Extended Definition of the Proposed Open Standard for *IoT* Device Identific*A*tion and Reco*G*nition (*IoTAG*)

Lukas Hinterberger*
and Bernhard Weber†
Dept. Electrical Engineering and Information Technology
Ostbayerische Technische Hochschule
Regensburg, Germany
email:
lukas.hinterberger@st.oth-regensburg.de*
bernhard1.weber@st.oth-regensburg.de†

Sebastian Fischer
Secure Systems Engineering
Fraunhofer AISEC
Berlin, Germany
email:
sebastian.fischer@aisec.fraunhofer.de

Katrin Neubauer‡
and Rudolf Hackenberg§
Dept. Computer Science and Mathematics
Ostbayerische Technische Hochschule
Regensburg, Germany
email:
katrin1.neubauer@oth-regensburg.de‡
rudolf.hackenberg@oth-regensburg.de§

***Abstract*—Internet of Things (IoT) devices are critical to operate and maintain, because of their number and high connectivity. A lot of security issues concern IoT devices and the networks they are integrated. To help getting an overview of an IoT network, the devices and the security, we propose a scoring system to get a good impression of IT security. This system generates single scores for each device, using features like encryption, update behavior, etc. Furthermore, a summarized score for the whole network is calculated, to show the status of the network security in an easy way for the administrator. To enable the scoring system, a precise list of the existing devices and their operating status is necessary. To achieve this, we present an open standard for the *IoT* Device Identific*A*tion and Reco*G*nition (short *IoTAG*), which requires that devices report, e.g., their name, an unique ID, the firmware version and the supported encryption. The proposed standard is described in detail and an implementation guideline is given in this paper. Additionally, information about how to realize the serialization, the integrity and the communication with IoTAG.**

***Keywords*—Internet of Things; device identification; open standard; IoTAG; security rating.**

I. Introduction

This paper extends the already published paper "IoTAG: An Open Standard for IoT Device IdentificAtion and RecoGnition" [1] with more detailed information and definitions of the proposed open standard *IoT* Device Identific*A*tion and Reco*G*nition (IoTAG) and a guideline for the implementation.

Internet of Things (IoT) continues to be an innovation topic and trend in the industrial sector and smart homes. The development of new IoT devices, systems and services are progressing extremely fast. This raises the problem that the security risks of IoT devices, networks and services are underestimated or not even taken into account at all. It is precisely the reason that leads to insecure devices. An example of this would be the missing encryption or authentication. Some of these risks and vulnerabilities lead to attacks such as destroy the device. Serious attacks can lead to hijacking of complete company networks. In general, a large number of IoT devices are critical to operate [2] [3].

There are some solutions to these security problems. For example, with device detection, it should be possible for the user to detect devices in the IoT network and also check the software status. At present, there are no existing frameworks, software or systems for automated device scanning. With individual steps, it is possible to obtain individual units or parts of the required information. For example, addressable network ports can be found with the network scanners Nmap [4] or Fing [5]. The problem with Nmap or Fing is that the result of the scan will not be analysed or evaluated. Only a technical user or expert can perform and understand this technical analysis. A non-technical user needs a simple scoring system for IoT devices and networks.

The basic idea of our research project can be summarised as follows. The IoT devices of a network are identified and get a security rating during an initial scan. The rating is based on the provided metadata, information collected by the scanner itself and a database of known vulnerabilities, which are collected from multiple publicly available sources. An overall network rank results from the inheritance of the individual ranks. As part of the visual presentation for the end user (non-technical user), the rating should be shown as well as a list of all known vulnerabilities and general risks of the IoT setup.

The aim of the IoTAG project is to propose an open standard for IoT devices. This standard is intended to provide the required metadata for the risk and security rating and to verify the authenticity of the received information.

This paper begins with our hardware setup to test the idea of a network security score. Next, we started with the device scanning process and found out that it is not possible to get all the requirement information for our security evaluation and in some cases not even the device name or type at all.

We continue with the security criteria needed to create a device rating and then created the actual rating from this. As

a result, the entire network can be evaluated based on the individual scores.

As already discussed, the detection of the devices and their further details is not possible with available tools, we present a proposal for a standard which makes this possible. Since, the standard is still under development, a newer, more detailed version than in our last paper [1] is presented here. Furthermore, we have started with a sample implementation and give guidelines regarding the development.

The paper is structured as follows. Section II describes the related work. Section III introduces our hardware setup and device scanning, while Section IV defines the security criteria. Section V shows the device rating and Section VI the results. The standard IoTAG is presented in Section VII, followed by the conclusion in Section VIII.

II. Related Work

A popular solution for the identification of devices is the utilisation of so-called device fingerprints. Those can be used for basic categorisation and classification as secure or insecure. Miettinen et al. [6] show this procedure with device fingerprints for categorisation and classification. The development of an anomaly recognition system for smart home networks is taking place on the basis of a research project [7]. The subject device identification with device fingerprints and similar approaches covering by several publications [8]–[11]. The current working approach in the area of IoT device detection is shown. Currently, it is not possible to identify detailed information such as the current firmware version or a device ID for further recognition.

Khaled et al. [12] and Kaebisch et al. [13] proposing a machine readable description for IoT devices. These descriptions are not intended for risk and security device ratings. They are intended only for the functionality of a device and cover information like the turn off command. The goal of IoTAG is to get the security characteristics of an IoT device and no further information of the functions.

The Thing Description (TD) [14] by the World Wide Web Consortium (W3C) provides metadata of a device, e.g., stored setting or sensor data. An optional "Security" information on the authorization procedures is also available. But this is only a small part of all information needed for a security evaluation.

IoT Sentinel is a tool which detects and evaluates devices by creating a fingerprint and comparing it to a database of known devices. It is also able to isolate devices which are classified as insecure and filter their network traffic. The tool was developed by a team of researchers from the Technical University of Darmstadt, the Aalto University and the University of Helsinki [6]. In contrast to the commercial solutions discussed later in this Section, an example implementation is available under the MIT license, which allows for code reviews and further development [15].

Another approach is the security and privacy assessment for IoT devices with different security ratings. To calculate the device rating, this approach [16] uses the information protocols, open ports and encryption. This approach is very similar but it is not very flexible and user-friendly. The reason for this is the missing weighting of each criteria and the missing overall score of the network. Park et al. [17] and Ali et al. [18] show a list of security requirements for IoT services, which can be used as a basis for a risk assessment. These security requirements can be used to evaluate the weighting. A further approach to generate a metric value for the security of an IoT device is to use vulnerabilities and known exploits [19].

There are also multiple commercially available IoT security evaluation tools. One is Norton Core Router, which is developed by the anti-malware vendor Symantec Corporation [20]. Another one is Avira SafeThings, which is developed by Avira [21].

The scoring system, Norton Core Security Score, is deficit based, meaning it starts at 500 and each problem found reduces the score until it reaches the lowest score of 50. For example, if the firmware version of the router is outdated the score gets deducted by 10%. Not installing the client software "Norton Security" on a compatible device lowers the score. Most of the examples in the manual are not IoT related which indicates that the device detection is not detailed enough to provide the scoring algorithm with the needed information. One of the examples, which also applies to an IoT device, is ignoring an vulnerability or intrusion alert [22]. The vulnerability detection could be based on scanning for open ports and detecting the version of the software listening on them. This information could then be used to search for known vulnerabilities in that specific software, e.g., a web server. The exact way could not be examined, as the router was discontinued on January 31, 2019 and, according to the manufacturer, will only return as a software based solution in the future [23].

Avira lists a per-device security score as a feature. This score seems to be completely intransparent as it is neither mentioned nor described in the manual or any other resource about the device. Knowing how Avira classifies the individual devices, SafeThing should have enough information to give a helpful score, but as it is not described anywhere and as the device is currently unavailable for purchase, the scoring part could not be validated.

III. Hardware Setup and Device Scanning

The test environment consists of ten devices, as stated in Table I, which were selected to reflect a variety of typical IoT devices found in a home environment. A first basic network scan with nmap [4] resulted in a list of found devices and their hostnames. While some of the devices use meaningful hostnames, the list also contains a lot of generic names like "ESP" and empty rows. To gain more information, an extended scan, which includes a scan for open network ports, can be done as shown in Table II. This scan results in a list of found open ports and how the open port was found. Additionally, nmap lists the service which is registered for the found port at

the IANA (Internet Assigned Numbers Authority) [24]. This provides a first look at which services are used by the devices and how they communicate. For example, port 80 is specified to be used for http servers, which utilise unencrypted data transmission.

TABLE I. HARDWARE OVERVIEW

device	hostname
Amazon Echo 2	amazon-183e3c119
Apple iPhone 5	Kluges-iPhone
Floureon M32B	
Google Home mini	Google-Home-Mini
Grandstream GXP1610	
Raspberry Pi 3 Model B	raspberrypi
Sonoff Wi-Fi Smart Switch	ESP_6A768B
Wi-Fi Smart Bulb	ESP_4C3210
Wi-Fi Smart Plug	ESP_3D1EB6
Wi-Fi Touch Switch	ESP_469ACF

TABLE II. OVERVIEW OF OPEN AND RESTRICTED PORTS

Raspberry Pi 3 Model B				
port		**state**	**service**	**reason**
22	TCP	open	ssh	syn-ack
53	TCP	open	domain	syn-ack
Sonoff Wi-Fi Smart Switch				
port		**state**	**service**	**reason**
		restricted		
Wi-Fi Touch Switch				
port		**state**	**service**	**reason**
8081	TCP	open	blackice-icecap	syn-ack
Wi-Fi Smart Plug				
port		**state**	**service**	**reason**
10000	TCP	open	snet-sensor-mgmt	syn-ack
Grandstream GXP1610				
port		**state**	**service**	**reason**
22	TCP	open	ssh	syn-ack
80	TCP	open	http	syn-ack

After all, the information provided by these scans is still not enough to know the exact device model used in the network. For example, the running services on a device could vary based on the configuration of a device. The same applies to hostnames: there are no rules or limitations what devices can use as their hostname. Many devices, for example the iPhone, even allow the user to change it to a custom one.

IV. Security Criteria

For an automated security evaluation of an IoT network, a general applicable evaluation scheme is needed. The scheme has to be modular to allow for different devices being evaluated based on the used technologies. Every module is limited to a specific part of the device and the regarding security risks. The individual results can then be weighted against each other to obtain an overall evaluation of a device.

The scheme described below serves as a first approach for the evaluation of individual devices. It shall serve as a basis for the definition of the desired scan results and device properties and illustrate their later use.

TABLE III. SECURITY CRITERIA

audit criteria			**score**
radio technology			
WPA/WEP or no encryption			0
WPA2/WPA3			2
Bluetooth version			0-2
ZigBee version			0-2
manufacturer			
unknown manufacturer			0
usual patch time			0-2
experience			0-2
known unpatched devices			0-2
bug bounty program			0/2
services			
service	**default port**	**comment**	
HTTP	80	unencrypted login details	0
MQTT	1883	unencrypted control data	0
UPnP	49152/1900	firewall manipulation	0
rtsp	554	unencrypted video data	0
SIP	5060	unencrypted	0
service	**default port**	**comment**	
HTTPS	443	encrypted	2
MQTTS	8883	encrypted	2
SCP	10001	encrypted	2
SIPS	5061	encrypted	2
SSH	22	encrypted	2
LAN and WAN communication			
service	**default port**	**comment**	
HTTP	80	unencrypted login details	0
MQTT	1883	unencrypted control data	0
UPnP	49152/1900	firewall manipulation	0
rtsp	554	unencrypted video data	0
SIP	5060	unencrypted	0
service	**default port**	**comment**	
HTTPS	443	encrypted	2
MQTTS	8883	encrypted	2
SCP	10001	encrypted	2
SIPS	5061	encrypted	2
SSH	22	encrypted	2
other			
vulnerable to replay attacks			0
create own Wi-Fi			0
data retrieval without authentication			0
vulnerable to jamming			0-2
vulnerable to Denial of Service (DoS)			0-2
insecure configuration			0
continuous device number			0-2
known vulnerabilities			0
support lifetime			0-2
insecure / default password			0/2
firmware version			0-2
technical guidelines			0-2
certification			0-2

The aforementioned scheme utilises a three-value score system reaching from zero to two. If a module detects a critical security violation it results in a score of zero. A potential, but non-critical, violation would result in a score of one. If no problems are found, the score would be two. Similarly to the overall score calculation, each module runs several individual evaluations and weights them against each other to calculate the resulting score. A list of the modules can be found in Table III and are described in the following Subsections.

A. Physical connection

Although the software properties of a device play the main role for security risks, the physical connection to a

network could also be a potential attacking point. Therefore, a distinction is made between wired and wireless connections. If the connection is wireless, the used encryption technology is taken into account. A wireless connection results in a score of one, weighted against the score of the used encryption. A wired connection on the other hand is scored two, as physical access would be needed to interfere the connection.

Wireless connections can be unencrypted or use a variety of different encryption technologies. Obviously, the use of unencrypted or open Wi-Fi (wireless local area network) is considered dangerous and scored with zero points. The older Wi-Fi encryption standards, namely WPA (Wi-Fi Protected Access) and WEP (Wired Equivalent Privacy), are also scored zero points, as they use "RC4" (Rivest Cipher 4) for the encryption which is considered broken [25]. The use of the newer WPA2 and WPA3 standards results in the highest score of two.

B. Services

This module looks at the services, which are reachable from the network for the communication with a device. The rating of the security level is done for each listed service separately, but in this case the lowest individual score is used and not a weighted average. The evaluation is based on the protocol and encryption used. This is done using black and white lists. Services on the black lists either use obsolete protocols, which are considered vulnerable, vulnerable encryption or are completely unencrypted. A blacklisted service results in zero points, a whitelisted one in two points and if the service is not listed it is scored one point.

C. Communication

Besides the communication from the network, devices are also able to communicate by themselves to other devices. For example, many IoT devices connect to servers in the cloud or a local gateway. This communication is also evaluated based on the encryption used. As this type of connection cannot be detected by a scan of the network, the actual traffic needs to be analysed. This analysis also utilises a predefined list of protocols for the evaluation. Furthermore, the communication is split into LAN (local area network) and WAN (wide area network) in order to take the different security requirements in account. For example, data sent over the WAN leaves the relatively protected home environment and could therefore be seen by third parties. As both categories look quite similar, they are displayed as one in Table III.

In addition to the encryption, it is also possible to check the number of external resources a device communicates with and where they are located. An additional point that could be evaluated in the course of this analysis is whether a device requires a continuous connection to a cloud service. If no such service is used, two points could be are awarded. Otherwise, one point could be deducted.

D. Default passwords

When talking about passwords, a major security concern is the use of default credentials, which apply to all devices of the same type and manufacturer. If an attacker knows the credentials for a device, most of the other security measures are useless. Therefore, this module checks if a login with known credentials is possible. If it is the case, the score is set to zero. If the login was unsuccessful, the score is two, but is deducted by one if the username cannot be changed by the user.

E. Firmware version

Outdated firmware or unmaintained firmware increases the possibility of security vulnerabilities. Most known vulnerabilities are collected and provided to others in form of several vulnerability databases. CVE (Common Vulnerabilities and Exposures), for example, is one of the popular lists of known vulnerabilities, which is maintained by the MITRE Corporation and contains nearly 140000 entries [26]. As this information is also available to potential attackers, it allows for systematic attacks against outdated or unfixed firmware versions. Therefore, the module needs to be able to detect the installed version and check, if new versions are available. Additionally, it has to search vulnerability databases for known issues with the installed firmware version. If known vulnerabilities are found and no update is available, the lowest score of zero is given. If an update is available, the score is one and the user needs to be notified of this problem. If nothing of the mentioned applies, the device is up to date and is awarded with the highest score.

V. Device Rating

In this section, we describe the proceeding to receive the information for all the security criteria and how they are rated in detail.

A. Physical connection

In our test environment, a Raspberry Pi serves as a router through which all devices are connected to the network. For wireless and wired connections, different address spaces were used, which means that the physical connection of the individual devices can be determined via these address spaces.

The encryption technology of the wireless network can be taken from the router configuration. Since the software used for the access point is "hostapd", the configuration can be done in the "/etc/hostapd" file. The entry "wpa=2" indicates the exclusive use of the WPA2 standard, which in turn results in a score of two points for each device. If an insecure technology is used, this also affects the rating of each device, as the entire network is weakened. In this case, all devices in this category must be scored zero.

B. Services

The running services are recorded by scanning the individual network components. “Nmap” is used for this scanning process [27], which provides the results shown in Table IV.

TABLE IV. PORT SCAN

port	protocol
22	ssh
80	http
5060	sip

Based on these results and the categorization lists already mentioned, the device can be classified. The example in Table IV results in a score of 0.66 points, since http and sip are scored zero and ssh two points.

C. Communication

The communication of the devices with external resources is evaluated by recording and analyzing the network traffic. The MAC address of the local resource, source and destination port and the communication protocol are extracted from the communication packets by using the “tshark” software [28]. Incoming and outgoing traffic are analyzed independently of each other and similarly to the evaluation of the services, they are evaluated on the basis of predefined protocol lists. With the scan results shown in Table V, the device would be scored zero points.

TABLE V. COMMUNICATION SCAN

source device	destination port	protocol
00:11:22:33:44:55	5060	sip

The necessity of a cloud connection could not be checked automatically at this point, as it is not possible to determine whether a device is still fully functional after a possible interruption of such a connection.

D. Default passwords

Checking a device for the use of insecure login credentials is done using the software “THC-Hydra” [29], which performs a dictionary attack against the corresponding device. Both, the user name and the password are checked against known and frequently used terms. The information about the type of service for which a login check should be performed is taken from the previous service scan.

The use of non-standard logon procedures may cause a problem with this type of password check. For each specific procedure, a separate check algorithm would have to be developed, which might have to be adapted again after an update of the device firmware. An example of a vendor-specific login procedure is the challenge-response mechanism that AVM uses for the web interface of its Fritz!Box routers [30].

E. Firmware

In the absence of a standardized procedure for identifying the device firmware, it was not possible to check it in an automated procedure. The use of Nmap [4] allows assumptions about the operating system and other software components used on a device. However, due to rough inaccuracies, these are not sufficient for a valid risk evaluation. In addition, Nmap is only able to identify an operating system if it has already been fingerprinted in the past [31].

While it would be possible to create a Nmap fingerprint for each network device, this method can be considered irrelevant in practice because of the need to know the software running on each device. There is also no guarantee that the identifiers will not change after a software update, which would require the fingerprint to be recreated. These concerns can be transferred to procedures developed independently of Nmap.

F. Overall rating

After all the categories have been evaluated, an overall rating for a device can be calculated by averaging the ratings. This rating describes the vulnerability of a device based on Table VI. A ports score of 0.66 points, a communication score of 0.00 points, and a password score of 0.00 points results in an overall rating of 0.22 points, indicating that the device is highly vulnerable.

The security of the devices is indicated with an average evaluation instead of the lowest individual evaluation, since an overall impression of the device security should be given.

TABLE VI. VULNERABILITY CATEGORIES

score	category
0.00 to 0.80	high vulnerability
0.81 to 1.80	moderate vulnerability
1.81 to 2.00	small vulnerability

VI. RESULTS

For the verification and validation of the presented evaluation system, the following devices were tested as examples: Amazon Echo 2 (1), Apple iPhone 5 (2), Floureon M32B (3), Google Home mini (4), Renkforce RenkCast (5), Sonoff Wi-Fi Smart Switch (6), Wi-Fi Smart Bulb (7), Wi-Fi Smart Plug (8) and Wi-Fi Touch Switch (9). The results of the automatic evaluation related to these example devices can be found in Table VII. The devices were then checked manually and the resulting evaluation was compared with the automatically determined values. The values for “cloud only” and “default password”, shown in Table VII, where added manually. All the values are calculated as described in Section V.

In a later step, weightings can be assigned to the previously mentioned categories in order to make clear their different influences on device security.

The overall network rating is determined by the value of the least secure device.

TABLE VII. EXAMPLE RESULTS

parameter	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Wi-Fi encryption	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
services	1.00	2.00	0.33	1.00	2.00	2.00	1.00	2.00	2.00
LAN communication	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
WAN communication	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00	2.00
wired connection	1.00	1.00	2.00	1.00	1.00	1.00	1.00	1.00	1.00
cloud only	1.00	1.00	2.00	1.00	2.00	1.00	1.00	1.00	1.00
default password	2.00	2.00	1.00	2.00	2.00	2.00	2.00	2.00	2.00
overall score	**1.57**	**1.71**	**1.62**	**1.57**	**1.86**	**1.71**	**1.57**	**1.71**	**1.71**

While the calculation of the device ratings can be fully automated, it is not possible, as explained in Section III, to collect the necessary information automatically. For this reason, in the following chapters we present a new approach to retrieve device-specific information from the devices themselves.

VII. IoTAG Definition

The *IoT* Device Identific*A*tion and Reco*G*nition, short *IoTAG*, should be implemented into IoT devices by the manufacturers. It provides static and dynamic information about the device and its current state.

The focus of the IoTAG definition is on the standardized provision of critical device data, the integrity maintenance of the data sets to be transmitted and the relevance of the information for an individual classification of each device with regard to the implementation of security specifications and recommendations [32].

The core component of the IoTAG definition is the dataset specification. In the first instance, this is a list of attributes whose content is described. In the course of the technical specifications for the serialization of data for transmission over the network, data types and formatting specifications are assigned to these attributes.

In addition to a unique identifier for identifying individual devices, the dataset contains general product information such as a serial number, the device type according to fixed type definitions, a device category related to the main use scenario of the device, a product name and the manufacturer. Information about the installed hardware, such as the presence of a secure element or the use of a secure boot procedure, is also taken into account.

With regard to the connectivity of a device, information on the availability of LAN, WLAN or Bluetooth connections and their version is provided.

In addition to these static values, IoTAG also includes dynamic information about the update behavior of the device (automatic updates, end of support, etc.) as well as the device firmware (e.g., the current version). Furthermore, an overview of communication services such as SSH or HTTP servers, the associated software and the cryptographic algorithms that are used is provided.

In order to be able to associate the origin of the transmitted data with a device, a signature procedure will be presented, which is intended to ensure the integrity of this dataset. The signature is applied to the data serialized for communication as described later.

A. Dataset

The IoTAG Dataset consists of thirteen information about the IoT device:

1) Manufacturer
2) Name
3) Serial number
4) Type
5) ID
6) Category
7) Secure boot
8) Firmware
9) Client software
10) Updates
11) Cryptography
12) Connectivity
13) Services

1) Manufacturer: The manufacturer information is important to identify the device correctly and in case of a security issue, to contact the right company. The value of this information is a string that contains the company name as it is officially registered. This allows a clear assignment of the company, which is responsible for the device.

2) Name: The name is also a string, which contains the name of the device. It should be named as it is listed by the manufacturer with all the additional revision numbers like “Test Cam rev 3A”, to ensure an exactly identification in the case of security issues. Sometimes, not the complete batch of products is affected, because there could be a software update in later devices. This difference should be identifiable.

3) Serial Number: The next item, the serial number should be assigned by the manufacturer as a unique identification. It can be necessary for a network administrator to know all the serial numbers of his devices, if some of them are broken and need support or, if a security issue concerns some devices with a specific production date (which the manufacturer can identify by the serial number).

4) Type: To determine the potential damage of an attack, the device type is necessary. For example, a smart speaker cannot harm people directly. But if an attacker deactivates the smoke detector, it is a safety issue. The different types can help to estimate the damage and therefore, to separate the devices. We give some first suggestions for the type, but this list needs to be extended for all the different kind of IoT devices.

Suggestions for device types:

- Alarm system
- Camera
- Smart lock
- Smart speaker
- Smart TV

- Smoke detector
- Production machine
- Temperature sensor
- Security camera
- Emergency switch
- ...

5) ID: Besides the serial number, which is on required to be unique for one product, the identifier (ID) should be unique for every device worldwide. To achieve this requirement, the ID is created by concatenating the manufacturer name, the product name and the serial number. This string is hashed, using the SHA-256 algorithm [33] and encoded as base16 string [34], to ensure the right format. As for the use in IoT devices, the faster algorithm SHA2, prior to SHA3, is used [35] [36] [37]. The composition of the ID is shown below:

```
ID = BASE16 ( SHA-256 ( MANUFACTURER & PRODUCT NAME &
     SERIAL NUMBER ) )
```

6) Category: Similar to the type, the product category should help to determine the risk of an attack. But the category is not as accurate as the product type, because it should be used to categorise the different kind of products. Additionally, this can be used to separate the networks for the different device categories. Some examples are given in the following list:

- Assisted living
- Entertainment
- Household
- Industry
- Infrastructure
- Lighting
- Personal assistance
- Security
- ...

7) Secure Boot: The boolean value (true or false) for the secure boot indicates, if the device has a secure boot mechanism and therefore can ensure the integrity of its firmware at system startup.

8) Firmware: The firmware version is needed to check if there are new updates available. Additionally, an internet address must be given to download the newest version of a devices firmware. This is important, if the automatic update process is not working. Technically, the firmware is not one value, but two separate strings: the firmware version and a Uniform Resource Locator (URL) [38] to get the firmware. The version should consist of lexicographically ascending terms (higher number or character).

9) ClientSoftware: The client software is structured exactly like the firmware: version number and download URL. If a device does not use software for third-party devices, an empty string is returned.

10) Update: The update consists of multiple values. First, if the device updates itself automatically. This includes the whole process: check for new version, download and installation. It is an boolean value and named "Automatic updates".

The next value indicates, if the automatic update process is technically possible. If a connection to the update server can be established and the check and installation of updates is possible, it is set to "true". This value is also an boolean and named "Automatic updates possible".

The third value contains the date of the last update ("Last update on") and the fourth value the date of the end of support ("end of life") according to ISO 8601 [39].

11) Cryptography: To be able to make statements about the cryptographic capabilities of a device, it is necessary to know the algorithms used by the device. In addition, it must also be possible to make a statement as to whether these are implemented in hardware or software. It should also be specified whether secret keys are managed exclusively in secure hardware or in the main memory of the device.

The private key required for the signature of IoTAG as described in subsection C, is treated as a separate variable, as it is essential for the reliability of IoTAG.

Under the generic term cryptography, two identical structures are classified. Each of these subsections contains an attribute "IoTAG key", which is represented as a boolean value. If the key used for signature is managed exclusively in a secure hardware environment, the value "true" is assigned in the hardware structure and the value "false" in the software structure. If the key is accessible via software, the values are reversed.

Whether secret cryptographic keys are stored in any of the above-mentioned areas, is indicated by the boolean variable "key store". This variable can have the value "true" in both structures. The variable "algorithms" gives an overview of the cryptographic algorithms used in a device. This is a collection of character strings, which in turn represent a cryptographic algorithm according to its standardized name (example: "ecdsa-sha2-nistp256", defined in RFC 5656 [40]).

12) Connectivity: The physical possibilities of a device to connect to other devices, are subsumed under the term "connectivity". In the case of IoT devices, the connection is achieved using several different communication standards, like Ethernet (IEEE 802.3) and Wi-Fi (IEEE 802.11) [41], which are developed by the Institute of Electrical and Electronics Engineers (IEEE). Additionally industrial standards, like Bluetooth [42] and ZigBee [43], are also common in IoT environments. As connectivity standards evolve over time, which often includes security improvements, they are versioned. To improve user experience and compatibility, older versions are often still supported by the devices. This can decrease security, as older versions are more likely to contain security issues [44] [45].

IoTAG utilises a multi-part data structure to list the supported communication standards. The attributes of this structure are named like the standards, e.g., "IEEE802_11", "Bluetooth" and "ZigBee". Each attribute contains an collection of strings, which contain the supported versions and some times the encryption used, for example, in the case of IEEE

802.11. The version string can only contain alphanumeric versions. Bluetooth and ZigBee, for example, have numeric versions and Wi-Fi versions are named after their IEEE 802.11 suffix. As mentioned earlier, the collection can contain the supported encryption standards like “WEP”, “WPA”, “WPA2” and “WPS”.

13) Services: Services describe the ways other clients can communicate with the device. This communication also needs to be encrypted. Additionally, the software running on the device, which exposes the server to the network could have security flaws, hence the currently running version should also be included.

The services of a device are listed under a separate data structure as a collection of several services, which contain the following attributes: The name of the service, the port that is utilised by it, the protocol used for communication and the name and version of the software. The name and version of the software are combined into one string in the format <designation>-<version>. The port is a string which is the combination of the port and either TCP or UDP and is separated by a slash: <Port>/<UDP|TCP>.

B. Serialization

Now, as the data contained in IoTAG has been defined, there is a need for a uniform format to transport and process this data. The goal is to avoid incompatibilities due to misinterpretations.

For serialization, the Javascript Object Notation (JSON), according to the specification in ECMA-404 [46] and RFC 8259 [47] with UTF-8 encoding is chosen.

Because of its lower memory consumption and better computing performance, JSON is preferred to the Extensible Markup Language (XML) [48].

Below is a fully serialized IoTAG data set whose attribute names have been transferred into a uniform format. The value of the attribute ’ID’ had to be wrapped into several lines to be displayed completely.

```
{
 "Manufacturer": "Beispiel GmbH",
 "Name": "Example-Device",
 "SerialNumber": "D1.0",
 "Type": "example device",
 "ID": "2071c7736acd16f6cea3727d
        3b7ecde53f4c2e97b421f355
        0248e19d7309c636",
 "Category": "infrastructure",
 "SecureBoot": false,
 "Firmware": {
    "Version": "1.0",
    "URL": "https://192.168.102.94:10000/FirmwareInfo"
 },
 "ClientSoftware": {
    "Version": "",
    "URL": ""
 },
 "Updates": {
    "AutomaticUpdates": false,
    "AutomaticUpdatesPossible": false,
    "LastUpdateOn": "2020-08-01T00:00:00"
    "EndOfLife": "2021-01-01T00:00:00"
 },
 "Cryptography": {
    "Software": {
        "IoTAGKey": true,
        "KeyStore": true,
        "Algorithms": [
            "RSASSA-PSS",
            "SHA-256",
            "TLS_AES_128_GCM_SHA256",
            "TLS_AES_256_GCM_SHA384",
            "TLS_CHACHA20_POLY1305_SHA256",
            "aes128-ctr",
            "aes192-ctr",
            "aes256-ctr",
            "ecdsa-sha2-nistp256",
            "ecdsa-sha2-nistp384",
            "ecdsa-sha2-nistp521",
            "ssh-rsa",
            "ssh-dss",
            "ecdh-sha2-nistp256",
            "ecdh-sha2-nistp384",
            "ecdh-sha2-nistp521",
            "diffie-hellman-group-exchange-sha256",
            "hmac-sha2-256,hmac-sha2-512"
        ]
    },
    "Hardware": {
        "IoTAGKey": false,
        "KeyStore": false,
        "Algorithms": []
    }
 },
 "Connectivity": {
    "IEEE802_11": [
        "WPA2",
        "b",
        "g",
        "n",
        "ac"
    ],
    "Bluetooth": [
        "4.2"
    ],
    "ZigBee": []
 },
 "Services": [
    {
        "Name": "IoTAG",
        "Port": "27795/TCP",
        "Protocol": "HTTP/2",
        "Software": "IoTAG-Server-1.0"
    },
    {
        "Name": "SSH",
        "Port": "22/TCP",
        "Protocol": "SSH-2",
        "Software": "OpenSSH-8.1"
    }
 ]
}
```

C. Integrity

The definitions presented so far do not yet include a procedure for verifying the provided data. It is not possible to verify whether a received data set was actually provided by the device it describes. In the course of this chapter, the signature mechanism for IoTAG will be introduced. First, the signature procedure is presented, then, the generation of the data to be signed is explained and finally the complete signing process and the subsequent validation of the signature based on examples is illustrated.

1) Signature algorithm and authentication: The RSA procedure serves as the basis for the signature mechanism of IoTAG. This is an asymmetric encryption method in which a message is encrypted with the recipient’s public key, whereby the plaintext can only be restored with the corresponding private key. By reversing this procedure and encrypting a message with the sender’s private key, the source text can be calculated using its public key. This ensures that the message is only created and sent by an instance that has the private key [49]. The keys themselves are random bit sequences for which a minimum length of 2048 bits is recommended [50].

Since the RSA algorithm would always generate the same encryption text for identical messages, methods were developed that combine the plaintext with a random value, the padding, before each encryption process. The Public-Key Cryptography Standards (PKCS) define two signing procedures for RSA in PKCS #1 that take such padding into account. These procedures, called signature schemes with appendix (SSA), are RSASSA-PKCS1-v1_5 and RSASSA-PSS. The latter is preferred for new developments, which is why it is used for IoTAG signatures using the standard options defined in PKCS #1 [51].

To verify the signature, the message recipient must know the sender's public key. However, this must also ensure that an attacker has not published his key to the recipient and is therefore able to generate misleading messages whose signature is considered valid by the recipient. To counteract this, the signer's public key is published in conjunction with a certificate, which in turn is signed by a trustworthy third party and provides certainty about the origin of the verification key [49]. In IoTAG certificates are used according to the specification in ITU-T X.509 [52] and RFC 2459 [53], as they are also used in the Transport Layer Security (TLS) protocol [54].

Such a certificate can be issued directly by the manufacturer of a device and stored on the device, or it can be created when the device is set up and then signed by a local or external certification authority. In all cases it must be ensured that each device receives an individual certificate. It is the responsibility of the message recipient to check the validity of the certificate.

2) Signed dataset: After a suitable signature procedure has been selected, it is now necessary to determine which data is to be signed. Basically, the target of the signature is always the IoTAG data record in serialized form and thus a UTF-8 encoded character string. However, not this entire string is used for the signature, but instead a hash sum is calculated from it, which is then signed. The SHA-256 algorithm is used to generate this sum, as recommended by NIST (National Institute of Standards and Technology) [35].

Before the hash algorithm can be applied, the IoTAG string is converted into a byte array. Only from this array, the hash sum is calculated, to which the signature algorithm is then applied. If the array contains a terminating null byte, this is ignored in the hash calculation.

3) The signing process based on examples: This example is intended to illustrate the following sequences of the signing process: The creation of the hash sum, the signing of the hash sum and the validation of the signature. To do this, an RSA key pair and an IoTAG must first be defined. A size of 2048 bits is chosen for the key pair. For reasons of clarity, the IoTAG is not serialised in its entirety, but only using the fields "Manufacturer", "Name", "SerialNumber" and "ID". Also, no certificates, but only the required keys are used. The implementation of the program code required for the example is done in the programming language Go [55].

Before the actual signing process can be started, an RSA key pair with a size of 2048 bits and an IoTAG object with exemplary attribute values must be created:

```
privkey, _ := rsa.GenerateKey(rand.Reader, 2048)
pubkey := privkey.Public().(*rsa.PublicKey)

iotag := struct {
    Manufacturer string
    Name         string
    SerialNumber string
    ID           string
}{
    "Example Company",
    "smoke detector",
    "R1.234",
    "db0fb9870ffc08ccc" +
    "b59b9d65a0ceb0cd0" +
    "108265471a89e3c35" +
    "e21edfe7c00d3",
}
```

The IoTAG object can now be converted into a JSON object:

```
serialized, _ := json.Marshal(iotag)
```

In case of Go, the serialization process returns a non zero terminated byte array, which can be directly used for the calculation of the hash sum. The byte chain, generated by the serialization, can now be transferred to the hash algorithm:

```
hashed := sha256.Sum256(serialized)
```

By which the following Hashsumme in hexadecimal representation results:

```
f278178e0a885a074f7bf8e06968f11b
53931a00108dd46eb4b1a238dd312959
```

This can now be used to create the signature using the RSASSA-PSS procedure, which additionally requires the private RSA key:

```
signature, _ := rsa.SignPSS(rand.Reader, privkey,
    crypto.SHA256, hashed[:], nil)
```

The signature is now ready to be transmitted.

To be able to check whether the signature generated in the previous step is valid, the receiver needs the following additional information:

- The serialized IoTAG object
- The public key

In this example, it is assumed that this information has already been transmitted to the verifier of the signature and the hash operation has been performed, so that the signature verification can be executed with the corresponding parameters:

```
result := rsa.VerifyPSS(pubkey, crypto.SHA256, hashed
    [:], signature, nil)
if result == nil {
    fmt.Println("Signature valid!")
} else {
    fmt.Println("Signature invalid!")
}
```

D. Communication

The last open point to be defined, is the IoTAG related communication behaviour. This includes the retrieval of IoTAG data from a device, as well as the retrieval of software resources via a URL, which must be provided by the device

firmware via IoTAG. The same technologies are used for both procedures, which is why a general description of the communication endpoint, the transmission protocol and the data format is given, before the two procedures are explained in more detail.

1) General description: HTTP Version 2 with Transport Layer Security (TLS) is selected as the transmission protocol [56] (Hypertext Transfer Protocol Secure, HTTPS for short). For querying information, an HTTPS-capable server application must be provided as the communication endpoint, which has a trustworthy certificate for encrypted communication. This application does not have to support the full scope of the operations defined in RFC 2616 [57], but only has to be able to respond to an individual GET request by providing the respective data record. The addressed resource is determined by the respective URL.

The JSON format is used to format the data for transmission within HTTP packets.

2) Retrieving Software Resources: It was determined that the IoTAG data set provided by a device should contain a URL to obtain the latest available device firmware and, if necessary, software for client systems. It is not possible to download the software directly via this URL. Instead, it is used to perform the HTTP request described before. The response to this request contains a JSON object, which in turn has the string attributes “URL” and “Version”. This URL can now be used directly to download the firmware. The second specification informs about the version of the software.

3) Retrieving IoTAG: Every IoTAG compatible device must provide a communication interface to retrieve the IoTAG dataset. In order to make this procedure uniform, a unique HTTP URL must be defined, which is used to access a corresponding resource. This requires a uniform port number and a predefined path for the request to the HTTP server. 27795 is specified as the network port. The path consists of a single segment called “iotag”. This results in the following URL scheme, where the “<host>” specification is to be interpreted according to the definition in RFC 3986 paragraph 3.2.2:

```
https://<host>:27795/iotag
```

The example created in the course of the description of the signature process shows that in addition to the actual IoTAG data record, additional information is required to verify its correctness. This is a certificate that contains the key needed to verify the signature, as well as the signature itself. A separate JSON object is also defined for this purpose, which contains this information in the form of the attributes “IoTAG”, “Certificates” and “Signature”.

Since, the signature is present as a byte sequence, it will first be encoded to base64, which allows it to be integrated into the JSON object as a string. The format in which the certificate is stored on the respective devices depends on the implementation by the manufacturer. It must therefore be converted into a uniform format for transmission. For the transmission of ITU-T X.509 certificates in non-binary form, the encoding according to RFC 7468 [58] is suitable. Basically, the certificate is first converted into a binary structure, taking into account the encoding rules specified in ITU-T X.690 [59], and then encoded to base64, which means it can also be embedded as a string in the JSON object.

If additional certificates are required to verify the certificate, all certificates are first encoded and the resulting character strings are then concatenated. The order according to the specification in RFC 5246 chapter 7.4.2 [54] must be taken into account.

The IoTAG data record could be entered directly as an object, since it is JSON-serialized for transmission anyway. In order to check the signature, the recipient must extract the IoTAG object from the parent object. This can be done in two ways: the recipient can continue to treat the transmitted data as a character string and try to extract the IoTAG object by manipulating it. However, this procedure is unusual and involves additional development effort, since the corresponding extraction routine must be implemented. Alternatively, the received JSON object can be deserialized to an object of the respective programming language used and then be processed further.

Although, the latter approach is preferable, it also makes signature verification more problematic. To perform this step, the IoTAG object must be serialized back to a string after extraction to calculate the hash sum. This serialization produces different results depending on the software used, which ultimately results in different hash values. The problem of the different serializations can be illustrated with an example. First, an object is created in the programming language Python [60], which is identical to the object before. This object is then serialized and hashed:

```
iotag = {
    "Manufacturer": "Beispiel GmbH",
    "Name": "Rauchmelder",
    "SerialNumber": "R1.234",
    "ID":  "db0fb9870ffc08cccb59b9d"
           "65a0ceb0cd0108265471a89"
           "e3c35e21edfe7c00d3"
}
serialized = json.dumps(iotag).encode('utf-8')
hash = hashlib.sha256(serialized).hexdigest()
print(hash)
```

This process results in the following hash value:

```
5063aec9e300b6d4a61ce3dd6f7b0b42
98ddc230914ca3b5676df694fbe632e7
```

By comparing this result with the one before, it can be seen that the values are different. A signature verification based on the respective hashes would thus fail, although the information would remain unchanged.

To counter this problem, the IoTAG data set must be transferred within a JSON object in such a way that it can be extracted by deserialization without affecting the formatting. This can be achieved by treating the serialized IoTAG data for transfer as a string rather than an object. However, all JSON control characters within this string must be replaced with appropriate escape sequences before transmission to allow for

error-free interpretation. These must also be removed by the receiver before the hash calculation.

In order to avoid the resulting programming effort, a further approach is preferred. The transmission of the IoTAG data as a string is retained, but the character string resulting from its serialization is first base64 encoded. The result of this process is then set as the value of the IoTAG attribute. This enables the recipient of the data to parse the received JSON object and decode the information inside. This information will then be available in the same format as it was processed by the sender.

VIII. Conclusion

The scoring of the network is in the early stages of research and continuously being developed. There are still some points which are unclear and the individual weightings have to be adjusted in detail. Nevertheless, we show an advanced approach which can already be tested in practice.

As mentioned in our previous publication [1], we want to improve our scoring system by scanning vulnerability databases to change the scoring and warn the user, if a new vulnerability emerges.

To get the best results and an accurate overview of the devices, we proposed our standard IoTAG. It solves the problems with device detection and provides reliable information about the current security status of the individual devices.

The definition of the individual points of IoTAG is already far-reaching, but can be flexibly extended by further parameters. This should keep open the possibility to add further features (e.g., the functions of a device) or, to merge with other existing approaches.

In this paper, we have additionally shown that it is possible to implement IoTAG in Go with little effort. The same is true for the C programming language, which attests to a broad application, especially in the field of IoT. So far there are still missing further implementations and public code repositories, which we plan to submit in the near future.

In addition to the advantages of IoTAG, the view of an attacker should be briefly considered. IoTAG enables the possibility for an attacker to get all the device data from a network without much effort. This can help to identify the most vulnerable device within the network. To avoid those attacks, devices can release the IoTAG data to client systems only if their TLS certificate is signed by a trusted certification authority. Another possibility would be to store the device that is authorized to retrieve IoTAG data in the configuration.

But there is still an unsolved problem. Manufacturers must integrate IoTAG into their devices to enable the comprehensive device detection and the associated network scoring.

References

[1] S. Fischer, K. Neubauer, L. Hinterberger, B. Weber, and R. Hackenberg, "IoTAG: An Open Standard for IoT Device IdentificAtion and RecoGnition," in SECURWARE 2019, Thirteenth International Conference on Emerging Security Information, Systems and Technologies, 2019, pp. 107-113.

[2] "Rash of in-the-wild attacks permanently destroys poorly secured IoT devices," 2017, URL: https://arstechnica.com/information-technology/2017/04/rash-of-in-the-wild-attacks-permanently-destroys-poorly-secured-iot-devices/ [accessed: 2020-08-18].

[3] "Five nightmarish attacks that show the risks of IoT security," 2017, URL: https://www.zdnet.com/article/5-nightmarish-attacks-that-show-the-risks-of-iot-security/ [accessed: 2020-08-18].

[4] "Nmap: the Network Mapper - Free Security Scanner," URL: https://nmap.org [accessed: 2020-08-18].

[5] "Fing - IoT device intelligence for the connected world," URL: https://www.fing.com [accessed: 2020-08-18].

[6] M. Miettinen et al., "IOT SENTINEL Demo: Automated Device-Type Identification for Security Enforcement in IoT," in IEEE 37th International Conference on Distributed Computing Systems (ICDCS), 2017, pp. 2511-2514.

[7] T. D. Nguyen et al., "DÏOT: A Federated Self-learning Anomaly Detection System for IoT," in IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 756-767.

[8] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," in IEEE Trans. Dependable Secure Comput., vol. 2, no. 2, April 2005, pp. 93–108.

[9] J. Cache, "Fingerprinting 802.11 implementations via statistical analysis of the duration field," Uninformed, org 5, 2006.

[10] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. Van Randwyk, and D. Sicker, "Passive data link layer 802.11 wireless device driver fingerprinting," in USENIX Security Symposium, USENIX, 2006, pp. 167-178.

[11] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in International Conference on Mobile Computing and Networking, ACM, 2008, pp. 116–127.

[12] A. E. Khaled, H. Abdelsalam, L. Wyatt, and L. Choonhwa, "IoT-DDL device description language for the T in IoT," in IEEE Access 6, 2018, pp. 24048-24063.

[13] S. Kaebisch and A. Darko, "Thing description as enabler of semantic interoperability on the Web of Things," in IoT Semantic Interoperability Workshop, 2016, pp. 1-3.

[14] "Web of Things (WoT) Thing Description," Apr. 2018, URL: https://www.w3.org/TR/wot-thing-description/ [accessed: 2020-08-18].

[15] "andypitcher/IoT_Sentinel: IoT SENTINEL : Automated Device-Type Identification for Security Enforcement in IoT," December 9, 2018, URL: https://github.com/andypitcher/IoT_Sentinel [accessed: 2020-08-18].

[16] F. Loiy, A. Sivanathany, H. H. Gharakheiliy, A. Radford, and V. Sivaraman, "Systematically Evaluating Security and Privacy for Consumer IoT Devices," in IoT S&P 2017, 2017, pp. 1-6.

[17] K. C. Park and D. Shin, "Security assessment framework for IoT service," in Telecommun Syst, 2017, pp. 193–209.

[18] B. Ali and A. I. Awad, "Cyber and Physical Security Vulnerability Assessment for IoT-Based Smart Homes," in sensors journal, vol 18(3), 2018, pp. 817.

[19] R. I. Bonilla, J. Crow, L. Basantes, and L. Cruz, "A Metric for Measuring IoT Devices Security Levels," in IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 2017, pp. 704-709.

[20] "Norton Core Router — Secure WiFi Router," URL: https://us.norton.com/core [accessed: 2020-08-18].

[21] "Avira SafeThings™ - IoT Security for the Connected Home," URL: https://safethings.avira.com/for-partners [accessed: 2020-08-18].

[22] "Norton Core Security Score," Mar. 30, 2020, URL: https://support.norton.com/sp/en/za/norton-core-security/current/solutions/v118380521 [accessed: 2020-08-18].

[23] "How do I purchase Norton Core?," Apr. 23, 2020, URL: https://support.norton.com/sp/en/us/norton-core-security/current/solutions/v131932667 [accessed: 2020-08-18].

[24] "Service Name and Transport Protocol Port Number Registry," Aug. 12, 2020, URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt [accessed: 2020-08-18].

[25] "Cryptography in IT - recommendations on encryption and procedures, Kryptographie in der IT - Empfehlungen zu Verschluesselung und Verfahren," June 17, 2016, URL: https://www.heise.de/security/artikel/Kryptographie-in-der-IT-Empfehlungen-zu-Verschluesselung-und-Verfahren-3221002.html [accessed: 2020-08-18].

[26] "CVE - Common Vulnerabilities and Exposures (CVE)," URL: https://cve.mitre.org/ [accessed: 2020-08-18].

[27] "Dienst- und Versionserkennung," URL: https://nmap.org/man/de/man-version-detection.html [accessed: 2020-08-18].
[28] "tshark - Dump and analyze network traffic," URL: https://www.wireshark.org/docs/man-pages/tshark.html [accessed: 2020-08-18].
[29] "GitHub - vanhauser-thc/thc-hydra: hydra," URL: https://github.com/vanhauser-thc/thc-hydra [accessed: 2020-08-18].
[30] "Login to the FRITZ!Box Web Interface," 2018, URL: https://avm.de/fileadmin/user_upload/Global/Service/Schnittstellen /Session-ID_english_13Nov18.pdf [accessed: 2020-08-18].
[31] "OS Detection - Nmap Network Scanning," URL: https://nmap.org/book/man-os-detection.html [accessed: 2020-08-18].
[32] L. Hinterberger, S. Fischer, B. Weber, K. Neubauer, and R. Hackenberg, "IoT Device IdentificAtion and RecoGnition (IoTAG)," in CLOUD COMPUTING 2020, The Eleventh International Conference on Cloud Computing, GRIDs, and Virtualization, Accepted, 2020.
[33] U.S. Department of Commerce and National Institute of Standards and Technology, "Secure Hash Standard (SHS)," 2015.
[34] "RFC 4648 - The Base16, Base32, and Base64 Data Encodings," Oct. 2006, URL: https://tools.ietf.org/html/rfc4648 [accessed: 2020-08-18].
[35] "NIST Policy on Hash Functions," June 22, 2020, URL: https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions [accessed: 2020-08-18].
[36] R. K. Dahal, J. Bhatta, and T. N. Dhamala, "Performance Analysis of SHA-2 and SHA-3 finalists," in International Journal on Cryptography and Information Security (IJCIS), Sept. 2013, pp. 720-730.
[37] U.S. Department of Commerce and National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," 2015.
[38] "RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax," Jan. 2005, URL: https://tools.ietf.org/html/rfc3986 [accessed: 2020-08-18].
[39] International Organization for Standardization, "Data elements and interchange formats — Information interchange — Representation of dates and times," 2004.
[40] "RFC 5656 - Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer," Dec. 2009, URL: https://tools.ietf.org/html/rfc5656 [accessed: 2020-08-18].
[41] "Institute of Electrical and Electronics Engineers," URL: https://ieeexplore.ieee.org/browse/standards/get-program/page/series?id=68 [accessed: 2020-08-18].
[42] Bluetooth SIG, Inc., "Bluetooth Core Specification, Revision 5.2," 2019.
[43] ZigBee Alliance, "ZigBee Specification," 2015.
[44] P. Kraft and A. Weyert, "Network Hacking," Franzis Verlag GmbH, 2015, pp. 345-360.
[45] J. Erickson, "Hacking," dpunkt.verlag GmbH, 2009, pp. 472-488.
[46] ECMA International, "The JSON Data Interchange Syntax," 2017.
[47] "RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format," Dec. 2017, URL: https://tools.ietf.org/html/rfc8259 [accessed: 2020-08-18].
[48] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML data interchange formats: A case study," in International Conference on Computer Applications in Industry and Engineering, CAINE, 2009, pp. 157-162.
[49] A. S. Tanenbaum, "Moderne Betriebssysteme," Pearson Deutschland GmbH, 2009, pp. 717-721.
[50] U.S. Department of Commerce and National Institute of Standards and Technology, "Recommendation for Key Management," 2015.
[51] "RFC 8017 - PKCS #1: RSA Cryptography Specifications Version 2.2," Nov. 2016, URL: https://tools.ietf.org/html/rfc8017 [accessed: 2020-08-18].
[52] International Telecommunication Union, "Recommendation ITU-T X.509," 2016.
[53] "RFC 2459 - Internet X.509 Public Key Infrastructure Certificate and CRL Profile," Jan. 1999, URL: https://tools.ietf.org/html/rfc2459 [accessed: 2020-08-18].
[54] "RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2," Aug. 2008, URL: https://tools.ietf.org/html/rfc5246 [accessed: 2020-08-18].
[55] "The Go Programming Language," URL: https://golang.org/ [accessed: 2020-08-18].
[56] "RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)," May 2015, URL: https://tools.ietf.org/html/rfc7540 [accessed: 2020-08-18].
[57] "RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1," June 1999, URL: https://tools.ietf.org/html/rfc2616 [accessed: 2020-08-18].
[58] "RFC 7468 - Textual Encodings of PKIX, PKCS, and CMS Structures," Apr. 2015, URL: https://tools.ietf.org/html/rfc7468 [accessed: 2020-08-18].
[59] International Telecommunication Union, "Recommendation ITU-T X.690," 2015.
[60] "Welcome to Python.org," URL: https://www.python.org/ [accessed: 2020-08-18].