IoT Mobile Applications Pentesting Methodology and Results of Research

Jan Adamski, Marek Janiszewski, and Marcin Rytel

***Abstract*—This article describes the methodology of pentesting mobile applications used to control Internet of Things (IoT) devices, which is part of a comprehensive methodology for testing the security of IoT solutions. This article shows the importance of mobile applications in the context of IoT security and what elements constitute the essential pillars for the security of users and their devices. The authors describe the methods of the pentesting process step by step depending on the application being tested and the complications encountered. This article also includes the most popular security flaws of this type of application, along with a description of how they pose a threat to their users. It also shows the complexity of conducting pentests of IoT solutions due to the variety of procedures and test areas that must be intertwined in order to obtain reliable and complete results. This article concludes with a description of vulnerabilities discovered and reported to the relevant authorities thanks to research conducted in accordance with the described methodology.**

***Index Terms*—Internet of Things (IoT) pentesting, IoT security, mobile apps pentesting.**

I. Introduction

CAMBRIDGE English Dictionary defines the Internet of Things (IoT) as a network of interrelated devices that connect and exchange data with other IoT devices and the cloud [1]. However, the world of IoT is evolving, and the above definition is not the only accepted one. For this reason, we have adopted the following definition for our research: "IoT device—an item (except a phone, PC, tablet, and data centre hardware) equipped with network connectivity and the ability to collect and exchange data" [2]. It refines the above definition and more closely reflects the range of devices we are considering.

According to researchers at IoT Analytics, for the first time in 2020, there were more IoT connections (e.g., connected cars, smart home devices, and connected industrial equipment) than non-IoT connections (smartphones, laptops, and computers) [3], which shows the scale of the devices to deal with. By 2030, there will be nearly 30 billion IoT devices or, on average, almost four IoT devices per person [4]. This growth has made the security of IoT devices a significant concern.

Ensuring the security of IoT is paramount, given that these devices often manage critical systems in homes and institutions and process sensitive data. The scope of IoT security extends far beyond the IoT itself to include a variety of methodologies not necessarily related to IoT. A distinctive feature of IoT devices is the ability to connect them to complex systems and control them from a great distance using a smartphone or computer. The smart home allows for remote control of lighting, heating, air conditioning, and household appliances. Although making life easier, such capabilities are associated with several risks that would not be present in solutions unrelated to the IoT. Vulnerabilities in IoT system security and a lack of updates lead to data breaches, making it necessary to address the security of these devices if we want to protect personal data and critical infrastructure from potentially malicious actors.

The simplest example illustrating the dangers of using an IoT device is a smart bulb, which, just like a regular bulb, must have standard safety and durability certificates, such as heat resistance or average life expectancy but also meet Internet security standards. None of the customers would accept if an intruder could remotely access their bulb and be able to blink it from outside. Moreover, gaining access to an IoT device also involves the danger of accessing other devices in the household connected to the same network. Thus, for example, through faulty network security, an unauthorised person could access the internal network and download the contents of an NAS drive with private data, such as photos.

Going forward, the security of IoT devices is highly dependent on network traffic and manufacturer APIs. More and more IoT devices not only support control via a mobile app but also require it to function. This trend means that the mobile app itself is becoming an essential element in terms of the security of the entire solution.

Security testing of IoT control applications differs from pentesting regular applications, such as news reading or weather forecasts. In classic applications, the most important thing is the confidentiality of user credentials, their private data, and appropriate remote access security. Particularly helpful in this case are constantly evolving new SDK (software development kit) distributions that are constantly being developed, which, with each successive release, help more and more in managing permissions and access restrictions, minimizing the risk of vulnerabilities. In the case of IoT applications, on the other hand, a considerable burden is placed on the security of

Received 21 November 2024; revised 7 January 2025; accepted 5 February 2025. Date of publication 10 February 2025; date of current version 23 May 2025. This work was supported by the project: "Laboratory for Vulnerability Testing of Desktop and Mobile Computing Devices and Algorithms and Software (LAVA)" were co-financed by the National Centre for Research and Development through the CyberSecIdent Program "Cybersecurity and Identity" under Grant CYBERSECIDENT/488240/IV/NCBR/2021. *(Corresponding author: Jan Adamski.)*

The authors are with the Cybersecurity Department, NASK—National Research Institute, 01-045 Warsaw, Poland (e-mail: jan.adamski@nask.pl).

Digital Object Identifier 10.1109/JIOT.2025.3540305

network traffic, cryptography that secures control messages, the way data is stored, as well as security mechanisms that will protect users' devices in the event of, for example, credential leakage. Application security is responsible for the security of the entire IoT solution, such as a modern smart home whose owner can turn off the alarm, open the gate or garage door using the app, and control any IoT device on their property.

IoT devices are often smaller, more constrained, and more specialized than traditional computing devices, making them more challenging to test. This is due to several reasons, including the limited capabilities of the operating systems used in the devices, low computing power, often proprietary nonstandard communication protocols or a narrow range of options for connecting directly to the device. These aspects mainly affect closed box testing, i.e., tests where the internal structures or implementations are unknown to the tester. As mentioned, in the context of security testing of IoT devices, a particular challenge is analyzing the communication protocol, which can be very difficult, complex, and time-consuming without the appropriate tools and documentation. It is worth remembering that the security and unambiguity of communication in this type of device are crucial to ensure the security of solutions. Therefore, thoroughly testing the security of IoT mobile applications is vital. Pentesting IoT mobile applications is a security assessment focusing on IoT devices and systems. It aims to identify vulnerabilities and weaknesses in ecosystems that could be exploited. It involves several activities that need to be performed to ensure the security of IoT device users.

In this article, the authors schematically present the security testing issues of IoT devices, the prepared methodology for testing the security of IoT Android OS applications, discuss the unique challenges of securing IoT devices, and compare it to other published work. The remainder of this article presents the vulnerabilities found through the testing using the pentesting methodology, confirming its effectiveness. Section II describes the state-of-the-art and overview of work related to penetration testing of mobile applications, Section III provides an overview of the IoT pentesting methodology, while Section IV discusses the methodology for security testing of IoT mobile applications. Section V discusses the results and provides an overview of the tests conducted, while Section VI writes down the conclusions and indicates further research plans for the authors of the work on the security of mobile applications of IoT devices.

II. State-of-the Art/Related Work

The review of mobile vulnerabilities by [5] mentions penetration testing as one of the mobile vulnerability assessment techniques. A broad study of mobile security was done by [6] in 2020, encompassing, besides the app security, the security of mobile devices themselves and surveying their users to measure the real-life adoption of best mobile security practices. Large-scale automated testing of 455 IoT mobile apps was performed by [7], while [8] performed in-depth analysis of 41 such applications, both showing that the current state of their security is decent, but many improvements can still be made to improve it, e.g., by reducing required permissions, removing privacy violating trackers and by paying more attention to the security by the app developers. The subject of permissions is further explored in [9] analyzing 25 000 applications, including 5 773 malware apps, and proposing a risk assessment model based on the permissions requested.

The widely recognized OWASP IoT Project [10], last updated in 2019, serves as a valuable resource. The IoT project from OWASP consists of smaller parts presenting the following topics. OWASP IoT Top 10 - ten most common IoT vulnerabilities with mapping, IoTGoat - purposely insecure firmware designed to help developers learn to test common IoT device vulnerabilities, Firmware Analysis Project - testing guidance for the device firmware attack surface, Firmware Security Testing Methodology - methodology of device firmware security testing, IoT Security Verification Standard - community effort to establish an open standard of security requirements for IoT ecosystems and ByteSweep - free software for IoT firmware security analysis. As presented, the project largely focuses on the research area of IoT devices, especially its firmware, but falls short in providing a comprehensive methodology for testing the security of IoT mobile apps, mentioning this layer only in the Top 10 project and redirecting users to the general OWASP Mobile Application Security Testing Guide - MASTG [11]. The general MASTG does not focus on areas particularly relevant to the study of apps controlling IoT devices. However, it contains extensive theoretical knowledge enabling an understanding of the security mechanisms used in the Android system and examples of faulty implementations of some functions and libraries. The document itself is quite long and difficult to use, and the examples are not set in the world of IoT devices. The methodology proposed by the authors of this article condenses important theoretical knowledge and shows a practical way to test real IoT apps. The last OWASP document relevant to our research is the OWASP Mobile Top 10 [12], updated in 2024. Similarly to previous examples, it is focused on the general area of its interest and is not a sufficient indicator of the most important vulnerabilities affecting the applications operating at the interface of the IoT and Mobile fields. Nevertheless, the 2024 update brings some issues affecting IoT particularly strongly to the spotlight, and puts them in the top two places, namely M1: improper credential usage and M2: inadequate supply chain security, which were both absent from the previous OWASP Mobile Top 10 version.

An alternative to OWASP materials is a methodology prepared by the HackTricks community [13] called android applications pentesting. However, like the mentioned MASTG, it applies to security testing of all types of applications without focusing on IoT-specific needs. While this methodology can help find vulnerabilities in the application itself, it does not cover testing the security of the entire IoT solution. When following this approach, potential attack vectors in the IoT ecosystem, such as encryption of Bluetooth messages controlling the device, may be overlooked or not assessed with sufficient accuracy.

The authors of the more recent work "PatrIoT: practical and agile threat research for IoT" [14], also draw attention to the

threats posed by the area of mobile apps, but they omit the issue of conducting research on the applications themselves, detailing only the self-developed potential security flaws. The authors later call them mobile weaknesses. Undoubtedly, the PatrIoT methodology has allowed researchers to discover many vulnerabilities in IoT devices. This is evidenced by an article [15], describing the security flaws thanks to the discussed methodology of testing. However, the vulnerabilities detected in the research were not found in mobile apps but in Web applications, APIs or IoT devices themselves. Some weaknesses listed in PatrIoT are similar to the flaws mentioned in this article, but unlike the research conducted by the authors of the PatrIoT methodology, the methodology described in this article managed to discover several irregularities in the tested applications.

The described deficiencies have spurred the need for targeted methodology PMIoT that specifically addresses the security intricacies of IoT mobile apps. Although the methodologies listed in this section are undoubtedly good materials on the basis of which it is possible to conduct pentests of IoT devices, each of them has certain disadvantages in the context of mobile application testing, which were mentioned earlier. This article introduces a methodology designed to fill the current gaps by focusing on the security aspects most crucial for IoT solutions. In doing so, it responds to the evolving landscape of interconnected devices, providing a more nuanced approach to pentesting in the realm of IoT mobile apps.

It is crucial to emphasize that while testing mobile apps used to control or communicate with IoT devices, we cannot use the methodologies of testing mobile apps in isolation from the context nor test just the mobile app on its own. By treating the application as a separate entity, the opportunity to conduct comprehensive testing of the IoT solution is lost, omitting, for example, tests of security measures used in the control instructions exchanged between the application and the IoT device. There is a need to consider communication with the device (directly through Bluetooth or Wi-Fi) and with the cloud through an IP network.

III. Overall Methodology of IoT Pentesting

To conduct comprehensive research on the IoT devices, possible attack scenarios were analyzed, and the issue was divided into several test areas. These were specified as follows: external clouds and servers, mobile apps and user interfaces, Web applications and management interfaces, IP networks, radio interfaces, firmware, and hardware. All areas with common points and highlighted mobile testing area are shown in Fig. 1. More information about the overall methodology, as well as dedicated methodologies for the other areas, will be provided in future publications of our research group.

It may seem that security testing of IoT mobile apps does not differ significantly from security testing performed on other apps, such as games or tools. However, the main areas around which the tests will focus are changing. From the perspective of both users and IoT mobile apps vendors, the most important thing is the security of control over devices

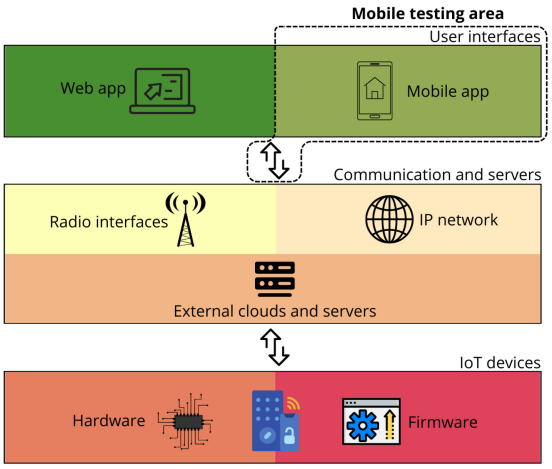

Fig. 1. Mobile IoT security testing area.

connected to the account. Securing secondary functions, such as micropayments or account management configuration files, is not as important as in the case of other types of apps. From the perspective of an IoT solution, the crucial aspect is that hackers should not be able to gain unauthorized access to user accounts or control IoT devices externally, thereby exposing users to danger. For example, in modern alarm systems, manufacturers provide the ability to arm and disarm the alarm with a few clicks in the app. If it turns out that hackers can deactivate such a system or, for example, open a garage door, such solutions will defy their basic functions. The app's resistance to potential hacker attacks that require the victim's interaction, such as clicking on a malicious link or installing a malicious app, should also be examined. Testing mobile apps is also an introduction to or part of API testing because the network traffic analysis stage is the best moment to obtain the exchanged control requests. On this basis, a pentester can then conduct research on access control and client authentication on servers, which are equally important in the context of the security of IoT solutions. To summarize, the most important areas of IoT mobile app testing focus on communication between the mobile app and servers and IoT devices, as well as the resistance of these apps to attacks aimed at the user.

IV. Methodology of IoT Mobile Apps Pentesting

The methodology of testing IoT mobile apps includes parts dedicated to the network communication area. Attack scenarios encompass a wide range of possibilities, including sniffing of network traffic, prediction, or using brute force methods to gain access and compromise specific or random devices from a given manufacturer. The tests also cover aspects of phishing attacks and the quality of used cryptographic functions. The suggested cycle for conducting a comprehensive security test of IoT mobile apps is presented in Fig. 2.

As shown in Fig. 1, the security of an IoT solution consists of several areas, and despite the presented division, during complete research, they should be treated as one large, interdependent area. For example, network traffic from an IoT

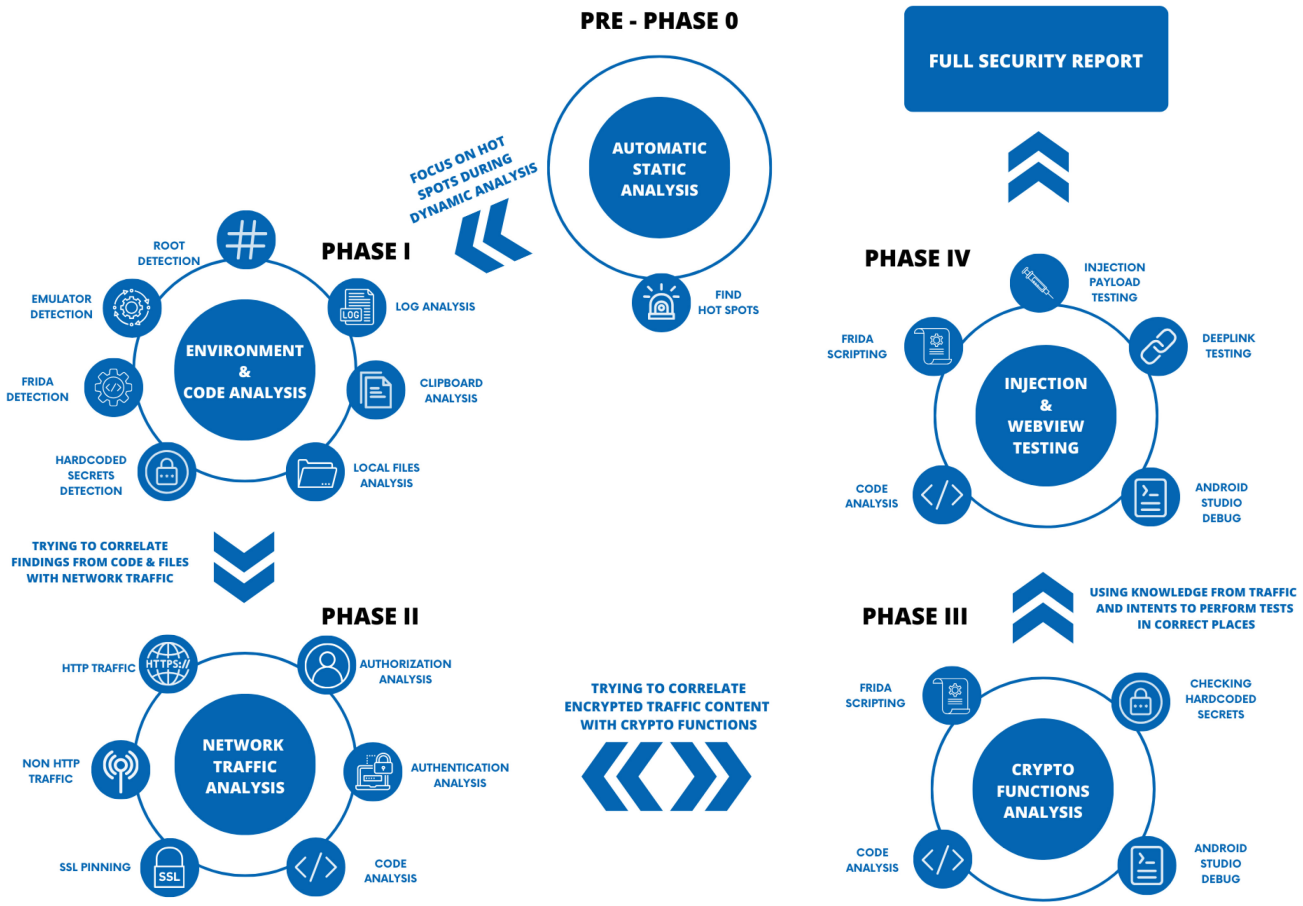

Fig. 2. Comprehensive security test of IoT mobile apps.

mobile app may be the key to deciphering the traffic between an IoT device and a server. The security testing cycle of IoT mobile apps has been divided into five main interrelated parts. The following phases have been identified: automatic static analysis, analysis of the environment and code, analysis of network traffic, analysis of cryptographic functions, analysis of injection possibilities, and WebView (a component of Android SDK that allows mobile apps to display Web content) activities. The division was made according to the tested areas, the type of tests, and undertaken testing activities. The whole process concludes with preparing a comprehensive report on the conducted research. Its content highlights encountered security issues and includes suggestions for improving the app's security level.

A. Phase 0—Automatic Static Analysis

Security examination starts with the prephase, which involves performing automatic tests to identify potentially vulnerable areas of the app, called Hot Spots. Those points will require special attention during dynamic testing. Automatic test results report also found possible hardcoded secrets, URLs, database links, domain names, etc. Similarly to designated Hot Spots, assessing and verifying the relevance of data identified during automatic analysis is necessary. The scanner used and recommended for conducting automated tests is the open-source MobSF scanner [16]. The scanner is regularly updated and supplemented with new types of tests and interpretations of mechanisms used in apps. The authors also utilized standalone tools that focus specifically on analyzing code for stored credentials or links. While these tools were useful, they did not provide as detailed or comprehensive information as MobSF, making them less effective for in-depth investigations. Certainly, the MobSF scanner is a trustworthy tool that provides valuable research results while requiring almost no additional effort on the tester's part.

B. Phase I—Environment and Code Analysis

After completing the automated testing procedures, it is time to move on to Phase I—environment and code analysis. During this phase, the tests combine both types of analysis—manual static analysis and dynamic analysis with the results of automatic examination. The main difficulty is to combine knowledge from all procedures and try to correlate it with possible vulnerability points.

It is suggested to start the examination by searching for root, emulator, and Frida [17] detection mechanisms (root detection, emulator detection, Frida detection). The tests should encompass both an empirical approach involving examining the app's behavior in various environments and a static analysis of the code to identify the presence of such mechanisms. Upon collecting the results, the next step involves searching the code for any sensitive data stored within it (hardcoded secrets

detection). It is hard to imagine a greater security issue than the possibility of reading the password, which allows access to the global database. In modern apps, thanks to the multitude of available data storage solutions, this test rarely shows useful results, but it still has a significant impact on app and user security.

The last part of the first phase is to start Dynamic Analysis with a dump of logs, clipboard and local files. During the tests, the tester should simulate everyday use of the app and perform potentially interesting actions in the scope of security and operations performed by the app (log analysis, clipboard analysis, local files analysis). This includes the procedure of registering the user, logging in, adding and controlling devices, etc. After completing the test, it is time to look through the dump results and check the correlation between actions performed in the app, logs and files. All suspicious findings should be stored for the next phase of the examination. An example of such information could be an unencrypted text file containing a username:password pair, or a cloud authorization key.

C. Phase II—Network Traffic Analysis

The second testing phase is based on examining network communication, data exchange, and authorization on servers. Verifying the connections between the mentioned parameters and the data saved in files and logs is very important to conduct comprehensive research and determine how much information is necessary to take over or brute force in order to gain access to the victim's account or their devices. The other important mechanisms are access control systems, which are crucial because they prevent unauthorized access to certain resources. These resources could range from full access to the vendor's cloud, potentially allowing the hijacking of all vendor-sold devices, to more local impacts, such as the possibility to control a device with a fixed ID. Tests should cover the full spectrum of potential attacks, including attacks that use valid user tokens for control queries directed to devices assigned to other accounts.

Phase II tests should start with an HTTP traffic interception during basic activities carried out in the app. Operations should include activities potentially attractive to attackers, such as logging in, removing the device from the account, adding the device to another account, disarming the alarm, etc. (HTTP Traffic). The app may be equipped with SSL Pinning and traffic will not be available by default, therefore, the tester should perform SSL Pinning protection deactivation procedures (SSL Pinning, Code analysis). SSL pinning is a security technique that developers use to ensure that the app only trusts a specific SSL certificate, preventing man-in-the-middle (MITM) attacks by verifying the server's identity during communication. If the app is online and the traffic dump still lacks important requests, the tester should perform an NON HTTP traffic analysis (NON HTTP Traffic), as IoT devices often use other protocols like MQTT [18], which is a lightweight messaging protocol, or CoAP [19], which is designed for efficient communication in resource-limited environments. This type of testing is also recommended even

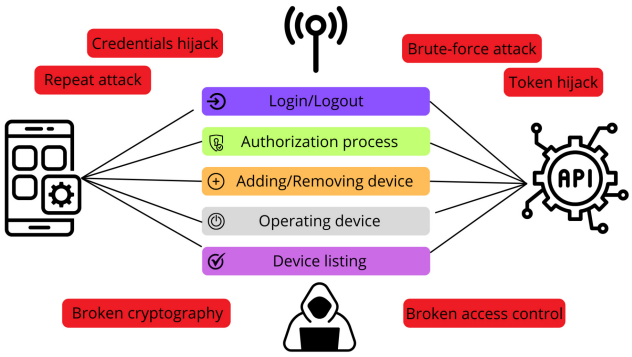

Fig. 3. Web traffic analysing test guide.

if the traffic was captured correctly in the previous tests. The results may identify additional areas for future testing, such as protocol tests, comparisons of encrypted data with that from Phase 0, and tests of endpoints identified during network analysis. Finally, when all interesting network traffic is available for inspection, it is time to perform API security tests, including mobile app authorization and authentication processes (Authorization analysis, Authentication analysis). If the tester can see the body of requests but cannot understand them due to additional encryption, it is suggested to move to Phase III and then return to Phase II with decrypted traffic. Based on the acquired knowledge and examination of communication details, the tester should prepare scenarios and verify the possibility of carrying out various types of attacks on devices or user accounts. The most important procedures and attack methods are presented in Fig. 3.

During this phase, it is essential to check if there is any possible correlation between traffic and suspicious findings from Phase I. If the app uses multifactor authentication (MFA), the tester should also test the security of its implementation (persistence of expected code in network traffic, validation based on response status, etc.). After performing the mentioned tests, the pentester should move to Phase III.

D. Phase III—Cryptographic Functions—Analysis

The list of tests to be performed depends heavily on the results of the tests conducted in Phase II. In this part, the tester is asked to examine the relevant cryptographic functions. Appropriate tests must be performed depending on the app procedures in which such functions are used.

Cryptography tests are critical in the context of additionally encrypted network traffic, procedures for operating and managing IoT devices, as well as storing data in the mobile device's memory. It sometimes happens that cryptographic operations performed on static parameters specified by the manufacturer, such as device ID or user number, are the only ones protecting against unauthorized access to customer resources and impersonation. Based on the security value sent in the query, the API validates incoming requests. The possibility of generating such a parameter based on knowledge of the structure of the original text that was later encrypted or hashed means there is a risk of an effective takeover attack, especially if this is the only security measure in this area.

The second important area is storing data in a space available to other apps. If it is necessary to save credentials or other sensitive data in this space, they should be protected with appropriate encryption. An example vulnerability is performing symmetric file encryption operations in the app using a global key hardcoded in the app's source code. If attackers find the key, it will be possible to prepare a malicious app that searches for files stored in a specific directory, sends them to the server and then mass decrypts and gains access to sensitive data. Key tools for cryptography research are code analysis (Code analysis), Frida scripts (Frida scripting), Smali plug-in for the Android Studio IDE [20] (Android Studio Debug), and multiple hardcoded secrets detectors (checking hardcoded secrets). The Frida solution allows monitoring and intercepting system calls, disassembling code, changing variable values and injecting program fragments into the app, which is very helpful in the context of searching and analyzing operations on strings. Preparing an appropriate script, for example, will allow reading the input of the encryption function, the result of which will be visible in the app or network traffic. Familiarizing with the input of such a function gives the opportunity to analyze the text before the encryption operation. Depending on the design of the function, it may also be possible to read the key. Combined with source code analysis (Code analysis), they constitute a comprehensive method for examining cryptographic operations. Sometimes, due to obfuscation and the high level of complexity of the app, the tester is unable to find appropriate functions in the code. In this case, the tester should debug the app using the Smali-code add-on for Android Studio IDE. For this purpose, the decompiled app should be loaded into the IDE as a project and then debugged step by step using watches and breakpoints. It should be noted that this method requires high computing power and uses a lot of RAM, so in the case of complex apps, this method may not be possible.

E. Phase IV—Injection and WebView Analysis

The fourth phase is slightly more independent than the previous ones because the research will mainly concern areas related to the IoT mobile app itself. Based on past interactions with the app, the tester must select appropriate functions, views and intents in which he will conduct tests. In this phase, the pentests should examine the possibility of carrying out various types of injection attacks and exclude or confirm the possibility of performing any dangerous actions using deeplinks, receivers and exported activities. Mentioned deeplinks are hyperlinks guiding users directly to particular sections or features in the app.

Injection tests should be performed on all activities that may be potentially susceptible and return any useful information. In practice, such tests are performed on activities that collect any data from the user. It is worth noting that the data does not always have to be in text form. It can be, for example, a malicious QR code (Code analysis, Injection payload testing). If this data is later sent to the server, to simplify testing, the tester can continue testing directly by modifying the requests sent by the app. In this way, the tester can also verify the correctness of the validation performed. For example, when renaming an IoT device, the app displays information that the name cannot contain special characters. The tester needs to verify how the API will behave by sending a parameter containing such characters and perform basic tests to verify the possibility of performing the injection. If error contents are noticed in the response, which may indicate how the input should be prepared, the chance of detecting a vulnerability increases significantly. Injection resistance tests are heavily supported by the analysis of the app's source code. When the app uses local databases, it is possible to read the structure of the SQL commands, which facilitates further tests.

The last testing area is the previously mentioned ability to interact with app components. Research should start by reading from the AndroidManifest file, which activities can be directly triggered from the system level – such activities are called exported activities (Code analysis). The next step is to identify actions potentially harmful to the user. Examples of such actions are removing the IoT device, logging out the user, opening a malicious link in the app, etc. The last element is to connect these two points and verify whether it is possible to perform any such actions only after activating the deeplink by the victim. Most often, the mentioned harmful activities do not have the status of exported, and it is impossible to trigger them directly. However, there are cases in which passing appropriate parameters to the exported activity makes it possible to trigger further activities without this status. The harm of initiating such an action as deleting all devices or accounts is obvious and does not require further description. However, why is the ability to open any URL in an app a high-risk vulnerability? There are two main reasons for this. First of all, WebViews are often very basic and devoid of any warnings about lack of trust or any server certificate issues. The second, more complex factor in an app with a user login function is the ability to perfectly replicate the login screen as a website and then trick the victim into logging in, often through social engineering, a common tactic used to deceive users [21]. After clicking on the link prepared this way, the victim will first see a standard app loading screen, often with the manufacturer's logo, and then a login screen that looks identical to the original one will be loaded. The process of relogging into the app presented in this way, accompanied by a logical-sounding story contained in the message with a link, does not raise any suspicions. It is worth adding that in some of the tested apps, manufacturers used deeplinks sent via e-mail or text messages to carry out various actions regarding the user's account, for example, responding to an invitation to join home control or sharing a device. Combined with the fact that many Internet users still use the same credentials for many services, and 2FA is still not popular enough, it poses a huge security threat not only to access to the user's devices and IoT cloud account but also to all other portals and services where the victim has an account [22].

Tests should be performed starting with Frida scripts that monitor the invocation of subsequent intentions along with the information transferred between them, supported by source code analysis, which can help find insecure navigation options between activities (Frida scripting, Code analysis, Deeplink testing). Additionally, both in the logs from running scripts

TABLE I
COMPARISON OF SECURITY MECHANISMS USED IN THE TESTED IOT MOBILE APPS

	Ajax	**Aqara**	**Deco**	**Dreame**	**eWeLink**	**Govee**	**Hue**	**iDoor**	**Nous**	**Mi Home**
Product type	Alarm	Home appl.	AP	Robot	Door sens.	Lights	Lights	Smart RF	Smart AC	Camera
Test date	19.07.23	29.06.23	28.07.23	29.07.23	08.09.23	30.06.23	19.06.23	31.08.23	29.06.23	20.07.23
App. ver.	2.29.1	3.2.6	3.4.25	1.6.0	4.35.1	5.7.01	4.44.0	1.52.44	1.1.6	8.6.709
Root det.	✓	X	X	✓	X	X	X	X	✓	✓
SSL Pin.	✓	✓	X	✓	X	X	△	X	✓	X
Net Res.	✓	X	✓	△	△	△	X	△	✓	✓
MFA	△	△	X	X	X	✓	△	X	□	△
Sens. data	△	X	△	△	△	△	△	X	X	X
Sec. exp.	△	✓	✓	✓	✓	X	✓	✓	✓	✓

and in the app source code, it may turn out that the tester encounters a full deeplink or a diagram of its construction, which will allow the construction of a malicious counterpart. If the proposed methods are insufficient, the tester can perform the Android Studio IDE debugging described in Phase III. Once the tester gets to the phase of verifying the effectiveness of the prepared deeplinks, apps, such as DeepLinkTester available in app stores become helpful, allowing to invoke deeplinks on the device without having to send them to the device in a more challenging way (Android studio debug, deeplink testing).

Before completing the tests and preparing the report, the tester should also test any nonstandard activities that may expose the user to danger if not used as intended. An example of such activity may be the vulnerability of an insecure QR code scanner, which the authors discovered during research on the security of IoT devices. The manufacturer's procedure for adding a device to an account required scanning the QR code on the device packaging with a smartphone. The original code contained the device's serial number. However, the scanner implemented in the app had no input data validation. After showing the QR code with the URL address in the camera's vision, the app immediately opened the website in the app view without any information, query or warning. Considering the location of the code on a box lying on a store shelf and the possibility of mass replacement by attaching a sticker, such an implementation poses a risk of a credible phishing attack. The danger is even more real because the user is exposed to it by following the instructions from the user manual. The QR code scanner activity described is just one example of a nonstandard activity that needs to be checked. The number of test cases depends on the complexity of the app, and the tester must select them independently by deeply examining the application functionality.

V. RESULTS (OVERVIEW OF RESEARCH PERFORMED)

Tests were focused on IoT devices used for smart home apps, especially devices from the popular ecosystems which can be purchased in electronics stores. Dozens of devices were tested as part of security tests. The researchers' goal was to maximize the chance of detecting as many different types of vulnerabilities as possible and to study as many ecosystems as possible. For this purpose, a list of all types of equipment has been carefully prepared: lamps, vacuum cleaners, remote controls, thermometers, thermostatic heads, smart locks, cameras, alarms, sensors, intercoms, and many others. To maximize the overview, it was decided to select devices from both specialist manufacturers producing, for example, only alarm systems and devices from manufacturers also producing other types of equipment. Attention was paid not to test the same solutions repeatedly. The equipment was ordered in batches, so it was possible to make corrections when it turned out that different manufacturers use one ecosystem, which differs only in its graphic overlay. Based on research, it can be concluded that specialist manufacturers tend to secure their devices more thoroughly, and greater manufacturer popularity usually correlates with higher quality security standards. It should also be noted that important exceptions to the rules mentioned above exist.

In this section, we describe some results obtained during penetration tests of selected IoT devices and their mobile apps with the use of the proposed methodology.

A. Apps Security Comparison

Work on the methodology concluded with comprehensive research on the security of apps controlling IoT devices using the developed approach. The successful tests, which uncovered vulnerabilities in the tested IoT solutions, demonstrate the effectiveness of this methodology. To facilitate a comparison of their security postures, a table summarizing key security mechanisms has been included, with a dedicated row indicating the test date. This addition acknowledges the dynamic nature of app development, accounting for the possibility of security measures being added subsequently to the initial testing phases. Notably, to maximize the effectiveness of the results, the authors performed complex tests on ten independent IoT solutions from various manufacturers during the preparation of this article. The forthcoming analysis will delve into the effectiveness of these security mechanisms, providing insights into the evolving landscape of IoT mobile app security.

The Table I uses the following notations.

1) *Product type*—Type of product tested with IoT mobile app
2) *App. ver.*—Version of tested app
3) *Root det.*—Presence of mechanisms verifying running app on the device with superuser privileges
4) *SSL Pin.*—Presence of SSL Pinning mechanism
5) *Net Res.*—Presence of additional traffic encryption. The table uses the following symbol: △ - which indicates that other techniques were used to block the possibility of repetition attacks and attacks using a compromised token
6) *MFA*—Presence of MFA. The table uses the following symbols: ✓- Turned on by default, △ - can be activated in settings, □ - notification after login on new device
7) *Sens. data*—Presence of sensitive data stored in unencrypted files. The table uses the following symbol: △ - which indicates that files are stored in directories inaccessible to other apps or the data is insufficient to carry out the attack
8) *Sec. deepl.*—Properly secured exported activities. The table uses the following symbol: △ - which indicates that there are no exported activities

All apps were tested in accordance with the methodology, and the research was carried out independently by the authors of this article. Due to time constraints and methodology improvement, tests of various apps were carried out at certain intervals, as shown in the table. Since the app development process is a dynamic procedure, it is possible that changing the order of app testing would affect test results. The most important tools used for research were the MobSF, Frida and Android Studio IDE solutions mentioned in the main content of this article. The data presented in Table I highlights the diverse approaches taken by different manufacturers regarding security mechanisms in IoT apps. Some apps demonstrate exemplary security by implementing all known mechanisms and best practices, including SSL pinning, root detection, and MFA. However, others lack even basic protections, exposing potential vulnerabilities like unsafe exported activities. The majority of tested apps fall somewhere in between, utilizing only a portion of available security mechanisms. This inconsistency underscores the need for industry-wide standards to ensure baseline security in IoT mobile applications.

B. Important Vulnerabilities Found

Several critical vulnerabilities were found while conducting a comprehensive penetration test on various IoT mobile apps, shedding light on potential security risks and concerns. Each vulnerability was assigned a method described in the research methodology, which allowed for its detection. The following vulnerabilities were identified along with their respective apps.

1) Insecure Exported WebView (Govee Home): The Govee Home mobile app was found to have an insecure exported WebView, exposing users to potential security threats. The lack of security consisted of the ability to open any URL through the app interface, creating a high risk of a successful credential attack. A prime example is when an attacker crafts an interface on the fraudster's website that matches the original login view of the app. Once the victim clicks on a malicious link or button in a malicious email or SMS message, the original IoT mobile app will open the address and display the login interface without arousing any suspicion, creating a considerable theft risk. It is worth noting that the use of deeplinks is quite a common practice, so a well-prepared message containing such a link should not surprise potential victims. The vulnerability was detected by the testing methods described in the WebView testing part of Phase IV of the methodology. This vulnerability, with a high CVSS score of 8.8, can potentially lead to unauthorized access and compromise of sensitive user data.

1) *CVSS Score:* 8.8.
2) *CWE:* CWE-749 Exposed Dangerous Method or Function.
3) *CVE Identifier:* CVE-2023-3612 [24].
4) *Research method:* Phase IV - Deeplink testing.

2) Insecure WebView and QR Code Scan (Imou Life): The Imou Life app exhibited vulnerabilities in both its WebView implementation and QR Code scanning functionality. Their harmfulness is similar to the one described in the previous point found in the Govee Home app. As in that case, the app has an insecure view that allows the tester to open any URL in the interface, enabling the attack on credentials described above. Apart from the scenario where the victim opened a deeplink in the Imou Life app, the attack could be carried out using a malicious QR code. Adding a new device required scanning the QR code in the app located on the device or box. After activating the add device function in the app, a scanner interface appeared, instructing the victim to scan the code. When a QR code containing a URL appeared in the scanner area, the app immediately opened the address in the app interface without any warning or message. Combining this defect with the fact that the QR code on the box lying on the shelf in the store could be pasted creates a very real and dangerous attack scenario because the victim would be exposed to it by following the manufacturer's instructions. The vulnerability was discovered during WebView activity tests, which are part of Phase IV in the methodology.

1) *CVSS Score:* 8.1.
2) *CWE:* CWE-384, Session Fixation.
3) *CVE Identifier:* CVE-2023-6913 [25].
4) *Research method:* Phase IV - Deeplink testing, injection payload testing.

3) Insecure Adding New Device Process (Govee Home): The Govee Home app and API was found to have a vulnerability in process of adding a new device, potentially exposing users to security risks. The vulnerability was critical and allowed taking control of devices of at least several series. The main disadvantage was the use of the approach of guaranteeing security through obfuscation. The procedure for pairing a new device is divided into two parts. The first one involved establishing a connection between the control device and the IoT device via the Bluetooth interface. After reading parameters from the IoT device, such as the ID number, which was mainly the MAC address, the second stage began. Using cryptographic functions, the app prepared a POST query containing the downloaded data and the security value. Sending this query to the server and receiving a response

completed the pairing procedure. Volatility analysis, reverse engineering and Frida scripts allowed us to discover how to generate the security parameter. Thanks to this, it was possible to prepare these parameters for each device and perform a brute force attack, taking over all devices one by one. It is worth noting that when adding a device to an account, it was removed from the account to which it was previously assigned, so the actual owner lost control over it until it was paired again using a complete procedure requiring Bluetooth communication. To detect this vulnerability, combining the results of tests conducted during Phases II and III was crucial. Conducting tests in the mentioned phases separately would not have allowed for the detection of such a significant vulnerability. In the case of separate tests, we miss the information needed to identify the vulnerability. Procedures tested individually may appear safe, but by testing the entire process and integrating knowledge from different areas, we can uncover critical vulnerabilities like this one.

1) *CVSS Score:* 10.0.
2) *CWE:* CWE-863, Incorrect Authorization.
3) *CVE Identifier:* CVE-2023-4617 [23].
4) *Research method:* Combination of test results performed in Phase II and III.

4) Insecure Deleting Device Process (Confidential Information): Similar to the previous vulnerability, the vulnerable app displayed a flaw in its device deletion process. Specific details and a CVE identifier are not disclosed due to the sensitive nature of the information. The defect described in this part actually concerns the API that acts as an intermediary between the control app and IoT devices. Network traffic analysis performed in Phase II revealed control queries being sent to the cloud. Based on the intercepted communication, authorization and access control tests were performed. Research has shown that it is possible to remove any device from the cloud-based only on its ID. When sending a request to remove a device, the server did not verify the fact that a given device was owned by the user whose token authorized the sent request. In this way, removing all devices connected to the manufacturer's cloud was possible using a brute force attack.

1) No additional details provided. No CVE Identifier assigned.
2) *Research method:* Combination of test results performed in Phase II and III

5) Insecure Lock Screen - eWeLink: The eWeLink mobile app was found to have an insecure lock screen, posing a potential security threat with a CVSS score of 7.7. The vulnerability discovered in this app was the ability to bypass the lock screen set by the user. In the available security settings, it was possible to activate additional security in the form of the need to draw a pattern before gaining access to IoT device control and other app functions. In practice, the mechanism was defective. Once the lock screen appears, closing the app and reopening it bypasses the lock screen. This was due to an incorrect implementation of the mechanism for counting the time since the last use of the app. The vulnerability was discovered during Phase II testing of access control. The app manufacturer assumed the lock screen should not be displayed if the user had just used the app, which is quite a correct assumption. However, the timestamp of the last time the app was used was overwritten every time it was closed, even if it was not unlocked properly. This vulnerability could allow unauthorized access to the device and compromise user privacy.

1) *CVSS Score:* 7.7.
2) *CWE:* CWE-269, Improper Privilege Management.
3) *CVE Identifier:* CVE-2023-6998 [26].
4) *Research method:* Combination of test results performed in Phase III and IV.

C. Minor Security Flaws Found/Identified

While the primary focus of this study is on critical vulnerabilities, we also identified several smaller security flaws that, although not classified as vulnerabilities, reflect suboptimal security practices that could be improved. These findings provide valuable insights into areas where security measures in IoT mobile apps can be enhanced to reduce potential risks.

For instance, some apps transmitted user credentials in plain text during the first login. While the communication was over HTTPS, the lack of SSL pinning left the credentials susceptible to interception through MITM attacks. Although this does not constitute a vulnerability per se, it highlights an opportunity to strengthen security by implementing SSL pinning.

Another noteworthy finding involved authorization token misuse across different app versions. In one case, a newer app version introduced improved security mechanisms, but stolen tokens from the updated version could still be used to access endpoints in an older, less secure API version. This issue arose because the older API lacked proper validation to ensure that tokens were tied to specific endpoints. Easy mitigation would involve verifying which endpoint the token was issued for, but the discovery of this issue was possible only because we tested different app versions and cross-referenced API behaviours. Without this approach, such flaws would likely remain undetected.

Additionally, smaller defects included the ability to block a device's connection to the Internet by spoofing requests for a new certificate. While this type of defect is not critical, it highlights the importance of robust mechanisms for certificate handling to prevent service disruptions.

Another issue involved storing (nonsensitive) historical data from user devices in server archives accessible via a simple URL without any authentication. This practice poses a potential risk if deterministc URL generation is used, allowing unauthorized access to archived data. While no sensitive information was exposed, securing these archives through access control measures or token-based validation would be a straightforward improvement.

Furthermore, the authors observed that several apps logged sensitive data, such as device IDs or usernames, into their internal logs. While this information alone was insufficient to compromise user accounts or gain control of devices, it could assist attackers in shortening their attack path. By combining such data with information obtained from other sources,

attackers could potentially increase the efficiency of their exploitation efforts. Addressing this issue by anonymizing or minimizing sensitive data in logs would be a simple yet effective way to reduce the associated risks.

Such findings highlight the nuanced security landscape of IoT mobile apps. While these flaws may not meet the criteria for critical vulnerabilities, addressing them would contribute to a more secure user experience and reinforce the perception of IoT apps as trustworthy systems. Notably, all these findings were made possible by the comprehensive testing methodology described in this study, underscoring its value in uncovering both critical vulnerabilities and less obvious security flaws.

VI. Conclusion and Future Works

The methodology presented in this article allows for comprehensive research on IoT mobile apps, focusing not only on the typical software part but also taking into account the security of the entire IoT solution. The proposed tests provide the opportunity to examine the most important areas of potential threats to users, owners, and producers of such systems in mobile apps. The methodology also provides the opportunity to obtain data from apps that can be used for subsequent testing of APIs and IoT devices themselves as ordinary devices connected to an IP network. The presented tests include attack scenarios based on various conditions, such as resistance to brute force attacks, repetition attacks, attacks with knowledge of residual data, attacks with access and lack of access to files, as well as phishing attacks, and many others. The methodology draws attention to the need to examine the IoT solution as separate areas (such as the area of IoT mobile app, API or the IoT device itself) and then compare these results. While it is difficult to look at all these areas holistically, it is essential to combine research results from various fields. Sometimes, the most critical vulnerabilities occur at the junctions of areas. For IoT mobile apps, the key security question revolves around their ability to securely manage the processes of adding and controlling devices, which fundamentally sets them apart from regular applications. Addressing this question requires the robust implementation of cryptographic functions to prevent misuse, such as generating parameters for unauthorized API requests, as well as securing network traffic to mitigate the risk of MITM attacks. Techniques like SSL pinning are critical steps toward achieving this level of protection and ensuring the integrity of IoT ecosystems. The tests carried out in accordance with the prepared methodology led to the detection and reporting of several high-severity vulnerabilities, as well as many smaller security defects, which are not vulnerabilities in themselves but negatively affect the security of the solution. Based on the experience of IoT security research from recent years, it should be noted that the level of security is gradually improving, and in many tested devices, particularly those from larger, more experienced manufacturers, no vulnerabilities were detected. However, while today's security level is better than it was in earlier days, it is still far from perfect. As a result, it remains difficult to honestly admit that the IoT world has completely eliminated the negative label of an (in)security zone.

The level of security of IoT devices in the context of mobile apps used to control them varies, but in the case of some devices, significant vulnerabilities were found. Some of them were not exploitable on a mass scale, which does not significantly reduce their criticality. What is most important is the fact that the maturity of some IoT device manufacturers is unsatisfactory not only in terms of ensuring security but also in responding to discovered vulnerabilities. Specifically, some manufacturers did not react at all or failed to release patches, which is why we are still unable to provide more details about the vulnerabilities found in their devices. Some of the vulnerabilities found do not concern the implementation issues (which could be relatively more straightforward to patch) but refer to the design issues, such as the ability to take control over a device without ensuring that we have physical access or own that device. We believe that the presented article, which describes the methodology of testing mobile apps in the context of IoT devices and the results achieved through research following that methodology, is a small but essential step toward building more resilient IoT ecosystems. By outlining the necessary procedures, tools, and techniques for evaluating IoT application security, this research offers practical guidance for enhancing security practices in IoT ecosystems. Additionally, the responsible disclosure of identified vulnerabilities to CVE numbering authorities (CNAs) led to patches that directly enhanced the security of these solutions, contributing positively to the safety of IoT devices and their users.

In the future, the authors plan to conduct security tests on a larger number of IoT devices to report any vulnerabilities to the relevant entities, thereby contributing to the improvement of user safety. Additionally, the results of the authors' research in the area of device communication via Bluetooth have inspired the development of a project called BlueSploit. This project aims to create a tool capable of detecting nearby Bluetooth devices (including IoT devices), enumerating available services on those devices, performing mass attacks, such as BLE Spam, and the possibility to execute control scripts when a matching script from the database is found.

References

[1] (Cambridge Univ. Press, Cambridge, U.K.). *Cambridge Advanced Learner's Dictionary & Thesaurus*. Accessed: Jan. 15, 2024. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/internet-of-things?q=Internet+of+things

[2] M. Janiszewski, A. Felkner, P. Lewandowski, M. Rytel, and H. Romanowski, "Automatic actionable information processing and trust management towards safer Internet of Things," *Sensors* vol. 21, no. 13, p. 4359, 2021, doi: 10.3390/s21134359.

[3] K. Lueth (IoT-Analytics, Hamburg, Germany). *State of the IoT 2020: 12 Billion IoT Connections, Surpassing Non-IoT for the First Time*. 2021. Accessed: Jan. 15, 2024. [Online]. Available: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time

[4] M. Georgiou (Imaginovation, Raleigh, NC, USA). *The Future of IoT Development: Trends and Predictions for 2024*. 2023. Accessed: Jan. 17, 2024. [Online]. Available: https://imaginovation.net/blog/iot-development-trends-predictions/

[5] H. Patil and K. Sharma, "Assessing the landscape of mobile data vulnerabilities: A comprehensive review," in *Proc. Int. Conf. Comput. Intell. Sustain. Eng. Solut. (CISES)*, 2023, pp. 79–87, doi: 10.1109/CISES58720.2023.10183390.
[6] P. Weichbroth and Ł. Łysik, "Mobile security: Threats and best practices," *Mobile Inf. Syst.*, vol. 2020, no. 1, 2020, Art. no. 8828078, doi: 10.1155/2020/8828078.
[7] S. Neupane et al., "On the data privacy, security, and risk postures of IoT mobile companion apps," in *Proc. 36th Annu. IFIP WG 11.3 Conf. Data Appl. Secur. Privacy (DBSec)*, Newark, NJ, USA, 2022, pp. 162–182, doi: 10.2139/ssrn.4121997.
[8] E. Chatzoglou, G. Kambourakis, and C. Smiliotopoulos, "Let the cat out of the bag: Popular android IoT apps under security scrutiny," *Sensors*, vol. 22, no. 2, p. 513, 2022, doi: 10.3390/s22020513.
[9] A. Alshehri, P. Marcinek, A. Alzahrani, H. Alshahrani, and H. Fu, "PUREDroid: Permission usage and risk estimation for android applications," in *Proc. 3rd Int. Conf. Inf. Syst. Data Min. (ICISDM)*, Houston, TX, USA, 2019, pp. 179–184, doi: 10.1145/3325917.3325941.
[10] "OWASP Internet of Things," Owasp.org. Accessed: Jan. 15, 2024. [Online]. Available: https://owasp.org/www-project-internet-of-things/
[11] "OWASP mobile application security testing guide." Owasp.org. Accessed: Jan. 15, 2024. [Online]. Available: https://mas.owasp.org/MASTG/
[12] "OWASP mobile top 10." Accessed: Nov. 20, 2024. [Online]. Available: https://owasp.org/www-project-mobile-top-10/
[13] C. Polop, "HackTricks mobile pentesting." Book.Hacktricks.xyz. Accessed: Jan. 15, 2024. [Online]. Available: https://book.hacktricks.xyz/mobile-pentesting/
[14] E. Süren, F. Heiding, J. Olegård, and R. Lagerström, "PatrIoT: Practical and agile threat research for IoT," *Int. J. Inf. Secur.*, vol. 22, pp. 213–233, Feb. 2023. doi: 10.1007/s10207-022-00633-3.
[15] F. Heiding, E. Süren, J. Olegård, and R. Lagerström, "Penetration testing of connected households," *Comput. Secur.*, vol. 126, May 2023, Art. no. 103067, doi: 10.1016/j.cose.2022.103067.
[16] "Mobile security framework." Mobsf.github.io. Accessed: Jan. 17, 2024. [Online]. Available: https://mobsf.github.io/docs
[17] "Frida-dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers." Frida.re. Accessed: Feb. 22, 2024. [Online]. Available: https://frida.re/
[18] "MQTT-the standard for IoT messaging," Mqtt.org. Accessed: Aug. 23, 2024. [Online]. Available: https://mqtt.org/
[19] "CoAP-constrained application protocol." Coap.space. Accessed: Aug. 23, 2024. [Online]. Available: https://coap.space/
[20] "Android studio-official integrated development environment for android app development." Developer.android.com. Accessed: Feb. 22, 2024. [Online]. Available: https://developer.android.com/studio
[21] H. Chheda, (Sprinto, San Francisco, CA, USA). *Social Engineering Statistics: How Can Your Business Avoid Being One?* 2024. Accessed: Aug. 23, 2024. [Online]. Available: https://sprinto.com/blog/social-engineering-statistics/
[22] A. Cardwell. "The growing threat of password insecurity." Linkedin.com. 2023. Accessed: Jan. 17, 2024. [Online]. Available: https://www.linkedin.com/pulse/growing-threat-password-insecurity-andrew-cardwell-p3xzc
[23] "CVE-2023-4617." Nist.gov. Accessed: Jan. 5, 2025. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-4617
[24] "CVE-2023-3612." Variotdbs.pl. Accessed: Aug. 23, 2024. [Online]. Available: https://www.variotdbs.pl/vuln/VAR-202309-0497/
[25] "CVE-2023-6913." Variotdbs.pl. Accessed: Aug. 30, 2024. [Online]. Available: https://www.variotdbs.pl/vuln/VAR-202312-1090/
[26] "CVE-2023-6998." Variotdbs.pl. Accessed: Aug. 30, 2024. [Online]. Available: https://www.variotdbs.pl/vuln/VAR-202312-2498/

Jan Adamski received B.S. degree in telecommunications from the Faculty of Electronics and Information Technology, Warsaw University of Technology, Warsaw, Poland, in 2023.

From an early age, he was infected with passion for modern technologies. He started his professional career by working in the American startup Cerebre, Warsaw, as a Software Engineer. In 2022, he was employed as a Software Engineer with the Cybersecurity Department, NASK, Warsaw, and from the beginning of 2024, he was promoted to the position of Senior Software Engineer. He was particularly involved in the LaVA (Laboratory for testing the vulnerabilities of stationary and mobile IT devices, algorithms, and software) project in the area of mobile applications. He was the speaker at the "THS 23" and "OMH 23" Cybersecurity conferences which took place in Poland.

Marek Janiszewski received the M.S. and Ph.D. degrees in information technology and telecommunication from Warsaw University of Technology, Warsaw, Poland, in 2012 and 2023, respectively, and the M.S. degree in economics from Warsaw School of Economics, Warsaw, in 2013.

Since 2016, he has been working with NASK—National Research Institute, Warsaw, where he is responsible for the conceptualization, design, development, and implementation of new cybersecurity tools and participates in international and national research projects. He has experience in conducting security audits and penetration testing, as well as consulting projects in building ICT infrastructure, administering IT systems, and managing development projects. He holds industry certifications, including Offensive Security Certified Professional (OSCP) and Certified Ethical Hacker (CEH). He is the author of several publications in polish and international journals and conference proceedings in the field of cybersecurity (in particular on trust and reputation management systems, vulnerability detection, and management). His research interests include issues related to cybersecurity, including research on the effectiveness and reliability of trust and reputation management systems, development and evaluation of vulnerability detection tools and methods, and security assessment in the Internet of Things.

Marcin Rytel received the B.S. degree in telecommunications from Warsaw University of Technology, Warsaw, Poland, in 2016.

He began his professional career with the Faculty of Electronics and Information Technology, Warsaw University of Technology, where he designed, built, and tested electronic devices operating at microwave frequencies. Since 2019, he has been working with NASK—National Research Institute, Warsaw, specializing in the security of Internet of Things devices. He played a key role in the VARIoT (Vulnerability and Attack Repository for IoT) project, focusing on acquiring information about IoT vulnerabilities from various public sources. In 2024, he earned the Certified Ethical Hacker (CEH) certification.