energies

Article

On the Design of IoT Security: Analysis of Software Vulnerabilities for Smart Grids

Christos-Minas Mathas [1], Costas Vassilakis [1,*], Nicholas Kolokotronis [1], Charilaos C. Zarakovitis [2,3] and Michail-Alexandros Kourtis [2]

[1] Department of Informatics and Telecommunications, University of the Peloponnese, 22100 Tripolis, Greece; mathas.ch.m@uop.gr (C.-M.M.); nkolok@uop.gr (N.K.)
[2] Institute of Informatics and Telecommunications, National Centre for Scientific Research Demokritos, 15341 Athens, Greece; c.zarakovitis@iit.demokritos.gr (C.C.Z.); akis.kourtis@iit.demokritos.gr (M.-A.K.)
[3] Innovation Department, Axon Logic P. C., 14231 Athens, Greece; c.zarakovitis@axonlogic.gr
* Correspondence: costas@uop.gr; Tel.: +30-2710372203

Abstract: The 5G communication network will underpin a vast number of new and emerging services, paving the way for unprecedented performance and capabilities in mobile networks. In this setting, the Internet of Things (IoT) will proliferate, and IoT devices will be included in many 5G application contexts, including the Smart Grid. Even though 5G technology has been designed by taking security into account, design provisions may be undermined by software-rooted vulnerabilities in IoT devices that allow threat actors to compromise the devices, demote confidentiality, integrity and availability, and even pose risks for the operation of the power grid critical infrastructures. In this paper, we assess the current state of the vulnerabilities in IoT software utilized in smart grid applications from a source code point of view. To that end, we identified and analyzed open-source software that is used in the power grid and the IoT domain that varies in characteristics and functionality, ranging from operating systems to communication protocols, allowing us to obtain a more complete view of the vulnerability landscape. The results of this study can be used in the domain of software development, to enhance the security of produced software, as well as in the domain of automated software testing, targeting improvements to vulnerability detection mechanisms, especially with a focus on the reduction of false positives.

Keywords: Internet of Things; software vulnerabilities; security; robustness

check for updates

Citation: Mathas, C.-M.; Vassilakis, C.; Kolokotronis, N.; Zarakovitis, C.C.; Kourtis, M.-A. On the Design of IoT Security: Analysis of Software Vulnerabilities for Smart Grids. *Energies* **2021**, *14*, 2818. https://doi.org/10.3390/en14102818

Academic Editors: Luis Hernández-Callejo and George Halkos

Received: 17 March 2021
Accepted: 12 May 2021
Published: 14 May 2021

1. Introduction

The 5G communication network will enable new applications to emerge in many societal and economic functions such as energy, transport, and health by providing improved and new features over its predecessors with respect to capacity, delay, service development, energy consumption, connectivity, and more. The next generation network will support an increasingly diverse set of new and emerging services, paving the way for unprecedented performance and capabilities in mobile networks. It introduces a new core network architecture with the main feature being network slicing which will enable the offering of dedicated network slices enabling services per need. This allows for flexible, smart and scalable adaptation of the network resources available at a given time to meet the requirements of the services supported [1]. All these characteristics will enable further proliferation of the IoT, introducing the "5G-enabled IoT" era, which will support the deployment of a massive number of IoT devices towards covering the ever-increasing demand for wireless services, delivering advanced and secure applications and services—including blockchain-supported services (e.g., [2–4])—and stimulating economic and social growth [5].

The Internet of Things (IoT) refers to Internet-enabled devices that can communicate, sense, and make changes in their environment. They comprise an extensively diverse range of consumer, enterprise, and industrial products like smart meters, smart watches, smart TVs, connected cars, inventory optimization sensors, and face recognition cameras. Various predictions have been published about the rate of the IoT proliferation over the next years. According to a survey conducted by IHS Markit, the number of connected IoT devices will increase by 12% on average annually starting from 27 billion in 2017 to reach 125 billion in 2030 [6]. Another survey conducted by Statista, estimates the connected IoT devices will reach 75 billion in 2025 [7].

The 5G technology along with the ever-increasing IoT will greatly benefit innovation in the power sector, through its utilization in smart grids. The coupling of 5G with smart grids will pave the way for efficient power consumption, management, and monitoring, along with automation and intelligent control [8]. This will result in a transformation of today's power network bringing radical changes in the production, transmission, and consumption of electricity while supporting bidirectional flow of data and power between power providers and end-consumers. Specifically, smart grids will connect the whole supply chain of the industry; from the existing legacy control and communications infrastructure, to the generation and transmission systems, to the distribution network and the end-consumers' premises [9]. The information transmitted in smart grids is generated by IoT devices called smart meters, and this information is utilized to supply the aforementioned applications and benefits. The smart grid communication channels also accommodate the exchange of control information between smart meters, meter data management systems, and other systems that partly comprise the power grid.

Even though 5G technology has been designed by taking security into account, the underlying cyber security threat landscape is evolving rapidly. The advent of the IoT has not only brought benefits; it has also brought serious security and privacy considerations. Numerous publications have provided a comprehensive mapping of the threat landscape in IoT, smart grid and 5G systems and the identification of relevant mitigation strategies, including [10–17]. More specifically, reference [10] portrays the threat landscape of next generation IoT-enabled smart grids, while reference [15] examines the security measures that are deemed appropriate for smart grids. The work in [16] presents dimensionality reduction-based approaches for detecting and classifying internet-scale probing attempts in the IoT, while reference [17] discusses artificial intelligence (AI)-based techniques for performing intrusion detection in critical infrastructures. Reference [11] focuses on the hardware trojans targeting IoT devices. Reference [13] provides an extensive analysis of 5G security, while reference [14] reviews the 5G specification and identifies aspects of user privacy and security that are not adequately covered

Considering the above, we can conclude that the main body of knowledge in the field focuses on policy, procedural, protocol design, architectural and contextual (including human) factors. It is important, however, to note that the root causes of many major security incidents that have been reported, including the Mirai botnet [18] and Stuxnet [19], and that can be traced to software faults. The importance of software faults is very high, considering that the very essence of the IoT is that they are accessible publicly through the Internet and, therefore, architectural defenses cannot offer full protection by completely isolating devices with vulnerable software. As smart grids infrastructure includes IoT devices (e.g., smart meters) that perform critical operations and handle sensitive information they are attractive targets for adversaries as well. The current literature reports numerous types of attack against smart grids. Attackers may attempt to capture or alter the data flowing through a smart grid network, thus compromising the confidentiality or integrity of the system respectively. Most importantly, attackers may attempt to disrupt the operation of the system, thus compromising its availability [12].

The need for decreased costs to create market demand, along with huge competition, has led to a situation where the security aspect of device software and firmware does not receive enough attention, and consequently products reaching the market suffer from various of vulnerabilities, allowing threat actors to compromise them with little or no effort at all. Indicatively, reference [20] reports that out of 13 small office-home office (SOHO) router and network attached storage (NAS) devices tested, all of them had vulnerabilities resulting in 125 common vulnerability exposures (CVEs), while the response of some of some of the manufacturers to the vulnerability reports filed to them was from non-existent to suboptimal.

Most vulnerabilities can be traced to insecure coding practices by the software developers [21]. While much work has been published on IoT vulnerability analysis, most of it employs black-box approaches like penetration testing and fuzzing [22–26]. Little work has been published focusing on white-box testing of open-source software running on IoT devices, by examining the underlying source code and the vulnerabilities stemming from it. Even less work has been published for software vulnerabilities in the context of smart grids. Vulnerability research in the IoT domain can greatly benefit from source code reviews, since they usually discover different vulnerabilities than those discovered through black-box approaches. Furthermore, vulnerabilities discovered in open-source software can be documented in detail providing useful information for avoiding the recurrence of the same coding errors.

Alnaeli et al. [27] evaluate three IoT software systems written in C/C++, namely: openWSN (https://openwsn.atlassian.net/wiki/spaces/OW/overview; accessed on 13 May 2021), Contiki (http://www.contiki-os.org/), and TinyOS (http://www.tinyos.net/). The evaluation is based on the usage of unsafe functions, i.e., functions where the developer is responsible to ensure that the buffer provided for the storage of the result is large enough; if not, a buffer overflow may occur. In the recent years, for each unsafe function, a safe counterpart has been introduced, where the maximum buffer size that can be used by the function is supplied as an additional parameter. The evaluation in [27] is broken down into three "research questions": (1) "What is the number of unsafe functions used per system?", (2) "Which unsafe functions are the most prevalent?", and (3) "Over the history of a system, is the presence of unsafe functions increasing or decreasing?". The detection of insecure functions was performed using the *UnsafeFunsDetector* tool developed by one of the main authors. The results of the work indicate that Contiki has the largest occurrence of insecure functions, while *memcpy* is the most-used insecure function between the three evaluated systems followed by *strlen*, *strcmp*, and *snprintf*. The evaluation for the third question was performed on 5 different versions of each software system, released over the course of 5 years, with the results showing a fairly flat to increasing trend in the usage of insecure functions between versions. The authors extended their work in [28] to include an additional 15 IoT software systems written in C/C++. The results of this larger dataset showed uOS-Embedded (https://github.com/sergev/uos-embedded/wiki/about; accessed on 13 May 2021) to have the largest number of unsafe functions calls, while *memcpy* remained the most used insecure function. The 5-year trend was examined for only 4 of the newly added systems where three of them showed a flat trend and one a decreasing trend.

Ferrara et al. [29] discuss the Open Web Application Security Project (OWASP) IoT Top 10 [30] list and how effective static analysis would be in preventing these vulnerability categories. The static analysis tool used as reference is *Julia (*https://www.juliasoft.com/, https://news.grammatech.com/grammatech-acquires-juliasoft?hs_preview=CMJoYrRi-32593222452; accessed on May 13 2021). Taking into account that the OWASP Top 10 list [31] is used as a reference for evaluating Static application security testing (SAST), the authors use this information to deduce the effectiveness of SAST on the IoT-focused list of most prevalent vulnerabilities. In other words, the judgment of the effectiveness of *Julia's* on the OWASP IoT Top 10 [30] is performed on a theoretical level at this point. Additionally, the authors present an extension of *Julia's* injection checker that aims to address the novel challenges arising from IoT systems. The extension includes two main functionalities. The first one utilizes *Julia's* taint analysis engine to detect possible sensitive information leakage by following the propagation of data supplied to sensors that provide a public application programming interface (API). The second one allows a user to specify sources and sinks through an Excel spreadsheet which contains all the API calls that handle sensitive data. Julia's taint analysis engine returns a report of all the possible data flow graphs with possible leakages. The authors test the extended version of *Julia* on five projects based on Android Things which were chosen to be designed so that an edge program and an Android application communicate through some channels. The experimental results showed that issues related to the IoT-specific vulnerability categories were present in 4 out of 5 systems and were detected successfully.

Medeiros et al. [32] evaluate the source code of energy metering software which provide a web interface for smart meters. The source code is evaluated from a security perspective using the WAP (http://awap.sourceforge.net/#acks; accessed on 13 May 2021) (Web Application Protection) tool, which performs static analysis on PHP code and was developed by the main author. Additionally, the tool inserts fixes that correct the identified flaws and produces a report that educates the developers on how to avoid similar mistakes. The energy metering software evaluated are *emoncms* (http://openenergymonitor.org/) and *measureit* (https://code.google.com/p/measureit/; accessed on 13 May 2021). The analysis reported three SQL injection and 14 XSS (Cross-site scripting) vulnerabilities, some of which are discussed in detail. From the 17 reported vulnerabilities, 14 were confirmed through their successful exploitation, while 3 of them were not successfully exploited, meaning they were either false positives or the writers did not achieve to implement the appropriate exploit.

Sachidananda et al. [33] propose a framework for the collection, analysis, and generation of static analysis reports for IoT open-source software. The writers have gathered a large number of IoT firmwares, Android packages (APKs), operating systems, etc. using a web crawler that they designed. Furthermore, they have designed a tool for statically analyzing IoT firmware binaries called FirmScanner. The risk rating of identified vulnerabilities is performed through a customized version of OWASP's Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology; accessed on 13 May 2021). The part of this work that is more pertinent to ours is the utilization of static analysis tools to scan 50 IoT open-source software systems, ranging from operating systems to network stacks. As each static analysis tool works differently, the writers take this into account by using each one to detect different types of issues. The tools used are Visual Code Grepper (VCG) (https://sourceforge.net/projects/visualcodegrepp/; accessed on 13 May 2021), CPPCheck (http://cppcheck.sourceforge.net/), Yasca (https://github.com/scovetta/yasca; accessed on 13 May 2021), FindBugs (http://findbugs.sourceforge.net/), FindSecBugs (https://find-sec-bugs.github.io/), and the PMD source code analyzer (https://pmd.github.io/). The paper discusses some of the vulnerabilities found, classified under the Buffer Overflow, Code Injection, Memory Leaks, and Time of Check vs. Time of Use (TOCTOU) categories. This analysis resulted in 7 new CVE requests and the identification of 342 existing CVEs and 894 vulnerable code clones.

In this paper, we assess the current state of the vulnerabilities in IoT software utilized in smart grids applications from a source code point of view. To that end, we identified and analyzed open-source software that is used in the power grid and the IoT domain that varies in characteristics and functionality, ranging from operating systems to communication protocols, allowing us to obtain a more complete view of the vulnerability landscape. The source code vulnerability assessment was performed through Static Application Security Testing (SAST). Specifically, the SonarCloud and the Codacy platforms was used for this purpose. The results were subsequently assessed manually, in order to discern true positives and false positives. True positives correspond to reports that actually involve some security issue and needs to be corrected, whereas false positives correspond to reports where the code does not pose any security concern, and no action needs to be taken. Reference [34] identifies two main root causes for false positives in software security analysis: (a) technical issues, where the detection engine is not able to properly analyze the call stack, the control flow or the referenced objects, resulting in an incorrect warning; (b) contextual issues, where the code that has been flagged cannot be executed in contexts where the issue would not be exploited (e.g., debug mode, which is normally turned off in production environments, or operations in trustworthy environments).

Furthermore, the results were categorized using a customized vulnerability categorization based on the OWASP Top 10 list [31] and the MITRE Common Weakness Enumeration (CWE) categorization scheme [35], as well as on the occurrence frequency and possible impact of the identified vulnerabilities in our dataset. The results of this study can be used in the domain of software development, to enhance the security of produced software, as well as in the domain of automated software testing, for the targeting of improvements to the vulnerability detection mechanisms, especially with a focus on the reduction of false positives.

Our work advances the state-of-the-art in the current literature in the following respects:

1. It provides an updated and comprehensive landscape of software security faults in the domain of IoT software used for the implementation of 5G devices and smart grids. To support comprehensiveness, we identified and analyzed software that is widely used in the target domain. The comprehensiveness of the landscape is further supported by using a wide multitude of tools for automated security scanning, which includes all scanners used by the platforms used in our experiments for the software security faults analysis, namely Codacy (Codacy scanners: https://docs.codacy.com/getting-started/supported-languages-and-tools/) and SonarCloud (languages and scanner rules for SonarCloud: https://www.sonarqube.org/features/multi-languages/).
2. It provides a classification of software faults into standardized categories, supporting the awareness of developers and researchers on the types of software security issues that may appear in software within the target domain.
3. It includes an extensive discussion on true and false positives, which can be exploited by researchers and practitioners to better focus on the areas needing review.
4. The paper results may be utilized in the IoT device software development domain, to raise awareness of software developers and testers so as to improve the security of built software, as well as in the domain of automated software testing, targeting improvements to the vulnerability detection mechanisms. Additionally, the results can be used in the overall security and risk assessment of 5G systems.

The rest of this paper is structured as follows: Section 2 presents the software that was used in the vulnerability analysis, and the categorization scheme that was utilized for vulnerability classification. Section 3 presents the results of the analysis, while Section 4 discusses the results obtained, concludes the paper and outlines future work.

2. Materials and Methods

In order to assess the current state of the software vulnerabilities in IoT software, we conducted an extensive search for sources of software that is extensively used in the context of IoT devices. The software could be used at any level of the software stack, including operating system-level, application-level and library-level. Open-source repositories, such as GitHub (https://github.com) and SourceForge (https://sourceforge.net/), but also extended to generic web searches to cover custom distributions available from company or organization sites. The evaluated software is presented in Section 2.1.

The following criteria were taken into account in the process of selecting the software to be analyzed was as follows:

- The software should be widely applicable in the implementation of 5G-enabled IoT devices.
- The overall software selection should provide the broadest possible coverage regarding 5G-enabled IoT device software stack components,
- The languages in which the software is implemented should at least cover C/C++ (mostly used for low-level implementations), Java (a portable compiled language used in many platform and application implementations) and Python (a scripting language widely employed for realizing applications, services and utilities).

When multiple implementations for some specific functionalities were found on GitHub (e.g., implementations of the IEC 61850 protocols), the implementation with the highest number of stars and forks was selected, since this is bound to be more widely used. For the one specific functionality (the implementation of the ModBus protocol), we opted to include two distinct implementations, to demonstrate that security issues across different realizations of the same functionality may diverge significantly.

The selected software was subsequently analyzed for security issues. Code security analysis was performed using automated security tools, and the results were subsequently manually inspected to distinguish true positives from false positives. For the automated analysis we used the SonarCloud (https://sonarcloud.io/) and the Codacy (https://app.codacy.com/) environments. In SonarCloud the following security analysis tools were utilized: SonarScanner for Maven (https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-maven/) and for Gradle (https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-gradle/) (Java); SonarScanner CLI (https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/) (C/C++); and automatic analysis (online) for Python. In Codacy, all rules were turned on. The results were then processed and organized according to security issue categories; to build the categorization scheme we examined the software issue categories recorded in OWASP top-10 [31] as well as the MITRE Common Weakness Enumeration (CWE) categorization scheme [35]. The IoT software issue categorization scheme used in this paper is presented in Section 2.2.

Each reported issue was independently assessed by two reviewers, who inspected the code related to the alleged vulnerability, considering the variables/objects involved and the context within which the code is executed. For true positives, the reviewer also classified the issue under the most relevant category. In case of a disagreement, the two reviewers discussed the different viewpoints, until a consensus was reached. Overall, the process used for conducting the security review is a combination of automated and manual review, following the prescription of MITRE [36], and is in-line with the procedures used to assess the security of software used in specific application domains, e.g., as reported for SDN controllers [37] and medical devices [38].

2.1. Evaluated Software

In the following paragraphs we present the software used in our evaluation. For each software, we present a brief description, including the functionalities of the smart grid devices that the software supports. Characteristics of all the software analyzed, such as source code location, release date, version etc. are summarized in Table 1.

2.1.1. jSML (Java Smart Message Language)

jSML is an open-source Java library that implements the Smart Message Language (SML) communication protocol [39], and is made available under the Mozilla Public License v2.0. SML is used for the transmission of smart meter data with messages following a similar notation to that of ASN.1 [40]. The message encoding rules defined by the SML standard are similar but not identical to those of the Basic Encoding Rules (BER) encoding [40]. The jSML library can be used to construct, encode and send SML messages as well as receive and decode them. Additionally, it provides an implementation of the necessary SML transport layers for communication over TCP/IP and serial connections.

Table 1. Software projects used in the analysis.

Software	Brief Description	Version	Release Date	Language
jSML (https://www.openmuc.org/sml/)	Java implementation of the Smart Message Language	1.1.2	19 Oct. 2018	Java
lib60870 (https://github.com/mz-automation/lib60870)	C implementation of the IEC 60870-5-101 and 60870-5-104 protocols	2.3.0	16 Oct. 2020	C
libiec61850 (https://github.com/mz-automation/libiec61850)	Implementation of the MMS, GOOSE and SMV protocols	1.4.2.1	04 Apr.2020	C
JavaSMQ (https://github.com/RealTimeLogic/JavaSMQ)	Implementation of the SMQ, publish–subscribe system	N/A	07 Dec. 2018	Java
Pymodbus (https://github.com/riptideio/pymodbus)	A full Modbus protocol implementation	2.5.1	28 Mar. 2021	Python
Modbus4j (https://github.com/infiniteautomation/modbus4j)	A Java implementation of the Modbus protocol	3.0.3	8 Jan. 2020	Java
Minnow server (https://github.com/RealTimeLogic/MinnowServer)	Small and fast embedded HTTP(S) WebSocket server	N/A	03 Apr. 2019	C
Boa Webserver (https://github.com/shrugly/boa-0.94.13)	simple and lightweight HTTP server for embedded devices	0.94.13	30 Jul. 2002	C
Thttpd (https://acme.com/software/thttpd)	Tiny HTTP server for embedded devices	2.29	23 May 2018	C
MicroWebSrv/2 (https://github.com/jczic/MicroWebSrv)	A micro HTTP server supporting WebSockets and numerous python/micropython features	2.0	12 Oct. 2020	Python
Busybox (https://busybox.net)	A set of tiny versions of many Unix utilities, packaged with a minimal operating system	1.31.1	25 Oct. 2019	C

2.1.2. lib60870

lib60870 is an open-source C library that implements the International Electrotechnical Commission (IEC) 60870-5-101/104 protocol [41,42], which includes provisions for sending basic telecontrol messages between two systems, employing a communication profile that utilizes permanent data circuits between the involved systems. The library is compatible with the C99 standard and it is designed to be easy to use. It implements both the client (master station) and server (slave or controlled station) sides and implements all data types defined in the IEC 60870-5-101/104 standards [41,42] (companion standards to support any of the standards within the 60870-5 family). The API for client/server communications is strictly asynchronous: the requests must be sent using non-blocking functions and the responses and other events must be handled in callback functions. Some of the functionality provided by the library includes but is not limited to: balanced and unbalanced modes support for IEC 60870-5-101 [41], client/server TCP/IP communication and TLS communication support for IEC 60870-5-104 [42].

2.1.3. libiec61850

libiec61850 is an open-source C library for the IEC 61850 standard implementing the MMS (Manufacturing Message Specification) [43], GOOSE (Generic Object Oriented Substation Event) [44] and SMV (Sampled Measured Values) [45] communication protocols. It is used for the creation of IEC 61850 client and server applications with support for Linux, Windows and MacOS PCs as well as embedded systems. The library provides many features which include but are not limited to TLS support, flexible API to connect custom databases, and a C#/.NET API additionally to the C API. The library is compatible with the C99 standard and licensed under a GPLv3 license.

2.1.4. JavaSMQ

JavaSMQ is an open-source Java/Android library implementing the Simple Message Queue (SMQ) protocol [46] which is used for resource-constrained devices communications. SMQ is based on the publish-subscribe model and is similar to publish/subscribe protocols like Message Queuing Telemetry Transport (MQTT) [47]. However, SMQ provides extended functionality to provide useful features for device management such as one-to-one messaging and inclusion of a virtualized sender's address in every message, enabling the recipient to reply to the sender. The SMQ broker is setup in a Barracuda App Server and can run on any device ranging from embedded devices to cloud servers. SMQ enables horizontal scaling and redundancy through the support of clustering. The JavaSMQ library is designed for operation only over TLS connections, which is not the case for SMQ which supports both non-secure and secure (TLS) connections. The software is designed to run on standard Java or Android and is licensed under an EPL 2.0 license.

2.1.5. Pymodbus

Pymodbus is an implementation of the Modbus protocol [48] written in Python. Modbus is an application-layer messaging protocol, mainly targeted to the communication with automation devices. Its newer versions provide authentication and message-integrity protection, through the encapsulation of MODBUS packets in TLS messages. The Pymodbus implementation uses asynchronous I/O for its communications core and the Tornado Python web framework for request/response handling, being thus highly efficient. Pymodbus is distributed under a permissive, non-standard license.

2.1.6. Modbus4j

Modbus4j is an implementation of the Modbus protocol [48] written in Java by Infinite Automation Systems and Serotonin Software. It implements both slave and master functionalities and supports a number of transports including Serial ASCII, Remote Terminal Unit (RTU), TCP, and UDP transports. automatic request partitioning and response data type parsing. It has high performance and ships with an easy-to-use API under the GNU General Public License v3.0, while commercial licenses are also available.

2.1.7. Minnow Server

Minnow server is an embedded HTTPS WebSocket server written in C, which is designed to be small and fast to allow microcontrollers with no external memory to be used as real-time web servers. The implementation follows the HTML5 WebSocket specification [49], which allows JavaScript applications to communicate with embedded devices by opening channels that are persistent, bidirectional, full-duplex and asynchronous. The Minnow Server is designed to host Single Page Applications (SPA) [50] that can be implemented using modern frameworks like AngularJS, ReactJS, and Vue.js. Minnow server offers both open-source and commercial licensing options. The open-source license is GPLv2.

2.1.8. Boa Webserver

Boa is a single-tasking HTTP server written in C. The single-tasking characteristic means that the server does not use multi-process or multi-threading models to achieve concurrent handling of multiple connections; rather, it utilizes an internal state-table for active connections and non-blocking I/O operations to achieve an efficient multiplexing all incoming requests. Forking is only used for CGI programs (which is required), automatic directory generation, and automatic file decompression [51]. The latest stable version of this software was released in 2002 with version number 0.94.13. The latest development version 0.94.14rc21 was released in 2005 [52]. Even though it's outdated and out of support, it can still be found occasionally on embedded firmware images [53]. We used the source code from a project stored on GitHub [53] that has made minimal changes to Boa in order to allow it to build successfully on modern systems. It is licensed under a GPL-2.0 license.

2.1.9. thttpd

thttpd is a HTTP server written in C. It is designed to be lightweight and portable. It runs in a single thread, it does not fork and performs efficient memory allocation [54]. It provides a useful feature called throttling which allows to set the maximum byte rates on URLs or URL groups. Other features include multihoming (i.e., handling systems with multiple network cards, allowing for different features and options to be specified for each network interface), basic authentication and access control. Additionally, it supports the CGI 1.1 specification [55]. The software is licensed under a BSD license.

2.1.10. MicroWebSrv/2

MicroWebSrv is a HTTP Web server written in Python. It supports WebSockets [49], html/python templating and routing handlers for MicroPython [56]. It is designed to be lightweight and easy to integrate. Its successor, the MicroWebSrv2, was written from scratch to meet the demands of the IoT world by making it more efficient. This is achieved through internal mechanisms that work directly at I/O level, operate asynchronously in an end-to-end fashion and manage memory efficiently. These servers are typically used in microcontrollers such as Pycom [57], ESP32 [58], and STM32 [59]. Both versions of the server are licensed under a MIT license.

2.1.11. Busybox

Busybox—also dubbed as "The Swiss Army Knife of Embedded Linux" [60]—is a single-binary application providing lighter versions of common Unix utilities. It was designed with size-optimization in mind and for use in environments with limited resources. The utilities provided have fewer options than their GNU counterparts, but the options provided are the most commonly used ones and behave similarly [61]. BusyBox takes advantage of the common elements shared by the Unix utilities, e.g., utilities like grep and find both use directory recursion code. Since all utilities are included in a single executable, they can share these common elements, thus saving more space [62]. Furthermore, it is designed to be modular, allowing for inclusion and exclusion of utilities at compile time, making embedded systems customization easy [63]. BusyBox is a very popular choice for embedded devices and can be found in most IoT devices [64]. It is licensed under the GPLv2 license.

2.2. Internet of Things (IoT) Software Issue Categorization Scheme

In this section we describe the categorization scheme used for the classification of the IoT device software issues that are reported in the results section (Section 3). The selection of categories started by considering software issue categories recorded in OWASP top-10 [31] as well as the MITRE CWE categorization scheme [35]. The MITRE CWE categorization scheme employs different abstraction levels, namely pillars, classes, base weaknesses and variants, in descending order of abstraction level. From this scheme, we utilize only the "classes" and "base weaknesses" levels in our reporting, since pillars are too generic and variants are too specific.

Subsequently, the software issues collected after analyzing the software (and filtering out false positives) were assigned to categories. For conciseness purposes, only selected software issue categories are presented in the results section; the categories that are included in the presentation were chosen based on two factors: (1) the occurrence frequency of a category in the generated reports and (2) the possible impact of a category, with higher impacts being favored over lower ones.

Since the categories of the OWASP categorization scheme overlap, the category that was deemed to better describe the nature and the characteristics of the identified issues was selected for use in the result reporting section.

2.2.1. Improper Certificate Validation

Improper certificate validation vulnerabilities occur when the software in question does not perform certificate validation or performs it incorrectly. The incorrect validation can occur through various inconsistencies in the code like improper validation of certificate expiration, revocation, chain of trust, etc. The possible impacts of this vulnerability include an attacker gaining privileges and/or assuming a trusted identity which could lead to complete host takeover [65]. For example, an accepted malicious certificate might allow an attacker to perform an MITM attack by spoofing a trusted entity and intercepting the communications between client and host. This can include the client accepting spoofed data thinking it originated from the host. An attacker could also mislead the client into connecting to an attacker-controlled host posing as a legitimate one [66].

The "Improper Certificate Validation" software issue category is associated with the OWASP Top 10 2017 Category A6 "Security Misconfiguration" [31] and the MITRE CWEs listed in Table 2:

Table 2. MITRE Common Weakness Enumerations (CWEs) related to Improper Certificate Validation.

CWE ID	CWE Description
MITRE CWE-295	Improper Certificate Validation" [65] (class-level category)

2.2.2. Buffer Overflow

Buffer overflow occurs when a program inserts data in a buffer with the data size exceeding the buffer's capacity and overwrites the subsequent space in memory [67]. A buffer overflow can cause a system crash, but it can also be exploited by an attacker to control the execution flow of the vulnerable program or even to execute arbitrary code with the privileges of the exploited program [68]. In a typical buffer overflow exploitation scenario, the overwritten memory contains the return address of the exploited function, i.e., the memory address that points to what will be accessed/executed next. An attacker would try overwriting this address with an address of his choosing. The new address usually points to a part of the memory where her/his malicious code resides [69]. Buffer overflows are more common in languages like C/C++ where the developers must perform memory management manually, unlike modern languages like Java and C# that have built-in features preventing buffer overflows [70].

The "Buffer overflow" software issue category relates to the OWASP Top 10 2017 Category A9 "Using Components with Known Vulnerabilities" [31] as well as with the MITRE CWE categories listed in Table 3:

Table 3. MITRE CWEs related to Buffer overflow.

CWE ID	CWE Description
MITRE CWE-676	Use of Potentially Dangerous Function [71] (base-level category)
MITRE CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer [72] (class-level category)
MITRE CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') [73] (base-level category).

2.2.3. Weak Cryptography

Weak cryptography can be the source of various vulnerabilities as many security-sensitive operations rely heavily on cryptography. Some of the most common weaknesses encountered in vulnerable applications are the use of insecure algorithms (e.g., DES, MD5, etc.), the use of insecure pseudo-random generators, the insecure implementation/utilization of secure encryption/hashing algorithms, and using algorithms for cryptographic purposes outside the recommended ones (e.g., password hashing with insecure/"fast" hash functions) [21]. For applications requiring encrypted communications, issues can occur by using insecure versions or/and insecure configuration of SSL/TLS [74]. Weak cryptography can affect the confidentiality, integrity and availability of an application, with its impact ranging from information disclosure to complete host takeover.

The "Weak Cryptography" software issue category is associated with the OWASP Top 10 2017 Categories A3 "Sensitive Data Exposure" and A6 "Security Misconfiguration" [31], as well as with the MITRE CWE categories listed in Table 4:

Table 4. MITRE CWEs related to weak cryptography.

CWE ID	CWE Description
MITRE CWE-326	Inadequate Encryption Strength [75] (class-level category)
MITRE CWE-327	Use of a Broken or Risky Cryptographic Algorithm [76] (class-level category)
MITRE CWE-330	Use of Insufficiently Random Values [77] (class-level category)
MITRE CWE-338	Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)" [78] (base-level category)
MITRE CWE-916	Use of Password Hash with Insufficient Computational Effort (base-level category)

2.2.4. Sensitive Data Exposure

Sensitive data exposure occurs when an application does not take the appropriate measures to protect the sensitive information handled by it. Sensitive data can include anything from personally identifiable information (PII) to credit card information and passwords [74]. It can occur through insecure cryptographic practices, e.g., insecure hashing of stored passwords, insecure communications, e.g., misconfigured HTTPS vulnerable to downgrade attacks [21], leftover debug code [79], SQL injection, and more. This category is closely related to the Weak Cryptography category. The impact of this category ranges from information disclosure to complete host takeover.

The "Sensitive data exposure" software issue category is associated with the MITRE CWE categories listed in Table 5:

Table 5. MITRE CWEs related to sensitive data exposure.

CWE ID	CWE Description
MITRE CWE-200	Exposure of Sensitive Information to an Unauthorized Actor [80] (class-level category)
MITRE CWE-215	Information Exposure Through Debug Information [81] (base-level category)
MITRE CWE-319	Cleartext Transmission of Sensitive Information [82] (base-level category)
MITRE CWE-489	Active Debug Code [79] (base-level category)

2.2.5. Race Condition

A race condition is a condition of a program where its behavior depends on relative timing or interleaving of multiple threads or processes [83] and can be found in multithreaded programs. A race condition attack includes forcing a system that's implemented to perform some tasks in a predefined sequence to perform two or more operations simultaneously. It exploits the time gap between the separate parts that consist an operation interfering with the intended sequence of execution. For example, when a database update takes place, an attacker could execute a race condition attack. The attack may be possible because the update is not executed in one single command, so during the update process, which could take little or much time, the system is unprotected (assuming lack of the relevant controls). During this period an attacker can compromise the system with malicious queries [84]. The potential impacts of a race condition attack affect the confidentiality, integrity, and availability of the vulnerable application and can even result in complete host takeover.

The "Race condition" software issue category is associated is associated with the OWASP Top 10 2017 Category A3 "Sensitive data exposure" and A6 "Security Misconfiguration" [31], as well as with the MITRE categories listed in Table 6:

Table 6. Race condition.

CWE ID	CWE Description
MITRE CWE-367	Time-of-check Time-of-use (TOCTOU) Race Condition [85] (base-level category)

2.2.6. Broken Access Control

Broken access control occurs when the access control mechanisms in place can be bypassed to perform unauthorized actions. An attacker can have unauthorized access to information, manipulate data, and even compromise the system completely [86]. The impact of these attacks can affect the confidentiality, integrity and availability of the application. Access control vulnerabilities are typically divided in horizontal and vertical privilege escalation vulnerabilities, where the first refers to a user performing actions or accessing data of another user with the same level of permissions and the second refers to performing actions or accessing data of users with elevated privileges [87]. An example of broken access control are insecure direct object references (IDOR) vulnerabilities, where the persistent object can be accessed directly through the application interface, e.g., a URL that includes the internal id of the object [88]. The impact of broken access control exploitation can range from information disclosure to complete host takeover.

The "Broken access control" software issue category is associated is associated with the OWASP Top 10 2017 Category A5 "Broken Access Control" [31], as well as with the MITRE categories listed in Table 7:

Table 7. Broken Access Control.

CWE ID	CWE Description
MITRE CWE-266	Incorrect Privilege Assignment" [89] (base-level category)
MITRE CWE-732	Incorrect Permission Assignment for Critical Resource" [90] (class-level category)

3. Software Vulnerability Analysis Results

In this section we present the results obtained from the analysis. Each subsection contains the results for each project summarized in a table along with some notes about the results and their assessment. SonarCloud uses a categorization of security-related issues, the categories are Vulnerability and Security Hotspot. The documentation states that the main difference between them is "the need of a review", i.e.:

- vulnerabilities are locations in the code for which the analyzer computes with a high degree of confidence that exploits can be crafted and successfully applied, leading to security incidents.
- security hotspots are locations in the code which are tagged as "suspicious" by the analyzer, albeit with a lower degree of confidence. Security hotspots should be more

rigorously examined by developers to determine whether they actually introduce some vulnerability or not.

We recognize this distinction in the presentation of our results with the corresponding categories "vulnerabilities" and "hotspots".

It is important to note here that code for which no vulnerabilities and no hotspots have been reported by the tools is not necessarily security risk-free. Security issues may exist and skip detection by automated tools and even during security expert code reviews. In the analysis presented in this paper, this is demonstrated by the existence of two known vulnerabilities in the Boa server, which evaded detection (c.f. Section 3.8). No claim is made that the analysis results provide a full report of the existing vulnerabilities, however the goal to portray the most common software security issues in 5G IoT device software can still be achieved, even if few security issue instances are missed.

For conciseness, the report for each software includes only the security issue categories for which at least one vulnerability or hotspot was detected; the remaining security issue categories are omitted.

3.1. *jSML*

Table 8 lists the number of security issues of different categories identified in jSML. Two vulnerabilities were reported for this application, both relating to improper certificate validation (CWE-295). Specifically, the application was found to have defined methods for certificates' validation that were empty.

Table 8. Number of security issues of different categories in jSML.

Security Issue Category	# of Vulnerabilities		# of Hotspots	
	Total	**False Positives**	**Total**	**False Positives**
Improper Certificate Validation	2	0		

3.2. *lib60870*

Table 9 lists the number of security issues of different categories identified in lib60870. The vulnerabilities reported for this application are categorized in the "buffer overflow" category (CWE-676 and CWE-119) and miscellaneous category.

Table 9. Number of security issues of different categories in lib60870.

Security Issue Category	# of Vulnerabilities		# of Hotspots	
	Total	**False Positives**	**Total**	**False Positives**
Buffer Overflow	5	2	9	9
Misc	2	2	0	0

All buffer overflow reports are associated with the use of the *sprintf* function; two of them are safe, since the buffers are carefully allocated to be able to store all characters to be stored by the *sprintf* function, while in three cases it is possible that the payload may exceed buffer capacity.

Similarly, nine security hotspots are reported for this application (5 by SonarCloud and 4 by Codacy) and all of them are categorized in the "buffer overflow" category (CWE-676 and CWE-119). Specifically, four issues were associated with the use of the *strcpy/strncpy* functions, however the manual review showed that the corresponding memory areas have been allocated to have ample size, hence no vulnerability is introduced by the use of these functions. Five issues are associated with the use of the *strlen* function, which again operates in a secure fashion since the argument is verified to be a non-null pointer and the contents of the buffer indicated are valid and appropriately terminated by a NULL character.

Codacy reported two instances of using the *usleep* function as vulnerabilities, under the rationale that the interaction of this function with the alarm signal (SIGALRM) and other timer functions such as *sleep(), alarm(), setitimer(),* and *nanosleep()* is unspecified (CWE-676). However, the application does not use any of these functions, therefore the two reports are characterized as false positives.

3.3. libiec61850

Table 10 lists the number of security issues of different categories identified in libiec61850. All buffer overflow reported vulnerabilities are associated with the use of the *sprintf* function, which operates however on buffers that are able to hold the relevant payload in its entirety (CWE-676 and CWE-119).

Table 10. Number of security issues of different categories in libiec61850.

Security Issue Category	**# of Vulnerabilities**		**# of Hotspots**	
	Total	**False Positives**	**Total**	**False Positives**
Buffer Overflow	7	7	392	100%
Sensitive Data Exposure	22	13		
Misc	3	3		

Buffer overflow hotspots were associated with a variety of string handling functions (*strcpy*, *strlen*, *sprintf* etc.), the usage of which has been found to be safe in the contexts they were used. Of the hotspots reported by each tool, 30% were manually inspected and the code in all instances was found to be safe, since it operates on buffers of adequate size. Considering the results of the manual inspection, a 100% false positive rate is reported in the table above.

Sensitive data exposure issues were associated with logging statements, which were directed to the console (or a file, if the console was redirected, which is a typical case). Of these messages, 13 have a constant text, which does not disclose any data and are classified as false positives. The remaining nine, include data objects that may leak and therefore should not be logged, except in debugging contexts, and therefore these instances are classified as true positives. Under a more strict interpretation, even the constant texts displayed disclose the operation that is being carried out, and consequently all 22 sensitive data exposure instances could be classified as true positives.

Codacy reported three instances of using the *usleep* function as vulnerabilities, under the rationale that the interaction of this function with SIGALRM and other timer functions such as *sleep(), alarm(), setitimer(),* and *nanosleep()* is unspecified (CWE-676). However, the application does not use any of these functions, therefore the two reports are characterized as false positives.

3.4. JavaSMQ

Table 11 lists the number of security issues of different categories identified in JavaSMQ. Three vulnerabilities were reported in this software, which were classified under the “Improper Certificate Validation” category (MITRE, CWE-295). Specifically, the certificate is not validated and the hostname in the certificate is not verified to match the actual hostname. Additionally, the report included a security issue under the “Weak Cryptography” category (CWE-327 and MITRE, CWE-326), where the application uses the insecure SSL protocol instead of TLS.

Table 11. Number of security issues of different categories in JavaSMQ.

Security Issue Category	**# of Vulnerabilities**		**# of Hotspots**	
	Total	**False Positives**	**Total**	**False Positives**
Improper Certificate Validation	3	0		
Weak Cryptography	1	0		

3.5. *pyModbus*

Table 12 lists the number of security issues of different categories identified in pyModbus. The pyModbus library analysis did not report any vulnerabilities, however a number of hotspots were identified.

Table 12. Number of security issues of different categories in pyModbus.

Security Issue Category	**# of Vulnerabilities**		**# of Hotspots**	
	Total	**False Positives**	**Total**	**False Positives**
Buffer overflow			2	2
Weak Cryptography			24	5
Sensitive Data Exposure			52	5
Misc			7	6

Two buffer overflow errors were associated with the use of the *strlen* function, which however operates on properly verified buffers and is, therefore, safe.

Out of the 24 weak cryptography" hotspots reported, 22 pertain to the use of URLs using the insecure "http" protocol (as contrasted to the use of the secure "https" protocol), while 5 correspond to the use of the *random.randint()* function, which is known to be inadequate for security-related tasks.

Considering the issues corresponding to the use of "http" URLs, 18 of them serve documentation purposes, however they are printed in log files and user interaction screens, and the user may then directly utilize the URL, which may lead to visits to attacking websites (e.g., through DNS poisoning). One "http" URL usage is embedded in a setup program which utilizes the URL to download packages; in this context the use of an "http" URL may lead to an attack with a higher probability. Under the rationale presented above, all 19 usages of "http" URLs are classified as true positives.

Considering the five "weak cryptography" hotspots reported, all of them pertain to the use of the *random.randint()* function, which is known to be inadequate for security-related tasks (CWE-330 and CWE-338). Two uses are located in testing code, while three more correspond to uses within the main application library code. However, these functions are used for internal purposes only, for which no plausible exploitation scenario can be formulated. Consequently, all reported "weak cryptography" hotspots are classified as false positives.

Out of the 52 "Sensitive Data Exposure" hotspots, 5 correspond to the detailed printing of stack traces in the event of an exception data (CVE-2018-1999007, CVE-2015-5306 and CVE-2013-2006) and 4 pertain to the usage of publicly writable directories.

The code related to the detailed printing of stack traces in the event of an exception is used solely in test programs, and hence does not pose any security risk, especially considering that typically code developed for production environments includes specialized exception handlers, and therefore the probability that this practice is directly ported into a production environment is low. Under this viewpoint, these 5 reports are classified as false positives.

Regarding the 43 reports pertaining to the use of loggers, 40 of them are found in example programs and three in the application's main code. The use of loggers is not a security vulnerability on its own right; however, a misconfigured logger may lead to leakage of security-sensitive data, which can be directly exploited (e.g., credentials) or serve as additional information to facilitate the attackers' work (e.g., system paths, values of parameters and so forth). Considering that developers may use the example with minimal changes to serve their purposes, it would be strongly desirable for examples to be crafted in such ways that they minimize risks when used in production environments (e.g., use a hardened configuration), or at least provide heads-up notices for developers and system administrators regarding the security risks involved and the best practices that can be followed. From this viewpoint, all 43 logger-related "sensitive data exposure" warnings are considered as true positives.

The four reports corresponding to the use of publicly writable directories are again in examples, but in the same line of thought (probability that the code is used directly into production applications with no security-conscious adaptations), they are considered as true positives.

The 6 reports under the "Misc" category correspond to the following cases:

- one use of code a regular expression (CVE-2017-16021 and CVE-2018-13863): Inappropriately crafted regular expressions may necessitate high central processing unit (CPU) power memory amounts to be evaluated, and consequently lead to resource depletion and denial of service attacks. However, in this case the regular expression is safe to use and, furthermore, it is used for configuration purposes, at the beginning of the program execution.
- One binding of the server to all available interfaces (using a bind address equal to 0.0.0.0) which, however, is desirable in order to serve all interfaces, and hence it is classified as a false positive.
- Four uses of the *pickle* Python library, which may involve security hazards if communicating with unknown peers and/or using unencrypted channels. None of these conditions are met however here, hence the reports are classified as false positives.
- One instance of using an XML file without proper validation. Typically, the relevant file is provided by the administrator; however, safeguarding the XML file processing procedure using more secure method counterparts or options would be desirable.

3.6. Modbus4j

Table 13 lists the number of security issues of different categories identified in Modbus4j. For the Modbus4j library, two vulnerabilities and two security hotspots were identified. The vulnerabilities, classified under the "misc" category, correspond to cases where the return values of methods (*File.delete()* and *File.renameTo()*) are not properly checked. This is a direct instance of the MITRE CWE-754 weakness, "Improper Check for Unusual Exceptional Conditions", which may lead to silently failing backup operations.

Table 13. Number of security issues of different categories in Modbus4j.

Security Issue Category	# of Vulnerabilities		# of Hotspots	
	Total	False Positives	Total	False Positives
Sensitive data exposure			2	0
Misc	2	0		

The 2 "Sensitive Data Exposure" hotspots correspond to the detailed printing of stack traces in the event of an exception. The relevant exceptions are within the main application code, and may expose sensitive data (CVE-2018-1999007, CVE-2015-5306 and CVE-2013-2006). Therefore, the relevant instances are classified as true positives.

3.7. Minnow Server

Table 14 lists the number of security issues of different categories identified in the Minnow Server. The Mininow server analysis did not report any vulnerabilities; however, a number of hotspots were identified. A "Weak Cryptography" security hotspot was verified as a vulnerability, since the *Math.random* function—which is known to be insecure due to the predictability of the pseudo-random number sequence—was used to produce random values (CWE-330 and CWE-338). The random values are used for collision backoff when generating unique IDs, and hence could be exploited for DoS attacks.

Table 14. Number of security issues of different categories in the Minnow Server.

Security Issue Category	# of Vulnerabilities		# of Hotspots	
	Total	**False Positives**	**Total**	**False Positives**
Buffer Overflow			6	6
Weak Cryptography			1	0
Sensitive Data Exposure			2	2
Misc			1	1

SonarCloud detected two Sensitive Data Exposure hotspots. However, the manual review concluded they are both false positives. In the first case, information is presented to the user through Javascript's alert function, however this is only information of debug nature that is already present in the client. In the second case, an alert message is displayed, which is a constant string that does not disclose sensitive information.

Six security hotspots under the "Buffer Overflow" category were identified, however their manual inspection resulted in their characterization as false positives, since they operate on appropriately allocated and validated string buffers.

Finally, one issue classified under the "Misc" category pertains to the use of code a regular expression (CVE-2017-16021 and CVE-2018-13863): Inappropriately crafted regular expressions may necessitate high CPU power memory amounts to be evaluated, and consequently lead to resource depletion and denial of service attacks. However, in this case the regular expression is safe, so the report is classified as a false positive.

3.8. Boa Webserver

Table 15 lists the number of security issues of different categories identified in the Boa Webserver. The analysis indicated four vulnerabilities that are classified under the "buffer overflow" category (MITRE CWE-676, MITRE CWE-119 and MITRE CWE-120). Two of these locations are quite improbable to exploit, since they relate to the configured host name and this is defined by the administrator, however the code should be amended to be fully secure. One reported vulnerability has been verified to be a false positive, while one additional bug has been verified to introduce in the software a case that can be exploited.

Table 15. Number of security issues of different categories in the Boa Webserver.

Security Issue Category	# of Vulnerabilities		# of Hotspots	
	Total	**False Positives**	**Total**	**False Positives**
Buffer Overflow	4	1	46	46
Weak Cryptography			2	2
Broken Access Control			2	1
Misc.			5	

Considering the reported security hotspots, for two cases categorized under Weak Cryptography, the *HTTP* protocol is hardcoded and this is reported as insecure (MITRE CWE-326); however, the version of the BOA server used in the analysis does not support *HTTPS*, hence the use of the *HTTP* protocol is expected.

Forty two buffer overflow hotspots were reported relating to uses of the *strlen* and *sctrcpy* functions. However, the strings referenced therein have been already checked for their size and the uses of the functions are safe. Four more buffer overflow hotspots were associated to reading of data in loops, however the code was verified to be secure.

One of the "broken access control" security hotspot relates to the existence of a library function that creates files in a fashion that grants read permissions to the general public (others), which may lead to sensitive data exposure (MITRE CWE 266 and MITRE CWE 732). While the creation of files with this level of permission may be desirable in some cases, it would be preferable for the calling code to explicitly specify the required permission level, rather than implicitly relying on the appropriate setting of the *umask* parameter [91], which is an implicit requirement and thus more probable to evade the developer's attention. The second broken access control security hotspot relates to the setting of *umask* to the value of *~0600*, which is highly secure; however, it appears that the bitwise negation operator could not be properly interpreted by the security analyzer.

In five cases reported under the "misc" category, the return values of functions are not properly checked (CWE-252); the relevant resources (directories) have been checked at previous times, however their status may have changed. Specifically, this relates to the absence of checking the return value of the function *chdir*, which could lead to broken access control, including the misinterpretation of relative pathnames.

Finally, it is worth noting that two known vulnerabilities exist for this version of the software, which SonarCloud failed to detect. The exploitation of vulnerability CVE-2009-4496 (https://nvd.nist.gov/vuln/detail/CVE-2009-4496) could allow an attacker to execute arbitrary commands or overwrite files through a specially crafted *HTTP* request containing an escape sequence for a terminal emulator. This vulnerability can be traced back to lack of input sanitization of non-printable characters. The other vulnerability (https://github.com/shrugly/boa-0.94.13/blob/master/README.md#unfixed; accessed on 13 May 2021) occurs from an insecure configuration of the application, where failing to use the *DefaultType* in *boa.conf* results in requests with invalid MIME type leading to a segmentation fault. This can be exploited for executing a DoS attack.

3.9. *thttpd*

Table 16 lists the number of security issues of different categories identified in the thttpd Webserver. Buffer overflow bugs are falsely reported, since the relevant uses of *sprintf* use carefully allocated buffers and/or size specifiers within the format.

Table 16. Number of security issues of different categories in the thttpd webserver.

Security Issue Category	**# of Vulnerabilities**		**# of Hotspots**	
	Total	**False Positives**	**Total**	**False Positives**
Buffer Overflow	5	5	155	155
Weak Cryptography			1	1
Sensitive Data Exposure			1	0
Race Condition	3	3		

Race conditions are reported owing to the existence of "time-of-check vs. time-of-use" conditions; yet, adequate checks are made in the time of use and the time of check is only used as an optimization method, therefore no vulnerability exists.

The sensitive data exposure hotspot relates to the creation of a directory with extraneous permissions (read and execute permissions are granted to "group" and "others"), which may lead to data leakage (CWE-266 and MITRE CWE-732).

Considering the hotspots reported, one hotspot is categorized under the "Weak cryptography category", where the *HTTP* protocol is hardcoded; however, the version of the *thttpd* server used in the analysis does not support *HTTPS*, hence the use of hardcoded *HTTP* is justified.

Many uses of *strlen*, *strncpy* and *sctrcpy* are reported as possible buffer overflows, yet the strings referenced therein have been already checked for their size, or the size of the destination buffer is correctly defined (*strncpy*), hence the uses of the functions are safe.

3.10. *MicroWebSrv*

Table 17 lists the number of security issues of different categories identified in the MicroWebSrv Webserver. This software was found to have only one hotspot, which was classified under the “Weak cryptography” category. Specifically, the SHA-1 algorithm—which is proven to be insecure—is used in the context of the creation of an *HTTP* response (CWE-916), therefore the security hotspot is verified to be a true positive.

Table 17. Number of security issues of different categories in the MicroWebSrv webserver.

Security Issue Category	# of Vulnerabilities		# of Hotspots	
	Total	False Positives	Total	False Positives
Weak Cryptography			1	0

3.11. Busybox

Table 18 lists the number of security issues of different categories identified in Busybox. Busybox was the software for which the largest number of bugs and hotspots were detected. This was expected, since BusyBox is a complete operating system containing also a number of applications, and its source code tree is much larger that the source tree of the other software used in the analysis.

Table 18. Number of security issues of different categories in Busybox.

Security Issue Category	# of Vulnerabilities		# of Hotspots	
	Total	False Positives	Total	False Positives
Buffer Overflow	83	100%	226	100%
Weak Cryptography			1	
Race Condition	9	1		
Broken Access Control			42	40
Misc			22	22

The analysis reported nine Race Condition issues (MITRE CWE-367); out of them, one was assessed to be a false positive. Specifically, a file is first removed (if existent) and then created back; however, the return values are properly checked and no vulnerability is introduced. Out of the remaining eight bug reports classified under the “Race Condition” category, one case of permission changes may be refactored to be more secure, and seven cases could be refactored to remove any race condition window.

A total of 83 “buffer overflow” alleged vulnerabilities were reported; after verifying 30% of the reported bugs, a false positive rate of 100% was estimated: all bugs were associated with the use of string handling functions (*sprint*, *strlen*, *strcpy*, *strcat* etc.), however the string buffers used therein are appropriately allocated and validated, therefore no vulnerabilities are introduced.

One hotspot was classified under the “Weak Cryptography”; this hotspot refers to one hardcoded use of the insecure *HTTP* protocol instead of the secure *HTTPS* protocol (CWE-326). 212 buffer overflow hotspots were associated with the use of string functions (*strcpy*, *strlen*, *strcat* etc.). After manually inspecting 30% of the reported cases, it was determined that functions are used safely by applying the appropriate checks on their arguments. 14 more buffer overflow hotspots were related to the use of loops for reading input, however they were verified to be safe. Taking the above into account, a false positive rate of 100% is reported in the table.

Broken access control notifications refer to (a) granting of permissions to “others” (i.e., general public) which, however, is the standard Unix behavior and can be regulated further by users and applications especially through the use of explicit permission setting or the use of the *umask* feature (CWE-266 and MITRE CWE-732); (b) ensuring that a suitable directory is selected after *chroot* (a special operation available under Unix where the program operates within a limited and restricted view of the filesystem) (CWE-754) and (c) setting of capabilities (CWE-754). Out of these items, two errors falling in case (b) are deemed to be true positives, i.e., items requiring intervention to ensure secure operation.

Codacy reported 22 instances of using the *usleep* function as vulnerabilities, under the rationale that the interaction of this function with SIGALRM and other timer functions such as *sleep(), alarm(), setitimer(),* and *nanosleep()* is unspecified (CWE-676). However, the application does not use any of these functions, therefore the two reports are characterized as false positives.

4. Discussion

In this paper, we have provided a map of the status of vulnerabilities in software that is used for IoT devices in the context of Smart Grids, which was compiled by conducting security analysis on the code of open-source projects. The goal of this mapping is to raise developer awareness regarding common security pitfalls in IoT device for Smart Grids software development, so that developers would avoid relevant errors and adopt secure software coding practices. Table 19 depicts the overall statistics of the software security analysis performed on the 11 software packages examined in this paper.

Table 19. Overall statistics of the analysis performed on all examined software.

Software Bug Category	**# of Bugs**		**# of Hotspots**	
	Total	**False Positives**	**Total**	**False Positives**
Improper Certificate Validation	2	0	0	0
Buffer Overflow	104	98	836	836
Weak Cryptography	1	0	30	8
Sensitive Data Exposure	22	13	57	7
Race Condition	12	4	0	0
Broken Access Control	0	0	44	41

While buffer overflows are the most reported type of security issue, both under the "vulnerability" and the "security hotspot" classifications, the vast majority of these reports are false positives. This indicates that developers are aware of the caveats present in languages that are prone to buffer overflows (predominantly C and C++), as well as the severity of this type of faults, and carefully develop and scrutinize their code to ensure that it is buffer overflow-free. On the other hand, we can observe that code analysis tools need further development to increase the precision of software vulnerability reporting, limiting the number of buffer overflow false positive reports. It has to be noted here that the features of the C/C++ language, such as dynamic memory (re)allocation, pointer passing, pointer arithmetic, forced typecasting etc. make the accurate detection of buffer overflows very difficult, and therefore code analysis tools may opt to overreport vulnerabilities, instead of risking not reporting some security-related bug.

Improper or non-existent certificate validation was identified as a bug in two of the analyzed software packages; this number is deemed to be high, since some of the tested software did not provide any support for encrypting data-at-transit, hence this type of vulnerability could not occur therein. Contrary to the case of buffer overflows, it appears that developers are not adequately informed of the risks associated with improper certificate validation or familiarized with the best practices for handling encrypted communications. Notably, this holds despite the fact that this type of error is recorded in the OWASP 2017 top ten [31], however the relevant category "A3—Sensitive Data Exposure" is too broad and, therefore, the association between the error type and the category may not be sufficiently apparent. More awareness needs to be developed, whereas the creation and promotion of code patterns for securely managing certificates could also contribute to the reduction of errors associated with improper certificate validation.

Similar remarks can be recorded for the use of weak cryptography. Developers seem to be unaware of the developments regarding the level of security that different ciphers or randomization functions offer, and use in their code elements that introduce vulnerabilities. It is worth noting that avoiding the use of weak cryptography is a continuous task, since the advent of cryptanalytic techniques, the discovery of flaws in methods and the increase of computer processing power renders methods and techniques that were formerly considered secure as vulnerable, while the introduction of new cryptographic techniques changes the best practice recommendations regarding the configurations used to achieve optimal security.

Race conditions is a class of errors that are present in many multiprocessing/ multitasking environments, where accidentally occurring or purposefully attempted interleaving of operations may lead to a number of hazards [92], notably including deadlocks, access to confidential data, demotion of integrity (including incorrect result computation) or privilege escalation. In order to avoid race conditions, the current toolchest of developers includes a number of instruments, techniques and methods, such as thread-safe libraries, process and thread synchronization methods, library functions that are guaranteed to operate atomically eliminating any TOCTOU-based errors [85] (e.g., the introduction of the *faccess* function enables the use of the secure *fopen*/*faccess* function sequence instead of the TOCTOU-vulnerable *access/fopen* function sequence). The exploitation of experience gained from operating system and parallel application development, coupled with the use of model-based checking tools such as [93–95], which are able to follow all possible application execution paths and more thoroughly cover the possible instruction interleaving sequences, may lead to a reduction of race condition introduction windows.

The broken access control-related vulnerabilities mainly pertain to over-permissive file and directory assignments, as well as omissions to ensure that the correct working directory has been set. The latter cases also relates to a number of security issues that have been classified under the "miscellaneous" category; however, the mechanics of the two error types are different (not invoking a necessary function vs. failure to check the result of some function, respectively). Developers should ensure that new objects are created with the minimum required set of permissions, and additionally ensure that the correct working directory is always set, to avoid misinterpretations of relative pathnames.

Finally, certain known security issues were not discovered by the security analyzers (CVE-2009-4496 and improper handling of invalid MIME type specifications in the *boa* server, c.f. Section 3.6). This fact demonstrates that while automated security analysis is a valuable and indispensable tool in a security-aware development process, it cannot currently replace the rigorous review by security expert teams.

It is worth noting that the methodology used in this paper for identifying the most important software security issues in the domain of 5G-enabled IoT devices is in-line with the procedures used to assess the security of software used in specific application domains (e.g., [27,28]) and can be used in other domains as well. The methodology starts with harvesting widespread software packages used in the domain, while subsequently the packages are analyzed for security issues using a combination of automated and manual methods, following the proposal of MITRE regarding software package security analysis [36].

Designers and implementers of smart grid and 5G-enabled systems need to consider that software deficiencies is only one of the factors that determine the overall system security level, and in this sense, other complementary factors need to be taken into account as well. Reference [14] identifies that the 5G protocol allows for using null encryption and null authentication in valid 5G environments, an option directly poses security and privacy issues. Ref. [14] also asserts that the 5G protocol specification places implicit trust on pre-authentication messages, a feature can be exploited by malicious parties, while additionally a number of LTE protocol exploits may have considerable impact on the security of a 5G system. However, simple specification and protocol deficiencies, such as the support of null encryption/authentication and implicit trust, specifications and protocols may exhibit more subtle issues, subject to specific entity interactions and state transitions. Such issues can be unveiled through exhaustive protocol validation using appropriate tools such as Scyther [96], Scyther-proof [97] and Tamarin [98]. Along these lines, reference [99] examines the vulnerabilities of the IEC 61850 GOOSE publish–subscribe protocol towards the substation protection within the smart grid setup, and assesses their impact. This research has demonstrated that the GOOSE protocol of IEC 61850 entails vulnerabilities and is prone to malicious attacks even under the compliance of the IEC 62351 security standard; these vulnerabilities need to be addressed in subsequent standards and protocol revisions. Reference [100] introduces a fuzzing tester termed as "IECFuzzer" which can be used to exploit the potential denial-of-service vulnerabilities of IEC61850 protocol, but also to verify the robustness of PLC devices using IEC61850 protocol.

Reference [101] presents a formal security assessment of the Modbus protocol through a Promela model, identifying counterexamples where the lack of security mechanisms can be exploited by malicious parties to realize man-in-the-middle attacks and inject falsified data. Similarly, reference [102] uses Scapy [103] to exploit weaknesses in the Modbus over TCP/IP, and reports on two attacks on the Modbus protocol, with one attack resulting in an infection of the master with malware and the second one realizing a man-in-the-middle exploit. Reference [104] introduces a real-time cyber-physical test bed utilizing Opnet's System-in-the-Loop (SITL) simulator for examining security properties, and analyses the Modbus protocol on top of this test bed. The experiments performed unveiled the possibility of a man-in-the-middle attack using carefully timed transmission of crafted packets, as well as the potential of DoS attacks severely delaying the flow of information from Intelligent Electronic Devices (IEDs) to the control center. IoT devices have proliferated and nowadays are extensively used in numerous domains to support a multitude of operations and automations. In the context of smart grids, IoT devices also play an important role, including the metering of consumption as well as power distribution and redirection. The advent of the 5G network technology will further boost the widespread use of IoT-based services and the development of new ones; however, software security for IoT devices is a key enabler for this development, since software vulnerabilities may cause a variety of adverse effects, ranging from private data leakages to the collapse of critical smart grid infrastructures. Developers and testers of code realizing services and applications in the domain of IoT should increase their awareness levels regarding best coding practices and security pitfalls, and employ appropriate tools to ascertain a maximal security level. Similarly, protocol designers should address deficiencies in the specifications that can be exploited to realize attacks against systems that are based on the corresponding protocols. It has to be noted that the examination of the protocols' security properties is performed using simulation, which may be conducted either in software-only fashion [101,102] or using a hardware/system in the loop approach [99,104]. In protocol security analysis simulations, only data transmission and protocol state transitions are modeled within the simulation system; by contrast, in static software security assessment, the properties of the instructions, libraries and instruction sequences are examined to identify potential vulnerabilities. In this sense, the two approaches work complementarily, but cannot be combined due to the fundamental differences in the modeling level and tools (e.g., Petri-Nets, Promela processes and messages, Scyther roles and claims or BAN constructs [105,106] vs. concrete software instructions. Typically, a protocol has first to be verified for correctness and security, and subsequently its implementations need to be checked, to ensure that protocol security guarantees are not undermined due to security issues in the code. In this paper, we focus only on the software security assessment aspects, while work and results on protocols' security properties verification can be used in a complementary fashion.).

Our future work will include the security analysis of additional software in the power grid domain, as well as the examination of software in other domains. We also plan to consider the combined assessment of results concerning communication protocol security through relevant simulation-based tools (e.g., Scyther) on the one hand (e.g., [99–101]), and results from the software implementation security assessment on the other hand, towards the computation of a comprehensive risk level of 5G and smart grid systems. Finally, the effectiveness of model-based checking tools such as [93–95] in the identification of security-related issues, and the interplay between secure software development practices and the operation of forensics in the domain of 5G and Smart Grid computing [107] will be studied.

Author Contributions: Conceptualization, C.-M.M., C.V., N.K., C.C.Z. and M.-A.K.; Investigation, C.-M.M., C.V. and N.K.; Methodology, C.-M.M., C.V., N.K., C.C.Z. and M.-A.K.; Validation, C.-M.M., C.V., N.K., C.C.Z. and M.-A.K.; Writing—original draft, C.-M.M. and C.V.; Writing—review and editing, C.-M.M., C.V., N.K., C.C.Z. and M.-A.K. All authors have read and agreed to the published version of the manuscript.

Funding: The research leading to these results has been supported by the H2020 PALANTIR ("Practical Autonomous Cyberhealth for resilient SMEs & Microenterprises") project (no. 883335) and the H2020 SANCUS ("Analysis Software Scheme of Uniform Statistical Sampling, Audit and Defense Processes") project (no. 952672). Part of this work has been conducted within the grant agreements no. 786698 and 833673 of the European Union's Horizon 2020 research and innovation programme. The work reflects only the authors' view and the Agency is not responsible for any use that may be made of the information it contains.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The software used in this paper is described in Table 1 and links to the download locations are provided therein. The data resulting from the software security analysis are available within the SonarCloud platform, at the following links: https://sonarcloud.io/dashboard?id=boa (accessed on 14 May 2021), https://sonarcloud.io/dashboard?id=busybox-1.31.1, https://sonarcloud.io/dashboard?id=JavaSMQ, https://sonarcloud.io/dashboard?id=jSML, https://sonarcloud.io/dashboard?id=lib60870 (accessed on 14 May 2021), https://sonarcloud.io/dashboard?id=libiec61850, https://soda.dit.uop.gr/sites/soda.dit.uop.gr/files/modbus4j.pdf, https://soda.dit.uop.gr/sites/soda.dit.uop.gr/files/pymodbus.zip, https://sonarcloud.io/dashboard?id=lighttpd-1.4.55 (accessed on 14 May 2021), https://sonarcloud.io/dashboard?id=Findorgri_MicroWebSrv2, https://sonarcloud.io/dashboard?id=MinnowServer (accessed on 14 May 2021) and https://sonarcloud.io/dashboard?id=thttpd (accessed on 14 May 2021).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Leligou, H.C.; Zahariadis, T.; Sarakis, L.; Tsampasis, E.; Voulkidis, A.; Velivassaki, T.E. Smart Grid: A demanding use case for 5G technologies. In Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Athens, Greece, 19–23 March 2018; pp. 215–220.
2. Al Ridhawi, I.; Aloqaily, M.; Boukerche, A.; Jararweh, Y. Enabling Intelligent IoCV Services at the Edge for 5G Networks and Beyond. *IEEE Trans. Intell. Transp. Syst.* **2021**. [CrossRef]
3. Otoum, S.; Al Ridhawi, I.; Mouftah, H.T. Blockchain-Supported Federated Learning for Trustworthy Vehicular Networks. In Proceedings of the GLOBECOM 2020—2020 IEEE Global Communications Conference, Taipei, Taiwan, 7–11 December 2020; pp. 1–6.
4. Ali, F.; Bouachir, O.; Ozkasap, O.; Aloqaily, M. SynergyChain: Blockchain-assisted Adaptive Cyberphysical P2P Energy Trading. *IEEE Trans. Ind. Inform.* **2021**. [CrossRef]
5. Li, S.; Da Xu, L.; Zhao, S. 5G Internet of Things: A survey. *J. Ind. Inf. Integr.* **2018**, *10*, 1–9. [CrossRef]
6. IHS Markit. *The Internet of Things: A Movement, Not a Market*; IHS Markit: London, UK, 2017.
7. Statista. *Internet of Things (IoT) Connected Devices Installed Base Worldwide from 2015 to 2025*; Statista: Hamburg, Germany, 2016.
8. Sofana, S.R.; Dragičević, T.; Siano, P.; Prabaharan, S.R.S. Future Generation 5G Wireless Networks for Smart Grid: A Comprehensive Review. *Energies* **2019**, *12*, 2140. [CrossRef]
9. Saghezchi, F.B.; Mantas, G.; Ribeiro, J.; Al-Rawi, M.; Mumtaz, S.; Rodriguez, J. Towards a secure network architecture for smart grids in 5G era. In Proceedings of the 2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC), Valencia, Spain, 26–30 June 2017; pp. 121–126.
10. Mavroeidakos, T.; Chaldeakis, V. Threat Landscape of Next Generation IoT-Enabled Smart Grids. In Proceedings of the 2020 IFIP International Conference on Artificial Intelligence Applications and Innovations, Neos Marmaras, Greece, 5–7 June 2020; pp. 116–127.
11. Venugopalan, V.; Patterson, C.D. Surveying the Hardware Trojan Threat Landscape for the Internet-of-Things. *J. Hardw. Syst. Secur.* **2018**, *2*, 131–141. [CrossRef]

12. Mathas, C.-M.; Grammatikakis, K.-P.; Vassilakis, C.; Kolokotronis, N.; Bilali, V.-G.; Kavallieros, D. Threat landscape for smart grid systems. In Proceedings of the 15th International Conference on Availability, Reliability and Security, Hamburg, Germany, 27–30 August 2018; ACM: New York, NY, USA, 2020; pp. 1–7.
13. *A Comprehensive Guide to 5G Security*; Liyanage, M.; Ahmad, I.; Abro, A.B.; Gurtov, A.; Ylianttila, M. (Eds.) John Wiley & Sons, Ltd.: Chichester, UK, 2018; ISBN 9781119293071.
14. Piqueras Jover, R.; Marojevic, V. Security and Protocol Exploit Analysis of the 5G Specifications. *IEEE Access* **2019**, *7*, 24956–24963. [CrossRef]
15. Cimpean, D.; de Quirós, P.C.B.; Gutiérrez, F.-G. *Appropriate Security Measures for Smart Grids*; ENISA: Iraklion, Greece, 2012.
16. Safaei Pour, M.; Bou-Harb, E.; Varma, K.; Neshenko, N.; Pados, D.A.; Choo, K.-K.R. Comprehending the IoT cyber threat landscape: A data dimensionality reduction technique to infer and characterize Internet-scale IoT probing campaigns. *Digit. Investig.* **2019**, *28*, S40–S49. [CrossRef]
17. Otoum, S.; Kantarci, B.; Mouftah, H. A Comparative Study of AI-based Intrusion Detection Techniques in Critical Infrastructures. *arXiv* **2020**, arXiv:2008.00088.
18. Herzberg, B.; Bekerman, D.; Zeifman, I. *Breaking Down Mirai: An IoT DDoS Botnet Analysis*; Imperva: San Mateo, CA, USA, 2016.
19. Veluz, D. *STUXNET Malware Targets SCADA System*; TrendMicro: Tokyo, Japan, 2010.
20. SOHOpelessly Broken 2.0; ISE: 2019. Available online: https://www.ise.io/casestudies/sohopelessly-broken-2-0/ (accessed on 14 May 2021).
21. *OWASP Code Review Guide v2*; OWASP: Los Angeles, CA, USA, 2017.
22. Sachidananda, V.; Bhairav, S.; Ghosh, N.; Elovici, Y. PIT: A Probe Into Internet of Things by Comprehensive Security Analysis. In Proceedings of the 2019 18th IEEE International Conference On Trust, Security And Privacy in Computing and Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), Rotorua, New Zealand, 5–8 August 2019; pp. 522–529.
23. Samtani, S.; Yu, S.; Zhu, H.; Patton, M.; Chen, H. Identifying SCADA vulnerabilities using passive and active vulnerability assessment techniques. In Proceedings of the 2016 IEEE Conference on Intelligence and Security Informatics (ISI), Tucson, AZ, USA, 28–30 September 2016; pp. 25–30.
24. Geneiatakis, D.; Kounelis, I.; Neisse, R.; Nai-Fovino, I.; Steri, G.; Baldini, G. Security and privacy issues for an IoT based smart home. In Proceedings of the 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 22–26 May 2017; pp. 1292–1297.
25. Overstreet, D.; Wimmer, H.; Haddad, R.J. Penetration Testing of the Amazon Echo Digital Voice Assistant Using a Denial-of-Service Attack. In Proceedings of the 2019 SoutheastCon, Huntsville, AL, USA, 11–14 April 2019; pp. 1–6.
26. Scully, C.; Wang, P. Router Security Penetration Testing in a Virtual Environment. *Adv. Intell. Syst. Comput.* **2018**, *558*, 119–124.
27. Alnaeli, S.M.; Sarnowski, M.; Aman, M.S.; Abdelgawad, A.; Yelamarthi, K. Vulnerable C/C++ code usage in IoT software systems. In Proceedings of the 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 12–14 December 2016; pp. 348–352.
28. Alnaeli, S.M.; Sarnowski, M.; Aman, M.S.; Abdelgawad, A.; Yelamarthi, K. Source Code Vulnerabilities in IoT Software Systems. *Adv. Sci. Technol. Eng. Syst. J.* **2017**, *2*, 1502–1507. [CrossRef]
29. Ferrara, P.; Mandal, A.K.; Cortesi, A.; Spoto, F. Static analysis for discovering IoT vulnerabilities. *Int. J. Softw. Tools Technol. Transf.* **2021**, *23*, 71–88. [CrossRef]
30. OWASP. *Internet of Things (IoT) Top 10 2018*; OWASP: Los Angeles, CA, USA, 2018.
31. OWASP. *OWASP Top 10-2017*; OWASP: Los Angeles, CA, USA, 2017.
32. Medeiros, I.; Neves, N.F.; Correia, M. Securing energy metering software with automatic source code correction. In Proceedings of the 2013 11th IEEE International Conference on Industrial Informatics (INDIN), Bochum, Germany, 29–31 July 2013; pp. 701–706.
33. Sachidananda, V.; Bhairav, S.; Elovici, Y. OVER. In Proceedings of the 35th Annual ACM Symposium on Applied Computing, Brno, Czech Republic, 30 March–3 April 2020; ACM: New York, NY, USA, 2020; pp. 729–738.
34. Späth, J. The Myth of False Positives in Static Application Security Testing; Medium.com: 2020. Available online: https://medium.com/all-about-modern-application-security-testing-a/the-myth-of-false-positives-of-static-application-security-testing-41ab5fad0f74 (accessed on 15 April 2021).
35. MITRE. CWE VIEW: Software Development. 2021. Available online: https://cwe.mitre.org/data/definitions/699.html (accessed on 2 March 2021).
36. MITRE. Secure Code Review. 2018. Available online: https://www.mitre.org/publications/systems-engineering-guide/enterprise-engineering/systems-engineering-for-mission-assurance/secure-code-review (accessed on 16 April 2021).
37. Arbettu, R.K.; Khondoker, R.; Bayarou, K.; Weber, F. Security analysis of OpenDaylight, ONOS, Rosemary and Ryu SDN controllers. In Proceedings of the 2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks), Montreal, QC, Canada, 26–28 September 2016; pp. 37–44.

38. Hanna, S.; Rolles, R.; Molina-Markham, A.; Poosankam, P.; Fu, K.; Song, D. Take two software updates and see me in the morning: The case for software security evaluations of medical devices. In Proceedings of the 2nd USENIX Conference on Health Security and Privacy, San Francisco, CA, USA, 9 August 2011; Available online: https://dl.acm.org/doi/10.5555/2028026.2028032 (accessed on 14 May 2021).
39. Wisy, M. *Smart Message Language 1.03*; EMSYCON GmbH: Berlin, Germany, 2008.
40. *ISO/IEC JTC 1/SC 6 ISO/IEC 8825-1:2015 Information Technology—ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, 5th ed.; ISO: Geneva, Switzerland, 2015.
41. International Electrotechnical Commission. *Telecontrol Equipment and Systems—Part 5-101: Transmission Protocols—Companion Standard for Basic Telecontrol Tasks (IEC 60870-5-101)*; 2.1.; IEC: Geneva, Switzerland, 2015.
42. International Electrotechnical Commission. *Telecontrol Equipment and Systems—Part 5-104: Transmission Protocols—Network Access for IEC 60870-5-101 Using Standard Transport Profiles*; 2.1.; IEC: Geneva, Switzerland, 2016.
43. ISO. *ISO 9506-1:2003 Industrial Automation Systems—Manufacturing Message Specification—Part 1: Service Definition*; ISO: Geneva, Switzerland, 2003.
44. International Electrotechnical Commission. *Communication Networks and Systems in Substations—Specific Communication Service Mapping (SCSM) Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3*; IEC: Geneva, Switzerland, 2003.
45. El Hariri, M.; Harmon, E.; Youssef, T.; Saleh, M.; Habib, H.; Mohammed, O. The IEC 61850 Sampled Measured Values Protocol: Analysis, Threat Identification, and Feasibility of Using NN Forecasters to Detect Spoofed Packets. *Energies* **2019**, *12*, 3731. [CrossRef]
46. RealTime Logic. SimpleMQ Secure IoT Protocol. 2021. Available online: https://realtimelogic.com/products/simplemq/ (accessed on 2 March 2021).
47. MQTT. MQTT: The Standard for IoT Messaging. 2021. Available online: https://mqtt.org/ (accessed on 2 March 2021).
48. Modbus Organization. Modbus Application Protocol Specification V1.1b3. 2012. Available online: https://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (accessed on 15 June 2020).
49. Mozilla Developer Network. *The WebSocket API (WebSockets)*; Mozilla: San Francisco, CA, USA, 2021.
50. Fink, G.; Flatow, I. Introducing Single Page Applications. In *Pro Single Page Application Development*; Apress: Berkeley, CA, USA, 2014; pp. 3–13.
51. Nelson, J. An Overview of the Boa Web Server. 2002. Available online: https://www.osnews.com/story/2217/an-overview-of-the-boa-web-server/ (accessed on 2 March 2021).
52. Doolittle, L.; Nelson, J. Boa Webserver News. 2005. Available online: http://www.boa.org/news.html (accessed on 2 March 2021).
53. Software Harm Reduction Guild (SHRuG). Boa 0.94.13. 2020. Available online: https://github.com/shrugly/boa-0.94.13 (accessed on 15 June 2020).
54. ACME. Thttpd-Tiny/Turbo/Throttling HTTP Server. 2021. Available online: https://acme.com/software/thttpd/ (accessed on 2 March 2021).
55. Poskanzer, J. Thttpd Man Page. 2000. Available online: http://www.acme.com/software/thttpd/thttpd_man.html (accessed on 2 March 2021).
56. Damien, G. MicroPython. 2021. Available online: https://micropython.org/ (accessed on 2 March 2021).
57. Pycom. 2021. Available online: https://pycom.io/ (accessed on 3 March 2021).
58. ESP32.net. The Internet of Things with ESP32. 2021. Available online: http://esp32.net/ (accessed on 2 March 2021).
59. STMicroelectronics. STM32 High Performance MCUs. 2021. Available online: https://www.st.com/en/microcontrollers-microprocessors/stm32-high-performance-mcus.html (accessed on 2 March 2021).
60. Wells, N. BusyBox: A Swiss Army Knife for Linux. *Linux J.* **2000**, *78es*, 10es.
61. Andersen, E. BusyBox: The Swiss Army Knife of Embedded Linux. 2008. Available online: https://busybox.net/about.html (accessed on 2 March 2021).
62. Jones, T.M. BusyBox Simplifies Embedded Linux Systems. 2006. Available online: www.ibm.com/developerworks/linux/library/l-busybox/ (accessed on 3 March 2021).
63. BusyBox. BusyBox–About. 2021. Available online: https://busybox.net/ (accessed on 3 March 2021).
64. MITF. Honeypot Support for IoT Devices. *Internet Infrastruct. Rev.* **2017**, *36*. Available online: https://www.iij.ad.jp/en/dev/iir/pdf/iir_vol36_focused1_EN.pdf (accessed on 2 March 2021).
65. MITRE. CWE-295: Improper Certificate Validation. 2021. Available online: cwe.mitre.org/data/definitions/295.html (accessed on 3 March 2021).
66. Fahl, S.; Harbach, M.; Muders, T.; Smith, M.; Baumgärtner, L.; Freisleben, B. Why eve and mallory love android. In Proceedings of the 2012 ACM Conference on Computer and Communications Security—CCS '12, Raleigh, NC, USA, 16–18 October 2012; ACM Press: New York, NY, USA, 2012; p. 50.
67. Mullen, G.; Meany, L. Assessment of Buffer Overflow Based Attacks On an IoT Operating System. In Proceedings of the 2019 Global IoT Summit (GIoTS), Aarhus, Denmark, 17–21 June 2019; pp. 1–6.
68. Veracode. What Is a Buffer Overflow? *Learn About Buffer Overrun Vulnerabilities, Exploits & Attacks.* Available online: https://www.veracode.com/security/buffer-overflow (accessed on 2 March 2021).
69. Cobb, M. Buffer Overflow. 2016. Available online: searchsecurity.techtarget.com/definition/buffer-overflow (accessed on 2 March 2021).

70. ENISA. Buffer Overflow. 2021. Available online: www.enisa.europa.eu/topics/csirts-in-europe/glossary/buffer-overflow (accessed on 3 March 2021).
71. MITRE. CWE-676: Use of Potentially Dangerous Function. 2021. Available online: https://cwe.mitre.org/data/definitions/676.html (accessed on 3 March 2021).
72. MITRE. CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer. 2021. Available online: https://cwe.mitre.org/data/definitions/119.html (accessed on 3 March 2021).
73. MITRE. CWE-120: Buffer Copy without Checking Size of Input. 2021. Available online: https://cwe.mitre.org/data/definitions/120.html (accessed on 3 March 2021).
74. OWASP. *Protect Data Everywhere*; OWASP: Los Angeles, CA, USA, 2018.
75. MITRE. CWE-326: Inadequate Encryption Strength. 2021. Available online: https://cwe.mitre.org/data/definitions/326.html (accessed on 3 March 2021).
76. MITRE. CWE-327: Use of a Broken or Risky Cryptographic Algorithm. 2021. Available online: https://cwe.mitre.org/data/definitions/327.html (accessed on 3 March 2021).
77. MITRE. CWE-330: Use of Insufficiently Random Values. 2021. Available online: https://cwe.mitre.org/data/definitions/330.html (accessed on 3 March 2021).
78. MITRE. CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG). 2021. Available online: https://cwe.mitre.org/data/definitions/338.html (accessed on 3 March 2021).
79. MITRE. *CWE-489: Active Debug Code*; MITRE: Bedford, MA, USA, 2021.
80. MITRE. CWE-200: Exposure of Sensitive Information to an Unauthorized Actor. 2021. Available online: https://cwe.mitre.org/data/definitions/200.html (accessed on 3 March 2021).
81. MITRE. *CWE-215: Insertion of Sensitive Information Into Debugging Code*; MITRE: Bedford, MA, USA, 2021.
82. MITRE. CWE-319: Cleartext Transmission of Sensitive Information. 2021. Available online: https://cwe.mitre.org/data/definitions/319.html (accessed on 3 March 2021).
83. 83. Baeldung. In *What Is a Race Condition?* Baeldug: Bucharest, Romania, 2020.
84. Veracode. *What Is a Race Condition?* Veracode: Burlington, MA, USA, 2021.
85. MITRE. CWE-367: Time-Of-Check Time-Of-Use (TOCTOU) Race Condition. 2021. Available online: https://cwe.mitre.org/data/definitions/367.html (accessed on 3 March 2021).
86. OWASP. Broken Access Control. 2021. Available online: owasp.org/www-community/Broken_Access_Control (accessed on 2 March 2021).
87. PacketLabs. Broken Access Control: Hidden Exposure for Sensitive Data. 2020. Available online: www.packetlabs.net/broken-access-control/ (accessed on 2 March 2021).
88. KumarShrestha, A.; Singh Maharjan, P.; Paudel, S. Identification and Illustration of Insecure Direct Object References and their Countermeasures. *Int. J. Comput. Appl.* **2015**, *114*, 39–44. [CrossRef]
89. Spiliotopoulos, D.; Kotis, K.; Vassilakis, C.; Margaris, D. Semantics-Driven Conversational Interfaces for Museum Chatbots. In *Culture and Computing*; Rauterberg, M., Ed.; Springer: Cham, Switzerland, 2020; Volume 12215, pp. 255–266; ISBN 978-3-030-50266-9.
90. MITRE. CWE-732: Incorrect Permission Assignment for Critical Resource. 2021. Available online: https://cwe.mitre.org/data/definitions/732.html (accessed on 3 March 2021).
91. Vazquez, A. Linux File System and Share and Service Permissions. In *Practical LPIC-3 300*; Apress: Berkeley, CA, USA, 2019; pp. 293–315.
92. Wang, Y.; Wang, L.; Yu, T.; Zhao, J.; Li, X. Automatic detection and validation of race conditions in interrupt-driven embedded software. In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, Santa Barbara, CA, USA, 10–14 July 2017; ACM: New York, NY, USA, 2017; pp. 113–124.
93. Sotiropolos, P.; Vassilakis, C. Detection of intermittent faults in software programs through identification of suspicious shared variable access patterns. *J. Syst. Softw.* **2020**, *159*, 110455. [CrossRef]
94. Spalazzi, L.; Spegni, F.; Liva, G.; Pinzger, M. Towards Model Checking Security of Real Time Java Software. In Proceedings of the 2018 International Conference on High Performance Computing & Simulation (HPCS), Orléans, France, 16–20 July 2018; pp. 642–649.
95. Engler, D.; Ashcraft, K. RacerX. *ACM SIGOPS Oper. Syst. Rev.* **2003**, *37*, 237–252. [CrossRef]
96. Cremers, C.J.F. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Computer Aided Verification*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 414–418.
97. Meier, S.; Cremers, C.; Basin, D. Strong Invariants for the Efficient Construction of Machine-Checked Protocol Security Proofs. In Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium, Edinburgh, UK, 17–19 July 2010; pp. 231–245.
98. Schmidt, B.; Meier, S.; Cremers, C.; Basin, D. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In Proceedings of the 2012 IEEE 25th Computer Security Foundations Symposium, Cambridge, MA, USA, 25–27 June 2012; pp. 78–94.
99. Reda, H.T.; Ray, B.; Peidaee, P.; Anwar, A.; Mahmood, A.; Kalam, A.; Islam, N. Vulnerability and Impact Analysis of the IEC 61850 GOOSE Protocol in the Smart Grid. *Sensors* **2021**, *21*, 1554. [CrossRef] [PubMed]

100. Tu, T.; Zhang, H.; Qin, B.; Chen, Z. A Vulnerability Mining System Based on Fuzzing for IEC 61850 Protocol. In Proceedings of the 2017 5th International Conference on Frontiers of Manufacturing Science and Measuring Technology (FMSMT 2017), Taiyuan, China, 24–25 June 2017; Atlantis Press: Paris, France, 2017.
101. Nardone, R.; Rodriguez, R.J.; Marrone, S. Formal security assessment of Modbus protocol. In Proceedings of the 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST), Barcelona, Spain, 5–7 December 2016; pp. 142–147.
102. Parian, C.; Guldimann, T.; Bhatia, S. Fooling the Master: Exploiting Weaknesses in the Modbus Protocol. *Procedia Comput. Sci.* **2020**, *171*, 2453–2458. [CrossRef]
103. Scapy Scapy Project. 2021. Available online: https://scapy.net/ (accessed on 3 May 2021).
104. Chen, B.; Pattanaik, N.; Goulart, A.; Butler-purry, K.L.; Kundur, D. Implementing attacks for modbus/TCP protocol in a real-time cyber physical system test bed. In Proceedings of the 2015 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR), Charleston, SC, USA, 11–14 May 2015; pp. 1–6.
105. Burrows, M.; Abadi, M.; Needham, R. A logic of authentication. *ACM Trans. Comput. Syst.* **1990**, *8*, 18–36. [CrossRef]
106. Chen, X.; Deng, H. Efficient Verification of Cryptographic Protocols with Dynamic Epistemic Logic. *Appl. Sci.* **2020**, *10*,6577. [CrossRef]
107. Razaque, A.; Aloqaily, M.; Almiani, M.; Jararweh, Y.; Srivastava, G. Efficient and reliable forensics using intelligent edge computing. *Futur. Gener. Comput. Syst.* **2021**, *118*, 230–239. [CrossRef]

Short Biography of Author

Charilaos C. Zarakovitis Received the BSc degree from the Technical University of Crete Greece in 2003, the M.Sc and G.C.Eng degrees from the Dublin Institute of Technology Ireland in 2004 and 2005, and the M.Phil and Ph.D degrees from Brunel University UK in 2006 and 2012, respectively, all in electronic engineering. In Academia, he has been Senior Researcher at Infolab21 LANCASTER University UK, 5GIC University of SURREY UK, and MNLaboratory NCSR "DEMOKRITOS" Greece. In Industry, he has been Research Engineer at AXON LOGIC P.C. Greece, MOTOROLA UK, INTRACOM Greece, and NOKIA-SIEMENS-NETWORKS UK. His research includes quantum neural networks, green communications, cybersecurity optimization, among others.