/>
        </xsd:complexType>
    </xsd:element>

    <!-- *********************************************************** -->
    <!-- Simple Types -->
    <!-- *********************************************************** -->
    <!-- CUSTOM TYPE DEFINITIONS-->
    <xsd:simpleType name="typeEnumType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="CAN"/>
            <xsd:enumeration value="CVE"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="statusEnumType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="Entry"/>
            <xsd:enumeration value="Candidate"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!-- need to verify enumeration -->
    <xsd:simpleType name="simplePhaseEnumType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="Proposed"/>
            <xsd:enumeration value="Interim"/>
            <xsd:enumeration value="Modified"/>
            <xsd:enumeration value="Assigned"/>
        </xsd:restriction>
    </xsd:simpleType>

    <!--  ********************************************************** -->
    <!--  Complex Types -->
    <!--  ********************************************************** -->
    <xsd:complexType name="ItemType">
        <xsd:sequence>
            <xsd:element name="status" type="statusEnumType" minOccurs="1"
    ↪ maxOccurs="1"/>
            <xsd:element name="phase" type="specificPhaseType" minOccurs="0"
                ↪  maxOccurs="1"/>
            <xsd:element name="desc" type="xsd:string" minOccurs="1"
                ↪ maxOccurs="1"/>
            <xsd:element name="refs" type="refsType" minOccurs="1" maxOccurs
                ↪ ="1"/>
            <xsd:element name="votes" type="votesType" minOccurs="0"
                ↪ maxOccurs="1"/>
            <xsd:element name="comments" type="commentsType" minOccurs="0"
                ↪ maxOccurs="1"/>
        </xsd:sequence>
        <!--Need to Verify Enumeration-->
        <xsd:attribute name="type" type="typeEnumType" use="required"/>
        <xsd:attribute name="name" type="xsd:token" use="required"/>
        <xsd:attribute name="seq" type="xsd:token" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="commentsType">
        <xsd:sequence>
            <xsd:element name="comment" minOccurs="0" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                            <xsd:attribute name="voter" type="xsd:token" use
                                ↪ ="required"/>
                        </xsd:extension>
                    </xsd:simpleContent>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="votesType">
        <xsd:sequence>
            <xsd:element name="accept" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="count" type="xsd:token" use
                    ↪ ="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="modify" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="count" type="xsd:token" use
                    ↪ ="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="noop" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="count" type="xsd:token" use
                    ↪ ="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="recast" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="count" type="xsd:token" use
                    ↪ ="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="reject" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="count" type="xsd:token" use
                    ↪ ="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="reviewing" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="count" type="xsd:token" use
                    ↪ ="required"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="revote" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
        <xsd:simpleContent>
```

```
                        <xsd:extension base="xsd:string">
                            <xsd:attribute name="count" type="xsd:token" use
                                ↪ ="required"/>
                        </xsd:extension>
                    </xsd:simpleContent>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>


    <xsd:complexType name="specificPhaseType">
        <xsd:simpleContent>
            <xsd:extension base="simplePhaseEnumType">
                <xsd:attribute name="date" type="xsd:token" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>

    <xsd:complexType name="refsType">
        <xsd:annotation>
            <xsd:documentation>holds all hyperlink elements</xsd:
    ↪ documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="ref" type="refType" minOccurs="0" maxOccurs="
    ↪ unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="refType">
        <xsd:annotation>
            <xsd:documentation>Holds individual hyperlink element</xsd:
    ↪ documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="source" type="xsd:token" use="required"
                    ↪ />
                <xsd:attribute name="url" type="xsd:anyURI" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:schema>
```

Appendix C

Listing 2: An example for a minimal XML file for the SWID tag specification described in Chapter 2.2, taken from [28]

```
<?xml version=˜"1.0˜" encoding=˜"utf-8˜"?>
<SoftwareIdentity
        xmlns=˜"http://standards.iso.org/iso/19770/-2/2015/schema.xsd˜"+
        xmlns:xsi=˜"http://www.w3.org/2001/XMLSchema-instance˜"
        xmlns:ds=˜"http://www.w3.org/2000/09/xmldsig#˜"
        xmlns:SHA256=˜"http://www.w3.org/2001/04/xmlenc#sha256˜"
        xsi:schemaLocation=˜"http://standards.iso.org/iso/19770/-2/2015/
            ↪ schema.xsd˜"
        name=˜"ACME System Protection˜"
␣␣␣␣␣␣␣␣tagId=˜"iso-sid-app-acme-endpoint-protection-v12-1-mp1˜"
␣␣␣␣␣␣␣␣version=˜"12.1.1˜"
␣␣␣␣␣␣␣␣versionScheme=˜"multipartnumeric˜"
␣␣␣␣␣␣␣␣corpus=˜"true˜">
<Payload>
<File␣name=˜"EPV12.cab˜"␣size=˜"1024000˜"
SHA256:hash=˜"
    ↪ a314fc2dc663ae7a6b6bc6787594057396e6b3f569cd50fd5ddb4d1bbafd2b6a˜"␣/>
<File␣name=˜"installer.exe˜"␣size=˜"524012˜"
SHA256:hash=˜"54
    ↪ e6c3f569cd50fd5ddb4d1bbafd2b6ac4128c2dc663ae7a6b6bc67875940573˜"␣/>
</Payload>
</SoftwareIdentity>
```

Appendix D

Listing 3: The mud-sbom augmentation to the MUD YANG model described in Chapter 2.2, taken from [34]

```
file "ietf-mud-sbom@2020-03-06.yang"
module ietf-mud-sbom {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-mud-sbom";
  prefix mud-sbom;
  import ietf-inet-types {
    prefix inet;
  }
  import ietf-mud {
prefix mud; }
  organization
    "IETF␣OPSAWG␣(Ops␣Area)␣Working␣Group";
contact "WG
␣␣␣␣␣Web:␣http://tools.ietf.org/wg/opsawg/
␣␣␣␣␣WG␣List:␣opsawg@ietf.org
␣␣␣␣␣Author:␣Eliot␣Lear␣lear@cisco.com␣";
  description
    "This␣YANG␣module␣augments␣the␣ietf-mud␣model␣to␣provide␣for
␣␣␣␣␣reporting␣of␣SBOMs.
␣␣␣␣␣Copyright␣(c)␣2019␣IETF␣Trust␣and␣the␣persons␣identified␣as
␣␣␣␣␣authors␣of␣the␣code.␣␣All␣rights␣reserved.
␣␣␣␣␣Redistribution␣and␣use␣in␣source␣and␣binary␣forms,␣with␣or
␣␣␣␣␣without␣modification,␣is␣permitted␣pursuant␣to,␣and␣subject␣to
␣␣␣␣␣the␣license␣terms␣contained␣in,␣the␣Simplified␣BSD␣License␣set
␣␣␣␣␣forth␣in␣Section␣4.c␣of␣the␣IETF␣Trusts␣Legal␣Provisions
␣␣␣␣␣Relating␣to␣IETF␣Documents
␣␣␣␣␣(https://trustee.ietf.org/license-info).
␣␣␣This␣version␣of␣this␣YANG␣module␣is␣part␣of␣RFC␣XXXX
␣␣␣(https://www.rfc-editor.org/info/rfcXXXX);␣see␣the␣RFC␣itself␣for
␣␣␣full␣legal␣notices.
␣␣␣The␣key␣words␣␣MUST␣,␣␣MUST␣NOT␣,␣␣REQUIRED␣,␣␣SHALL␣,␣␣SHALL
␣␣␣NOT␣,␣␣SHOULD␣,␣␣SHOULD␣NOT␣,␣␣RECOMMENDED␣,␣␣NOT␣RECOMMENDED␣,
␣␣␣␣MAY␣,␣and␣␣OPTIONAL␣␣in␣this␣document␣are␣to␣be␣interpreted␣as
␣␣␣described␣in␣BCP␣14␣(RFC␣2119)␣(RFC␣8174)␣when,␣and␣only␣when,
␣␣␣they␣appear␣in␣all␣capitals,␣as␣shown␣here.␣␣";
revision 2020-03-06 {
  description
    "Initial␣proposed␣standard.";
  reference
    "RFC␣XXXX:␣Extension␣for␣MUD␣Reporting";
}
grouping mud-sbom-extension {
  description
    "SBOM␣extension␣grouping";
  list sboms {
    key "version-info";
    leaf version-info {
      type string;
      description
        "A␣version␣string␣that␣is␣applicable␣for␣this␣SBOM␣list␣entry.
␣␣␣␣␣␣␣␣␣The␣format␣of␣this␣string␣is␣left␣to␣the␣device␣manufacturer.
␣␣␣␣␣␣␣␣␣How␣the␣network␣administrator␣determines␣the␣version␣of
␣␣␣␣␣␣␣␣␣software␣running␣on␣the␣device␣is␣beyond␣the␣scope␣of␣this
␣␣␣␣␣␣␣␣␣memo.";
}
choice sbom-type {
  case url {
    leaf sbom-url {
      type inet:uri;
      description
        "A␣statically␣located␣URI.";
    }
  }
  case local-uri {
    leaf-list sbom-local {
      type enumeration {
enum coap {
  description
    "Use␣COAP␣schema␣to␣retrieve␣SBOM";
}
enum coaps {
  description
              "Use␣COAPS␣schema␣to␣retrieve␣SBOM";
          }
          enum http {
            description
              "Use␣HTTP␣schema␣to␣retrieve␣SBOM";
          }
          enum https {
            description
              "Use␣HTTPS␣schema␣to␣retrieve␣SBOM";
          }
        }
        description
          "The␣choice␣of␣sbom-local␣means␣that␣the␣SBOM␣resides␣at
␣␣␣␣␣␣␣␣␣␣a␣location␣indicated␣by␣an␣indicted␣scheme␣for␣the
␣␣␣␣␣␣␣␣␣␣device␣in␣question,␣at␣well␣known␣location
␣␣␣␣␣␣␣␣␣␣␣␣/.well-known/sbom␣.␣␣For␣example,␣if␣the␣MUD␣file
␣␣␣␣␣␣␣␣␣␣indicates␣that␣coaps␣is␣to␣be␣used␣and␣the␣host␣is
␣␣␣␣␣␣␣␣␣␣located␣at␣address␣10.1.2.3,␣the␣SBOM␣could␣be␣retrieved
␣␣␣␣␣␣␣␣␣␣at␣␣coaps://10.1.2.3/.well-known/sbom␣.␣␣N.B.,␣coap␣and
␣␣␣␣␣␣␣␣␣␣http␣schemes␣are␣NOT␣RECOMMENDED.";
} }
    case contact-info {
      leaf contact-uri {
        type inet:uri;
        description
          "This␣MUST␣be␣either␣a␣tel,␣http,␣https,␣or
␣␣␣␣␣␣␣␣␣␣␣mailto␣uri␣schema␣that␣customers␣can␣use␣to
␣␣␣␣␣␣␣␣␣␣␣contact␣someone␣for␣SBOM␣information.";
} }
    description
      "choices␣for␣SBOM␣retrieval.";
  }
  description
    "list␣of␣methods␣to␣get␣an␣SBOM.";
}
}
  augment "/mud:mud" {
    description
      "Add␣extension␣for␣SBOMs.";
    uses mud-sbom-extension;
} }
```