smart cities

MDPI

Article

Validation of Authentication Measures Implementation in IoT Mobile Applications

Gema Fernández Bascuñana

European Union Agency for Network and Information Security, 1 Vasilissis Sofias Street, 15124 Maroussi, Attiki, Greece; gema.fernandez.bascunana@gmail.com

Received: 9 April 2019; Accepted: 14 May 2019; Published: 17 May 2019

Abstract: Smartphones are an integral cog of the IoT environment and a fundamental bloc of any related security solution, given that IoT mobile applications allow users not only to get information, but also to influence the environment. This paper presents a methodological instrument that can contribute to implementing and evaluating security measures in mobile applications by means of an automated analysis tool. A clear process for linking policy and high-level security guidelines and measures to concrete source code elements is depicted, as well as an automated way of testing a set of mobile applications against them. In addition, the obtained results highlight the current state of authentication measures' implementation in IoT mobile applications; at the same time, it is important to note that the proposed approach is generic enough to accommodate other security principles as well.

Keywords: authentication; mobile applications; internet of things; cybersecurity

1. Introduction

The Internet of Things (IoT) has brought a paradigm shift in traditional computing by bridging the digital and the physical worlds, and has penetrated all facets of modern life with the continuously growing IoT market. Cybersecurity is one of the most worrisome concerns and challenges for the successful deployment of the IoT [1]. A series of recent attacks on the IoT ecosystem [2] has highlighted the need for corresponding cybersecurity measures to be introduced in the realm of the IoT. Nevertheless, the efforts and resources that organizations and companies allocate toward cybersecurity are currently conditioned by economic constraints; traditional approaches are being invalidated due to issues such as being the first to market or the very high scalability and diversity of deployment [3]. Among several gaps that need to be tackled, the lack of strong authentication and authorization processes remains key to protect communications, privacy, and access to resources. These processes should be implemented across the whole IoT ecosystem (smart things, cloud, applications, etc.) to foster interoperability [2].

The IoT is predominantly user-centric, and in this respect, when it comes to the users, the elements by which they mainly communicate and interact with IoT elements such as smart devices or the cloud are smartphones; the latter play the role of an intermediator and interface for users to the IoT by means of mobile applications [4]. By using smartphones in the context of IoT cyber-physical systems, users can not only access information, but also command and control smart devices that can influence the physical world. Accordingly, the need for the secure implementation of mobile applications becomes crucial [5]. Moreover, smartphones are progressively becoming more important in the context of authentication, because they hold multiple authentication interfaces and processes (user, cloud, smart devices). In this respect, tackling authentication in IoT mobile applications emerges as a priority for security of the IoT environment [6], as they would pose a high risk if compromised.

This paper aims at providing a methodological instrument to evaluate the implementation of security measures concerning IoT mobile applications, by means of an analysis tool that enables the automation of such a process. We focus on a case study for security measures in the field of authentication, whereas it needs to be specified that the proposed methodology is generic and could be equivalently applied to other security aspects. This methodology enables the organized, structured, and effective assessment of mobile applications and contributes to addressing the challenge of IoT security. Assessment is conducted against a set of good practices, such as the ones provided by the European Union Agency for Network and Information Security (ENISA) [7] or other bodies [8]; furthermore, it will assist in the implementation and integration of recommended security measures, hence contributing to a more secure IoT in a pragmatic manner.

In what follows, we first present an overview and description of the authentication state of the art in the context of the IoT, including authentication interactions in the IoT environment, features inherent to the IoT that will condition the authentication processes applied to the environment, and the most used authentication protocols. Then, we detail the proposed generic methodology and list the steps to follow, while elaborating on the possible uses of this methodology. Thereupon, we report on an experimental validation of our methodology by considering two different IoT authentication protocols that inform both the application and the network layers. Finally, conclusions are drawn, and future lines of work are underlined.

2. Iot Authentication Mechanisms: State of the Art

In order to evaluate the implementation of authentication measures in IoT mobile applications, we first define the scope of the research by focusing on aspects such as which interactions involve authentication processes or which protocols are normally used to perform them.

2.1. Iot Authentication Reference Model

The IoT landscape embraces a variety of different elements with diverse characteristics that clearly condition the authentication processes. In order to ensure secure interactions among them, it is important to differentiate these elements and understand their particularities and constraints. Accordingly, we map these elements of the IoT in a reference model that is illustrated in Figure 1 to facilitate common understanding and promote consistent use throughout our work.

The IoT authentication reference model shown in Figure 1 is based on the IoT high-level model proposed by ENISA in its Baseline Security Recommendations for the IoT [2]. The reference model highlights the smartphone as a central element, communicating with all others in the IoT ecosystem, which is one of the most common deployments in IoT environments. The dotted arrows indicate authenticated communication links, thus exposing the ubiquitous penetration of authentication mechanisms across the IoT realm.

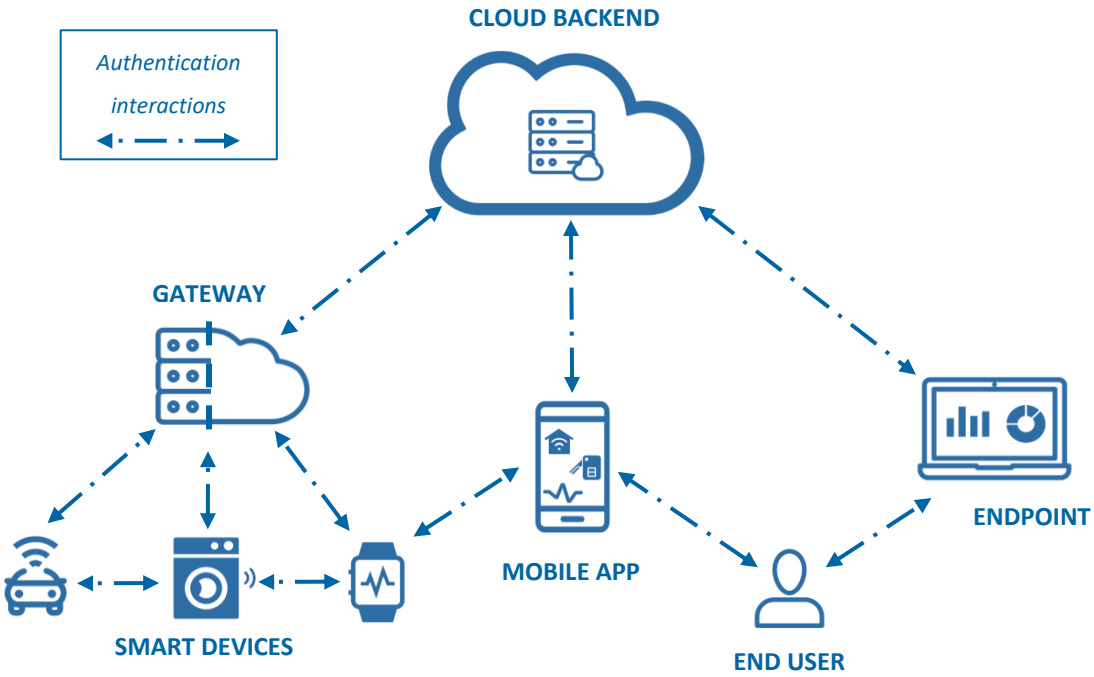

Figure 1. Internet of Things (IoT) authentication reference model.

2.2. Iot Authentication Challenges

IoT devices and systems present specific features and characteristics that distinguish them from their traditional IT counterparts. Accordingly, special consideration to these features needs to be given in regard to security processes and authentication in particular. According to the OWASP IoT Framework Assessment [9] as well as the research conducted by the CORE Working Group [10], key characteristics include:

- Amount and heterogeneity of devices: the incredibly high (and increasing) number of IoT devices as well as their heterogeneous nature (e.g. different applications, functions, manufacturers) pose a challenge when it comes to identity provisioning/management, as well as authentication processes.
- Resource and capability constraints: most authentication mechanisms rely on the possession and validation of credentials, which can be seen as a hindering factor for constrained devices.
- Changes in location and time: smart devices are not static either in terms of location (changes in physical location and consequently in network) or in terms of time (multiple events happening simultaneously). This volatility needs to be taken into account when it comes to providing suitable and resilient authentication processes.

These inherent features of the IoT, and particularly its edge components, give rise to concrete challenges when it comes to authentication processes and mechanisms. For instance, the use, management, provisioning, revocation, and storage of keys/certificates become difficult due to the amount of devices and their possible constraints. In addition, when such a large number of devices with changing locations and states need to continuously authenticate, the traffic load increases and can lead to network congestion [11]. Furthermore, if devices that use different protocols need to authenticate each other, interoperability must be ensured.

Aiming at alleviating these challenges and concerns, IETF Working Groups such as ACE [12] and CORE [13], as well as oneM2M [14] among others, produced several documents and drafts in which the analysis of specific recommendations and solutions were presented. Among them, noteworthy approaches include secure bootstrapping (e.g. pre-provisioned keys in devices),

group/delegated authentication (that would decrease the authentication traffic load), and dynamic authentication (to address the changes in location and time of the devices).

2.3. Iot Authentication Protocols, Frameworks, and Standards

When considering IoT authentication, there is a growing tendency to use proprietary or ad hoc solutions to meet security needs and requirements [15]. However, interoperability is a crucial aspect of reaching the full potential of the IoT, and this becomes even more important in the context of authentication. The multi-tenancy model of the IoT when it comes to devices and the cross-network interactions further necessitates the secure handling of authentication and authorization. In the following, we give a brief overview of some of the most popular general authentication protocols and frameworks, focusing on different layers in the communication stack in order to build the bigger picture for the IoT authentication landscape.

2.3.1. Application Layer

These protocols and frameworks apply to the interaction between users' endpoints (such as laptops or smartphones) and the rest of the environment at a high level in the communication stack. Here, we have included some of the most popular and used general authentication protocols and frameworks at an application layer.

- Security Assertion Mark-up Language (SAML): SAML is an open security standard developed by OASIS that allows the exchange of authentication and authorization information among different parties or entities [16]. It is an XML-based framework that supports Single Sign On (SSO) and identity federation mainly in accessing web resources. It defines the role of a trusted identity provider that stands between the user who wants access to some resources and the service provider in charge of their access control. SAML is well targeted for IoT deployments, as the standard defines possible communication flows among parties, profiles, and bindings with other communication protocols (HTTP, SOAP). In addition, SAML is compatible with multiple authentication mechanisms and architectures (e.g. Kerberos, (un)protected usernames and passwords over HTTP, X.509 Public Key Infrastructure, Pretty Good Privacy (PGP), Smart Cards ...), showing a huge degree of interoperability, which is certainly a desired characteristic for IoT deployments. Regarding resource and capability constraints, most authentication mechanisms rely on the possession and validation of credentials, which can be seen as a hindering factor for constrained devices.
- OAuth 2.0 and OpenID Connect (OIDC): OAuth 2.0 is an open standard developed by the IETF Auth WG. It consists of an authorization framework that aimed to provide secure delegated access to resources or services by a third party on behalf of the owner [17]. By means of introducing an authorization layer, the resource owner does not have to share its credentials with any third-party application. Although originally thought for HTTP resource access, the standard provides several extensions to allow interoperability, such as SAML2, JSON (JavaScript Object Notation), Web Token (JWT), or even for the use of OAuth with native apps or resource-constrained devices. It is worth noting that OAuth is an authorization protocol, and not an authentication one. To solve this issue, OpenID Connect [18] is an identity layer built on top of OAuth 2.0 to fill the authentication gap. This allows third-party apps (clients) to authenticate/verify the identity of the users. In terms of security, it is compliant with ISO/IEC 29115 Entity Authentication Assurance [19], and therefore offers four different levels of authentication assurance, allowing for example the use of encryption, which is quite pertinent also for IoT environments.
- Fast IDentity Online (FIDO): In terms of multi-factor authentication (MFA) in the context of IoT, FIDO aims at addressing the lack of interoperability between authentication devices and the inconvenience of remembering multiple passwords [20]. It combines PKI mechanisms with a second factor that protects access to the private key, thus strengthening the security of

authentication. Moreover, FIDO defines how its schema can be used in conjunction with other authentication protocols (such as the ones mentioned above) as well as in the context of the IoT, focusing on smartphones as users' primary interaction devices to communicate to the rest of the IoT ecosystem.

- Constrained Application Protocol (CoAP) on top of Datagram Transport Layer Security (DTLS): When devices communicate with each other, standardized as well as proprietary mechanisms exist to establish keys for communication and authenticate in a secure manner. IETF standards in this respect include the Datagram Transport Layer Security (DTLS) [21], which uses Transport Layer Security protocol (TLS) over User Datagram Protocol (UDP), and not over Transmission Control Protocol (TCP) as most implementations. In the case of IoT resource-constrained devices, DTLS can be extended with different security mechanisms when Constrained Application Protocol (CoAP) [22] is used at the application layer, e.g. security modes such as NoSec (no security at all), PreSharedKey (symmetric authentication), RawPublicKey (asymmetric authentication), and Certificate.

In order to make the most of these security modes, and at the same time tackle some of the challenges previously mentioned regarding the use, management, and storage of keys and certificates, several proposals have been made. For example, lightweight versions of X.509 certificates removing non-essential elements to make it more constrained or oriented have been proposed. Moreover, in order to address the network scalability and mobility of devices, a lightweight DTLS-based keying mechanism for CoAP IoT smart objects has been designed, as well as a DTLS-based security scheme with two-way authentication for the IoT.

2.3.2. Communication Layer

The other layer to tackle is the communication layer. Even when authentication protocols do not exist at a lower layer in the communication stack per se, the protocols used for communication between smart devices and smartphones provide mechanisms to perform the authentication process. Most used protocols for IoT devices interactions such as Zigbee [23], Wi-Fi [24], or Bluetooth Low Energy (BLE) [25] introduce specific authentication features and mechanisms. Due to space limitations, we focus here on BLE.

- Bluetooth Low Energy (BLE): Bluetooth Low Energy [25] (BLE) is a version of Bluetooth technology that focuses on low-energy consumption, tackling problems from old Bluetooth versions such as fast battery draining and the frequent loss of connection, requiring frequent pairing and re-pairing. Such features closely relate to the needs of IoT, which further supports why BLE is widely used in such environments. Bluetooth Core Specification provides a number of features that cover security aspects such as encryption, trust, data integrity, and the privacy of the user's data [26]. In terms of authentication, the pairing process allows BLE devices to exchange device information for the establishment of a secure link. The following options exist in the different versions of BLE:
    - Legacy Pairing (BLE 4.0/1): The devices exchange a Temporary Key (TK) and use it to create a Short-Term Key (STK), which is used to encrypt the connection. The security of the entire process depends on the pairing method used to exchange the TK:
        - Just Works is an implementation where the TK is set to '0000', making it easy for an adversary to brute force the STK and eavesdrop on the connection. Moreover, there is no way of verifying the devices in the connection, and thus offers no man-in-the-middle (MITM) protection.
        - Passkey is a similar concept, except that the TK is a six-digit number that is passed between the devices by the user. This method provides relatively good protection from passive eavesdropping, except if the adversary is present during the pairing process and is able to eavesdrop the exchanged values.

- Out-of-Band (OOB) pairing involves exchanging the TK using a different wireless technology such as NFC. This allows for the exchange of a very large TK, up to 128 bits, greatly enhancing the security of the connection

- Secure Connections (BLE 4.2/3): This new model introduces the numeric comparison method to the three methods mentioned above. Instead of using a TK and STK, LE Secure Connections use a single Long-Term Key (LTK) to encrypt the connection. The LTK is generated and exchanged using Elliptic Curve Diffie–Hellman (ECDH) public key cryptography. This approach offers significantly stronger security when compared to the original BLE key exchange protocol.

Regarding the IoT, the type of pairing that is being implemented depends highly on the capabilities of the device, meaning that if the device is not equipped with any input or output mechanism, the available types of pairing are limited.

3. Methodology for Evaluating the Secure Implementation of Authentication Protocols

In order to evaluate the implementation of the aforementioned authentication measures and protocols in IoT applications, we developed a methodology to guide the analysis and draw meaningful insights in a systematic manner.

As shown in Figure 2, this stepwise methodology aims at setting a well-constructed and organized roadmap for how to analyze IoT mobile applications in order to evaluate, in general, the security measures implementation within the application for the target selected protocol.

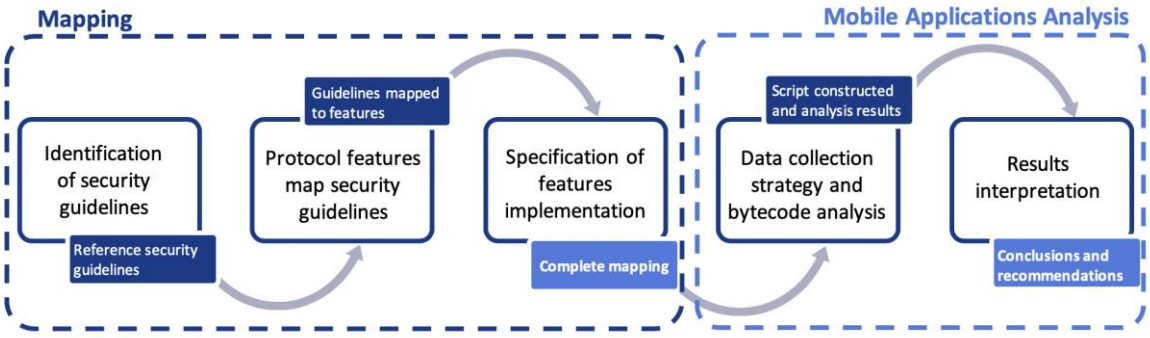

Figure 2. Methodology steps to evaluate the secure implementation of authentication protocols.

It consists of two complementary parts; each one is associated with a complete, meaningful and directly applicable output. The first one corresponds to the mapping of high-level security guidelines to specific code elements; this mapping represents a valuable instrument for information security developers who wish to implement good security practices and policies in an effective and demonstrable manner. The second one refers to the analysis tool, by means of which the Chief information security officers or any other person responsible for ensuring the secure implementation of mobile applications can evaluate them in an automated manner. The following sections will drill down on the steps of the methodology, with a focus on authentication.

3.1. Step 1: Identification of Security Guidelines

The first step involves identifying appropriate reference security guidelines that provide good practices and recommendations for the secure development of mobile applications, such as the Smartphone Secure Development Guidelines [7] developed by ENISA. These guidelines are aimed at the developers of smartphone applications in order to support them in protecting these applications from threats in the mobile computing environment. The output of this first step will be a dedicated set of security guidelines, the proper implementation of which will be evaluated in the following steps.

3.2. Step 2: Protocol Features Mapping Against Security Guidelines

The next step consists of translating the high-level security guidelines to the corresponding specific protocol features, e.g. methods or operations of protocols. This process involves identifying instances of each generic guideline, which are specified as a concrete action, parameter, or option in the protocol. For example, in the case of the CoAP, a guideline regarding using asymmetric authentication could be mapped, as shown in Figure 3. The output of this step is the complete set of protocol features mapped against selected security guidelines. This is not always an easy and straightforward process, depending on the specific protocol and security guidelines; however, it has to be highlighted that it can be a one-to-many mapping (one security guideline can be covered by a set of protocol features). This way, the process of finding features to cover all the target security guidelines is eased.

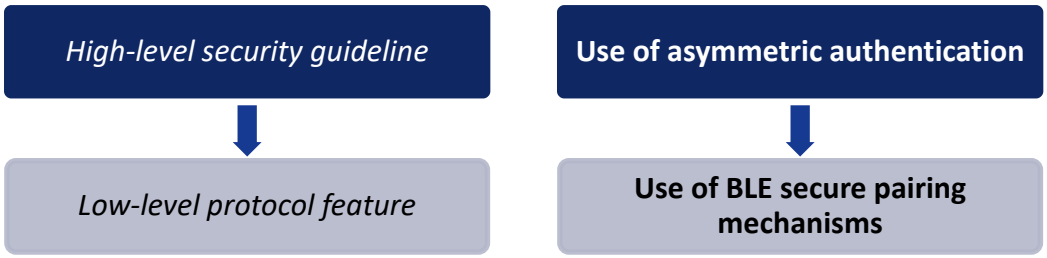

Figure 3. From high-level security guidelines to low-level protocol features.

3.3. Step 3: Specification of Features Implementation in API Calls

Once we have the mapped protocol features against security guidelines, the next step is to pinpoint specific instances of these features in the mobile application code. Ideally, the selected protocol would have a well-documented API for the specific programming language (e.g. Java for Android and Swift for iOS [27]), from which we can extract elements (e.g. methods, input parameters) that can be linked to the protocol features listed in Step 2. The output of this phase will be the mapping of all three first steps (security guidelines, protocol features, and code elements), as shown in Table 1.

Table 1. Sample mapping table.

Smartphone Secure Development Guidelines	**Protocol Features**	**Bytecode Search**
Summary	*Summary*	*Element to find*
First Section, guideline 2 Third Section, guideline 5	Protocol method/parameter 1 Protocol method/parameter 2	methodX(), PARAM_1 methodY(), PARAM_2

As stated before, this mapping table constitutes by itself a valuable output for information security developers, easing the process of implementing high-level security guidelines in mobile applications.

3.4. Step 4: Data Collection Strategy and Bytecode Analysis

The next step involves collecting a base set of mobile applications to analyze and evaluate whether they include the aforementioned code snippets in their source code. That would be an indicator of them respecting the corresponding security guidelines. For this step, bytecode analysis of the code of mobile applications is conducted (We acknowledge the limitation of this work in that it focuses on bytecode analysis, which is not the case for all mobile application programming languages. However, given the resources and time constraints of this work, we focused on bytecode analysis as a representative proof-of-concept. Further adapting this work can lead to a more generic approach. Nonetheless, we need to underline the generality of the proposed methodology and that an extension to other programming languages is merely subject to further development efforts.). Using the code

elements mapped in the previous step as input, bytecode instances that are identical to such elements have to be defined.

Following that, structured analysis of the bytecode takes place in order to be able to draw meaningful insights. The notion of 'structure' refers to a methodical ordering of the elements to be searched for in the bytecode. For example, if the implementation of a method implies the existence of a protocol feature translated into code, then it becomes a straightforward search. This process is mapped on decision trees to facilitate processing. The complexity of this strategy increases when the depth of the decision tree grows, as shown in Figure 4.

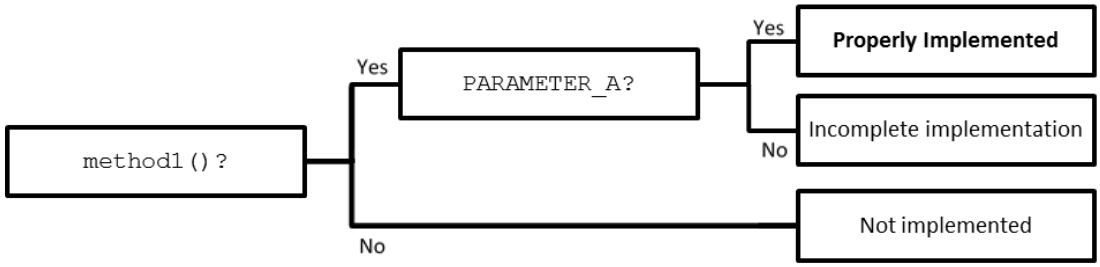

Figure 4. Decision tree built by cascaded searches (i.e., including more than one code element).

In order to automate and structure this process, we developed a Linux bash script. This script allows analyzing a set of applications located in a specific directory, providing the results by means of a text file. A sample option for configuring the script according to our needs follows:

searchArray[**1**]= "**element2find**" # Independent searches specified by array elements

These elements correspond to the 'Bytecode searches' specified in the mapping table, which means that they are directly linked to a given security measure.

Once the analysis is completed, the results are stored in a text file, specifying the path to the class where the sought element was found. Finally, some statistics are attached at the end of the file in terms of the number of applications containing the element:

1.element2find /com/example/app/smali/class.java [. . .]

100 results for: element2find

The script that we have developed has been specified to search for bytecode instances within smali files. However, it is extensible in that it could be customized to cater for other languages. In that case, it would be necessary to change the target file type and adapt the searches to the corresponding language.

3.5. Step 5: Results Interpretation

On the basis of the results obtained via the bytecode analysis previously described, conclusions can be drawn, statistics can be presented, and good practices can be identified. It is important to note that it remains a challenge to extrapolate and generalize findings, as the results refer to the specific set of mobile applications in our experimental dataset. It can be indeed seen as an indicator of generic tendencies, but the stochastic nature of the process needs to be stressed.

4. Experimental Validation

Having described our methodology, we tested its applicability to a real case study concerning authentication protocols in IoT mobile apps. For this reason, we first collected an indicative set of Android applications and then investigated the implementation of authentication elements in two

different authentication protocols, namely OAuth and BLE. We selected the Android mobile operating system because of its open source nature and its high market share, but it should be noted that our methodology can be equivalently applied to other mobile operating systems.

4.1. Oauth 2.0 in Combination with Oidc Analysis Findings and Conclusions

The first analyzed protocol concerns the application layer and is the combination of OAuth 2.0 with OIDC. These protocols have been selected because their combination results in one of the most used standards for secure delegated access, as well as the multiple options for secure implementation that their frameworks offer, which are suitable for IoT mobile applications.

We detail in the following the different steps of our methodology in the context of OAuth 2.0 combined with OIDC.

4.1.1. Steps 1 to 3: Complete Mapping

We begin by selecting the security guidelines upon which our investigation will be based. In this report, the Smartphone Secure Development Guidelines [7] developed by ENISA will be taken as a reference. To complete the second step, we identify and map the features of the protocol that cover those guidelines. Finally, the specific implementation of each feature in the selected programming language (Java for Android in this case) is pinpointed.

The result of this whole process is shown in Table 2, where the three categories have been mapped and will serve as input for the following step.

Table 2. OAuth 2.0 and OpenID Connect (OIDC) features mapping. ENISA: European Union Agency for Network and Information Security.

ENISA Smartphone Secure Development Guidelines	**OAuth + OIDC Features**	**Bytecode Search**
Summary	*Summary*	*Element to find*
- Instead of passwords, consider using longer-term authorization tokens that can be securely stored on the device (as per the OAuth model) - Use the latest versions of the authorization standards (such as OAuth 2.0)	1. Use of OAuth 2.0 model, which uses tokens as the basis for authentication and authorization	StringBuilder oauth JSON (token)
- Use authentication that ties back to the end user identity (rather than only to the device identity). - Authentication should not be used as a replacement of authorization security controls	2. OpenID Connect on top of OAuth	scope=openid id_token
- Do not rely on client-side security controls. Both authentication and authorization controls should be implemented on the server side. - Tokens can be issued by the backend service after verifying the user credentials initially.	3. Use Authorization Code Grant	getQueryParameter (“code”)
- Use unpredictable session identifiers with high entropy	4. Protect against CSRF	getQueryParameter (“state”)
- Require authentication credentials or tokens to be passed with any subsequent request - Provide the ability to the mobile user to change passwords or other authentication tokens	5. Use of refresh tokens	refresh_token
- Do not reveal registered usernames and remove any fingerprint of their existence from verbose error messages - Tokens revocable by server	6. Use of OAuth standard error messages	invalid_request, invalid_token, inssuficient_scope
- Secure the tokens in transit	7. Proof Key for Code Exchange	code_challenge=abc code_verifier=1242
	8. Use of HTTP over TLS (HTTPS) in token transmissions	https://urlencodedpost.setHeader (“Authorization” + “Bearer”)
- Tokens are time bounded	9. Token frequently expires frequently	JSON (expires_in)

Considering for example features number 7 and 8, and in order to clarify the contents in Table 2, a more detailed explanation is provided.

1. The ENISA Smartphone Secure Development Guideline that refers to securing the tokens in transit is selected for the analysis.
2. The OAuth and OIDC features that address this guideline are explored. The first one refers to the use of Proof Key for Code Exchange (PKCE) to protect against the interception of authentication communications, and the second one refers to use of HTTPS in order to protect tokens from disclosure in storage and transport [17].
3. The implementation of the aforementioned two features in Android has to be specified. Regarding the PKCE feature, two query parameters are used to perform the protection, 'code_challenge' and 'code_verifier', so these will be the two searches in the application bytecode. Regarding the use of HTTPS in all the requests, Android provides several methods to set the parameters in the HTTP request, but the common elements are the "Authorization" request header field and "Bearer" request field value to select the Bearer authentication scheme to transmit the access tokenFirst item.

4.1.2. Step 4: Data Collection Strategy and Bytecode Analysis

- Strategy: Taking as input the bytecode search elements from the mapping shown in Table 2, the strategy of searches is built. The main objective is to validate the implementation of the OAuth protocol (and consequent use of tokens) and the OIDC on top of it. Moreover, a secondary objective refers to evaluating whether the application code considers the security options that the OAuth + OIDC framework provides.
- When it comes to the strategy, all the searches are independent from one another, as each of them represents a sole authentication measure and can exist without the others in the code. Nevertheless, several of the searches need to be considered sequentially in order to examine whether sets of bytecode elements have been implemented. Following the previous example of feature 8, we need to identify whether tokens are being exchanged securely. We first look for the 'Authorization' request header field, and within the same class that this string is found, we look for instances of 'Bearer'.
- Analysis: An initial sample of 325 IoT Android mobile applications from Google Play Store was gathered. While we do not claim to extrapolate and generalize findings based on this dataset, we still consider it representative in order to highlight indicative results. The results of the analysis are summarized in Table 3.

Table 3. Results from OAuth 2.0 and OIDC analysis.

Bytecode Element	**Occurrences**
Summary	*Summary*
Oauth	144
getQueryParameter + code	192
getQueryParameter + state	165
openid	170
code_challenge	5
code_verifier	6
code_challenge + code_verifier	1
Authorization + Bearer	94
JSON + token	85
JSON + expires_in	70

A first observation refers to the elements of 'code_challenge' and 'code_verifier' that were expected to run in equal numbers, as they are jointly used for the same purpose within the framework. However, they were only found once together in the sample applications, thus indicating a lack of proper OAuth and OIDC implementation.

Furthermore, the search of 'openid' was revealing, as all the occurrences were found in the same class within different applications. This class is actually the library that Google has created to provide delegated authentication for applications. This implies that the majority of developers decide to somehow outsource the authentication mechanisms to an established authentication service rather than having one of their own.

Exploring in further detail the applications where almost all of the elements were found (more specifically the top five ones), it was evident that there existed a diversity of implementations of the framework. OAuth is a framework, and hence there does not exist a reference API for developers who want to implement it. Thus, custom implementations of OAuth emerge, making it very hard to automate our methodology and find specific elements.

This fragmentation in OAuth implementations (lack of reference API) exacerbates concerns in regard to its secure implementation due to the lack of standardized methods to perform authentication actions. While this is not a hindering factor for secure authentication in itself, the provisioning of a secure reference API could greatly facilitate the proper use of secure authentication means by the average developer.

4.1.3. Step 5: Results Interpretation

The main conclusions after the analysis performed against an indicative set of IoT Android mobile applications include:

- Tendency of a mobile application developer to delegate authentication and authorization processes to well-established authentication and authorization services.
- Diversity of OAuth 2.0 framework implementations due to the lack of a reference API for developers, thus increasing fragmentation.

Standardization efforts regarding authentication protocols applied to mobile applications would definitely help solve these issues. Not even with OAuth—one of the most used authorization frameworks—was it possible to determine whether it had been implemented, not even mentioning whether the secure implementation of the protocol was performed, as there is no standard way to do so in Android mobile applications (i.e. standard APIs). This fact derives from the fragmentation of APIs and from custom ad hoc protocols' implementation in mobile applications. A possible approach to alleviate relevant concerns could **involve mobile operating systems** specifying a concrete and complete **API to serve as a reference** to developers who wish to implement their own authentication/authorization scheme in a **standard, interoperable, and verifiable manner.**

4.2. BLE Analysis Findings and Conclusions

The second protocol is BLE, which has emerged as one of the most widely used communication protocols for IoT devices. Due to the wide adoption of the protocol, it is very important to tackle the security measures it includes (authentication measures in our case) and evaluate them.

At the communication layer, we describe the main outputs of our methodology in the case of BLE.

4.2.1. Steps 1 to 3: Complete Mapping

First, the ENISA Smartphone Secure Development Guidelines [7] are mapped to BLE protocol features concerning authentication. Subsequently, the specific implementation of every feature in Java for Android has been defined. The result of this whole process is illustrated in Table 4, where the mapping of the three categories is presented.

Table 4. Bluetooth Low Energy (BLE) features mapping.

ENISA Smartphone Secure Development Guidelines	**BLE Features**	**Bytecode Search**
Summary	*Summary*	*Element to find*
- Using asymmetric cryptography for authentication and authorization purposes	1. Use of BLE secure pairing mechanisms	sect163k1
- Ensure that a strong password policy is being followed - Use unpredictable session identifiers with high entropy	2. Enforce restrictions when setting the pin	setPin() + !setPin(0,0,0,0,0,0)
- Require authentication credentials or tokens to be passed with any subsequent request	3. Make pairing setup expire frequently	cancelBondProcess()
- Use authentication that ties back to the end user identity (rather than only to the device identity) - Do not store any passwords or secrets in the application binary	4. Use of the Passkey method	extra.PAIRING_KEY + extra_PAIRING_VARIANT = PASSKEY

Considering, for example, feature number 1 and in order to clarify the entries in Table 4, we describe the methodology in more detail.

1. We first select the ENISA Smartphone Secure Development Guideline to be tackled, namely the one that encourages developers to use asymmetric cryptography when possible.
2. Based on the previous selection, features covering the specific guideline are sought. By examining the security specifications of BLE [26], we identify that the secure pairing implemented for BLE versions 4.2 and 4.3 uses asymmetric cryptography.
3. Finally, the implementation of this feature, i.e. secure pairing, in Android has to be pinpointed. In the case of Java for Android, a dedicated native ECDH Key Pair Generator exists, which is included in the package java.security.*, and corresponds to the parameter 'sect163k1'.

4.2.2. Step 4: Data Collection Strategy and Bytecode Analysis

- Strategy: It is worth mentioning beforehand the existence of a BLE reference API Guide for Android developers [28], which has been the basis for the searches in the code. In this case, all the target findings refer to the pairing process. Therefore, the first element to search is 'connectGatt()' API call, which belongs to the GATT Profile [29], and is a general specification in all current BLE application profiles. Thus, we discard all the applications that use another type of Bluetooth technology, and only work on the ones implementing BLE.

Since the pairing process has a crucial role when authenticating devices in BLE protocol [26], we focus our analysis on the corresponding features. The decision tree shown in Figure 5 can be summarized as follows.

We first check whether the GATT Profile has been used (if not, no more analysis is made on those apps). Then, we examine whether a pairing request is issued; if it is not, the reading and writing actions with the device are inspected (as it would mean interaction with the device is happening without a previous authentication phase). If the pairing request is issued, the type of pairing is examined; depending on the combination of the elements 'setPin()' and 'extra.PAIRING.KEY' and their use (e.g., whether the pin is set to all zeros), the pairing type could result to be Passkey, Just Works, or even none or undefined, if the process is not followed as the API specifies. By following this decision tree, results are gathered for further interpretation.

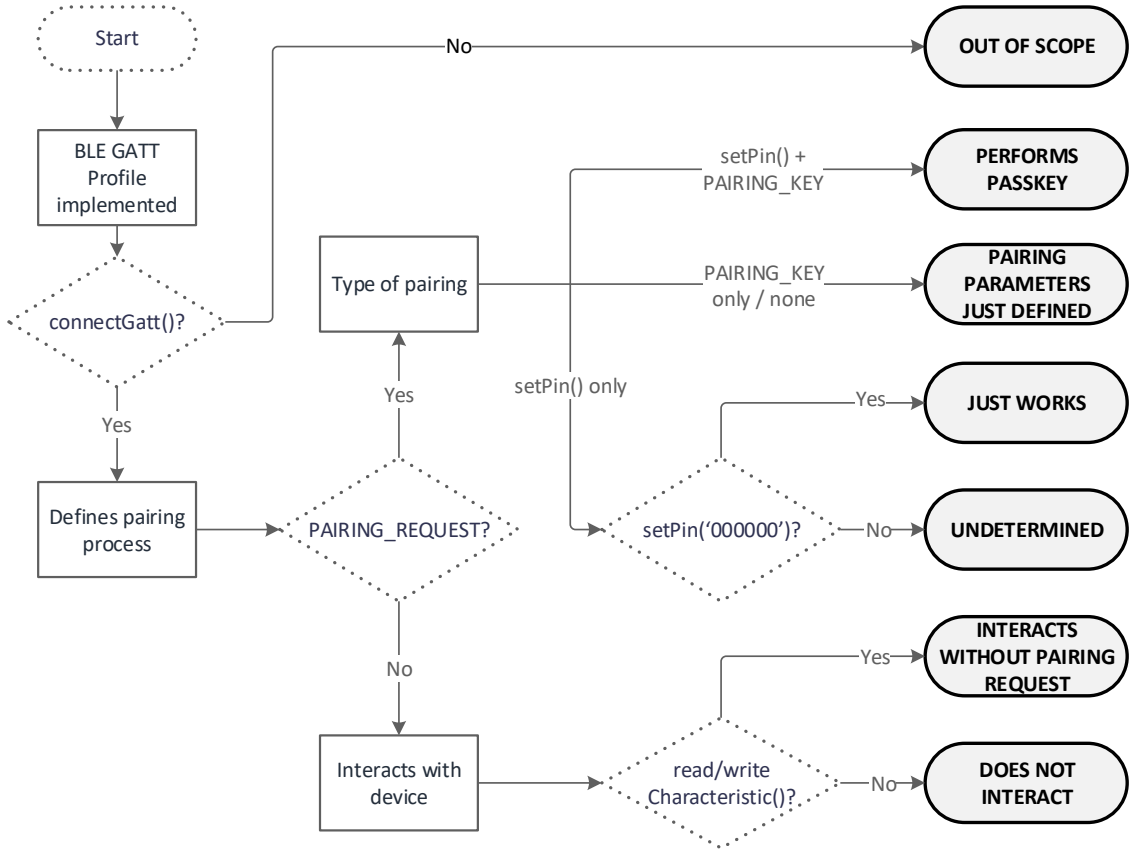

Figure 5. Searches decision tree.

- Analysis: For this analysis, a sample of 1022 IoT Android mobile applications from Google Play Store was gathered. These applications underwent the process defined in the decision tree shown in Figure 5, and were classified into the different output categories to determine their BLE pairing implementations. While we acknowledge the limited size of the dataset, these indicative findings can serve as input for further and more thorough investigation given adequate resources and time.

The results of the analysis are specified in Table 5. The first thing to highlight is the absence of occurrences when it comes to the category 'Undetermined' in Figure 5. Just two instances of only defining 'setPin()' (and not defining the 'PAIRING_KEY' parameter) were found, and both of them had set this pin to '000000'. This means that both of them implemented a Just Works type of pairing, which has been proven to be unsecure and vulnerable to eavesdropping during the whole communication time [26].

Table 5. BLE bytecode analysis results.

TOTAL APPLICATIONS: 1022				
Using Gatt: 534				*Not Using Gatt: 488*
Pairing	58	Not Pairing Requests	476	
Only defining pairing parameters	53	Communicating without requesting pairing	475	
Just Works	2	Just connecting	1	
Implements Passkey method	3			

Out of the 1022 analyzed IoT applications, only 52.25% were using a BLE GATT profile. Out of these 534 applications, 10.86% specified pairing parameters to perform pairing. From the remaining 476 applications that did not specify parameters for pairing, **a staggering 475** were reading/writing characteristics on the smart device, which means that operations were **performed without provable**

previous authentication (88.95% of the total using the GATT Profile). In our experimental dataset, 58 applications used a BLE GATT Profile and defined pairing parameters. Of these, 5.17% of them specified in a standard and verifiable way that authentication was incorporated to their processes by means of a Passkey pairing mechanism (a mere 0.56% of the total amount of applications using a BLE GATT Profile). Moreover, 3.45% implemented the Just Works pairing mechanism and the remaining 91.38% had indeed specified some pairing parameters, but were not making use of them.

4.2.3. Step 5: Results Interpretation

The general conclusions concerning the results of the analysis of BLE in Android mobile applications included:

- A limited number of applications were implementing pairing (authentication) mechanisms in a secure, complete, and verifiable way.
- There existed devices using BLE that allowed communication with smartphones without a previous secure pairing process.

One possible explanation for the absence of proper authentication mechanisms is the **assumed lack of necessity, or the risk assumption** (such as with a smart lightbulb), where the impact of the application being compromised is limited. Another possible justification could be the absence of capabilities in the smart device; however, even then, **additional efforts can be performed** to define a PIN (preferably different from all zeros). This way, even if they are vulnerable to sniffing/eavesdropping attacks, the PIN is harder to guess, and it would require more efforts to intercept communication.

5. Discussion

In this paper, we presented a systematic methodology that facilitates the evaluation of mobile applications and examines whether they include and properly implement authentication measures. In order to ensure consistency with existing security guidelines for the secure development of mobile applications, the proposed methodology drills down from high-level recommendations and guidelines toward specific code implementations that instantiate those good practices. Thus, the protocols undergoing this process can be evaluated in an implementation-systematic manner. This provides a unique way of linking policy guidelines and high-level security measures to specific source code elements. More importantly, the elements of the methodology (i.e. the mapping tables) can serve as concrete implementation good practices when it comes to high-level security guidelines, thus practically supporting IoT developers. In addition, the analysis tool effectively complements these mapping tables, moving from implementation toward evaluation processes; it gives the people responsible for evaluating security measures in mobile applications the possibility of doing so in an automated manner and against a set of such applications.

Our experimental evaluation results in the cases of OAuth/OIDC and BLE show that the implementation of authentication is fragmented and occasionally overlooked. This hinders the interoperable and verifiable character of the IoT mobile applications and poses a challenge if the smartphone is compromised; as a key element in the IoT environment, not only data within or accessible from the smartphone is in danger, but the command and control actions of cyber-physical IoT systems become susceptible as well. We aspire for the results of this work to raise awareness about the importance of the adequate implementation of authentication measures in the IoT world in general, and specifically in mobile IoT applications. Some actions that would promote a better level of cyber hygiene for such applications could be, for example, the creation of reference APIs regarding the authentication protocols/frameworks, which would bring homogeneity, promote interoperability, and help verify the suitability and correctness of their implementation. Furthermore, when these reference APIs are available, security by design principles can be incorporated.

Although meaningful conclusions were drawn out of this work, some limitations were encountered, which narrowed its extent and scope. Due to time and resource constraints, only two of the most

used protocols were analyzed, and only a limited set of authentication searches were performed. Further work could be oriented toward a wider variety of protocols, covering a wider set of security (not only authentication) measures, in order to build the bigger picture of authentication (and security potentially) in IoT mobile applications. Smartphones and their applications are commonly used as controllers of cyber-physical systems in the general IoT ecosystem; therefore, by securing them with methodological approaches such as the one presented in this paper, the security and safety of the entire IoT ecosystem is promoted.

Funding: There is no extra funding for this research.

Acknowledgments: I would like to thank my dear mentor Malatras, for his invaluable support and extraordinary help throughout this project, as well as Leguesse, for his wonderful collaboration.

Conflicts of Interest: This paper was produced as part of a traineeship contract at the European Union Agency for Network and Information Security (ENISA). Nevertheless, it is not part of the ENISA Work Program, and the project has been fully created, developed, and produced by the author.

References

1. Wolf, M.; Serpanos, D. Safety and Security of Cyber-Physical and Internet of Things Systems. *Proc. IEEE* **2017**, *105*, 983–984. [CrossRef]
2. European Union Agency for Network and Information Security (ENISA). *Baseline Security Recommendations for Internet of Things in the Context of Critical Information Infrastructures*; European Union Agency for Network and Information Security (ENISA): Athens, Greece, 2017.
3. Applied Cybersecurity Divsion (ACD)—NIST ITL. IoT Cybersecurity Considerations. Available online: https://www.nist.gov/itl/applied-cybersecurity/iot-cybersecurity-considerations (accessed on 1 June 2018).
4. Sa Silva, J.; Zhang, P.; Pering, T.; Boavida, F.; Hara, T.; Liebau, N.C. People-Centric Internet of Things. *IEEE Commun. Mag.* **2017**, *55*, 18–19. [CrossRef]
5. Dye, S.M.; Scarfone, K. A standard for developing secure mobile applications. *Comput. Stand. Interfaces* **2014**, *36*, 524–530. [CrossRef]
6. Open Web Application Security Project (OWASP). IoT Attack Surface Areas. Available online: https://www.owasp.org/index.php/IoT_Attack_Surface_Areas (accessed on 1 June 2018).
7. European Union Agency for Network and Information Security (ENISA). *Smartphone Secure Development Guidelines*; European Union Agency for Network and Information Security (ENISA): Athens, Greece, 2017.
8. Open Web Application Security Project (OWASP). OWASP Mobile Security Project. Available online: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Secure_M-Development (accessed on 1 June 2018).
9. Open Web Application Security Project (OWASP). IoT Framework Assessment. Available online: https://www.owasp.org/index.php/IoT_Framework_Assessment (accessed on 1 June 2018).
10. Internet Engineering Task Force (IETF). Design Considerations for Security Protocols in Constrained Environments. Available online: https://tools.ietf.org/id/draft-seitz-ace-design-considerations-00.txt (accessed on 1 June 2018).
11. Internet Engineering Task Force (IETF). Group Authentication. Available online: https://tools.ietf.org/id/draft-zhu-ace-groupauth-00.txt (accessed on 1 June 2018).
12. Internet Engineering Task Force (IETF). Authentication and Authorization for Constrained Environments (ace). Available online: https://datatracker.ietf.org/wg/ace/about/ (accessed on 1 June 2018).
13. Internet Engineering Task Force (IETF). Constrained RESTful Environments (core). Available online: https://datatracker.ietf.org/wg/core/documents/ (accessed on 1 June 2018).
14. oneM2M Partners. oneM2M—Standards for M2M and the Internet of Things. Available online: http://www.onem2m.org/ (accessed on 1 June 2018).
15. Ferrag, M.A.; Maglaras, L.A.; Janicke, H.; Jiang, J.; Shu, L. Authentication Protocols for Internet of Things: A Comprehensive Survey. *Secur. Commun. Netw.* **2017**, *2017*, 1–41. [CrossRef]
16. OASIS. Security Assertion Markup Language (SAML) Specifications. Available online: http://saml.xml.org/saml-specifications (accessed on 1 June 2018).
17. Internet Engineering Task Force (IETF). The OAuth 2.0 Authorization Framework. Available online: https://tools.ietf.org/html/rfc6749 (accessed on 1 June 2018).

18. OpenID. OpenID Specifications. Available online: http://openid.net/developers/specs/ (accessed on 1 June 2018).
19. International Organization for Standardization (ISO); International Electrotechnical Commission (IEC). *International Standard 29115: Information Technology—Security Techniques—Entity Authentication Assurance Framework*; International Organization for Standardization: Geneva, Germany, 2013.
20. Shikiar, A.; Lindemann, L. The Future of Authentication for the Internet of Things. 2017. Available online: https://www.slideshare.net/FIDOAlliance/the-future-of-authentication-for-iot (accessed on 1 June 2018).
21. Internet Engineering Task Force (IETF). Datagram Transport Layer Security Version 1.2. Available online: https://tools.ietf.org/html/rfc6347 (accessed on 1 June 2018).
22. Constrained Application Protocol (CoAP). CoAP—RFC 7252 Constrained Application Protocol. Available online: http://coap.technology/ (accessed on 1 June 2018).
23. Zigbee Alliance. Zigbee. Available online: http://www.zigbee.org/ (accessed on 1 June 2018).
24. Parekh, J. WiFi's Evolving Role in IoT. Available online: https://www.networkworld.com/article/3196191/lan-wan/wifi-s-evolving-role-in-iot.html (accessed on 1 June 2018).
25. Gomez, C.; Oller, J.; Paradells, J. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. Available online: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3478807/ (accessed on 1 June 2018).
26. Bluetooth SIG Proprietary. Security Overview. In *Specification of the Bluetooth System*; Bluetooth SIG, Inc.: Kirkland, WA, USA, 2014; Volume 1, Part A, Section 5.
27. Frew, J. (MakeUseOf). The Best Languages for Mobile App Development in 2016. Available online: https://www.makeuseof.com/tag/best-languages-mobile-app-development-2016 (accessed on 1 June 2018).
28. Android Developers. Bluetooth Low Energy Overview—Documentation. Available online: https://developer.android.com/guide/topics/connectivity/bluetooth-le (accessed on 1 June 2018).
29. Bluetooth SIG. GATT Overview. Available online: https://www.bluetooth.com/specifications/gatt/generic-attributes-overview (accessed on 1 June 2018).