I. J. Computer Network and Information Security, **2024, 2, 1-13**
Published Online on April 8, 2024 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2024.02.01

Vulnerability Detection in Intelligent Environments Authenticated by the OAuth 2.0 Protocol over HTTP/HTTPS

Gilson da Silva Francisco
IPT, Instituto de Pesquisas Tecnológicas, SP, Brazil
E-mail: gilsonsfn@gmail.com
ORCID iD: https://orcid.org/0009-0000-0012-5143

Anderson Aparecido Alves da Silva*
IPT, USP, SENAC, UNIP, SP, Brazil
E-mail: anderson@uol.com.br
ORCID iD: https://orcid.org/0000-0001-5426-6478
*Corresponding Author

Marcelo Teixeira de Azevedo
USP, Universidade de Sao Paulo, SP, Brazil
E-mail: marcelo.azevedo@pad.lsi.usp.br
ORCID iD: https://orcid.org/0000-0003-1676-7380

Eduardo Takeo Ueda
IPT, Instituto de Pesquisas Tecnológicas, SP, Brazil
E-mail: eduardoueda@ipt.br
ORCID iD: https://orcid.org/0000-0002-3776-961X

Adilson Eduardo Guelfi
UNOESTE, Universidade do Oeste Paulista, SP, Brazil
E-mail: guelfi@unoeste.br
ORCID iD: https://orcid.org/0000-0002-4636-1878

Jose Jesus Perez Alcazar
USP, Universidade de Sao Paulo,SP, Brazil
E-mail: jperez@usp.br
ORCID iD: https://orcid.org/0000-0003-3389-0401

Received: 08 March 2023; Revised: 19 October 2023; Accepted: 13 December 2023; Published: 08 April 2024

Abstract: OAuth 2.0 provides an open secure protocol for authorizing users across the web. However, many modalities of this standard allow these protections to be implemented optionally. Thus, its use does not guarantee security by itself and some of the deployment options in the OAuth 2.0 specification can lead to incorrect settings. FIWARE is an open platform for developing Internet applications of the future. It is the result of the international entity Future Internet Public-Private Partnership. [1,2] FIWARE was designed to provide a broad set of API to stimulate the development of new businesses in the context of the European Union. This platform can be understood as a modular structure to reach a broad spectrum of applications such as IoT, big data, smart device management, security, open data, and virtualization, among others. Regarding security, the exchange of messages between its components is done through the OAuth 2.0 protocol. The objective of the present work is to create a system that allows the detection and analysis of vulnerabilities of OAuth 2.0, executed on HTTP/HTTPS in an on-premise development environment focused on the management of IoT devices and to help developers to implement them ensuring security for these environments. Through the system proposed by this paper, it was possible to find vulnerabilities in FIWARE components in HTTP/HTTPS environments. With this evidence, mitigations were proposed based on the mandatory recommendations by the IETF.

Index Terms: Internet of Things (IoT), Smart Environments, FIWARE, OAuth 2.0, HTTP, HTTPS.

1. Introduction

A smart environment is a set of sensors working together to execute actions which make the life more productive, efficient and comfortable [3]. These are environments directly linked to the concepts of Internet of Things (IoT). Examples of smart environments include the smart cities, smart event monitoring, smart home/office and smart healthcare, among others [4].

In IoT architectures embedded within smart environments, it is not easy to achieve standardization due to the different types of applications and devices [4]. It is often necessary to use a middleware, which can be used to connect and manage all these heterogeneous components in a secure way [3, 5, 6]. In this scenario, an Application Programming Interface (API) plays a fundamental role in abstracting and providing integration between components [7]. API are computing interfaces offered to developers to access functions that facilitate the implementation of software and services, sharing code and enabling software reuse [8, 9].

There are examples in the literature that show the secure integration of smart IoT environments via API [10–12]. Also, there are some platforms in the literature that provide intelligent environments for the implementation of applications, such as Kaa [13], SOFIA [14], Snap4City [15] and CRYSTAL [14] that show integration with IoT environments. Among these, the FIWARE platform is a highlight, due to its ease of integration with API, compatibility with several applications and security resources [16, 17].

FIWARE is an open platform for developing Internet of the Future applications [1, 2] which uses a modular structure to reach a broad spectrum of applications such as IoT, big data, smart devices management, security, open data and virtualization, and others. It has a catalog of modules known as Generic Enablers (GE), selected by developers based on application needs [18, 19].

In an intelligent IoT environment, it is quite common for devices to be constrained, with few resources and dependent on compact power and network infrastructures [20, 21]. Therefore, despite having their own resources, in general, the devices take advantage of security features of the protocols they use, such as DTLS [22, 23], IPSEC [20, 24] and OAuth 2.0 [25, 26]. The advantage of this type of procedure is that the security built into the protocols is standardized and constantly updated. The disadvantage is that, as they are transparent to users, any vulnerabilities in these security features go unnoticed and can cause harm [27, 28]. An aggravating factor of this problem is that many smart environments are in constant development and often do not have all security controls enabled at the maximum level, precisely to ensure integration, performance and energy savings [29, 24]. Since they are development-oriented sites, essential security requirements, such as trustworthy [14], privacy [29], anomaly detection [24, 30], confidentiality and authentication [21], are often not met in these smart environments. All these facts motivated the application of this proposal in these scenarios.

Regarding security, FIWARE [31, 16, 2, 19] currently comprises three GEs (Keyrock, PEP Proxy Wilma, and AuthZForce) that have the role of offering identification, authentication, and authorization services. The exchange of messages between these GEs is based on the Oauth 2.0 framework, a delegation protocol, a means of ensuring that someone who controls a resource allows a software application to access that resource on their behalf [32, 33].

Although OAuth 2.0 is a framework focused on security, its use does not guarantee security by itself. If OAuth 2.0 is not correctly implemented, this causes critical vulnerabilities in applications, which in turn negatively impact companies [3, 25]. In a formal security analysis of OAuth 2.0, the work of [27] reveals some vulnerabilities such as 307 Redirect and AS Mix-Up. Both can be applied in HTTP environments, the latter also covering HTTPS. Another example is the Cross-Site Request Forgery (CSRF) vulnerabilities. The OAuth 2.0 standard emphasizes the danger of these vulnerabilities and notes that both the client and the server must implement protection against this type of threat [32, 33]. Unfortunately, many implementations of this protocol do not meet these specific warnings, leaving it as a suggestion to the developer. Although there are other vulnerabilities associated with OAuth 2.0, this paper seeks to precisely analyze and mitigate the action of these three cited threats (CSRF, AS Mix-Up, and 307 Redirect). The choice of their analysis is due to the scarcity of practical work on them, especially in smart environments – all three vulnerabilities occur in these environments. For example, the work of [34] performs a CSRF analysis, but it does not happen in a smart environment.

FIWARE has been widely used and provides authentication and authorization mechanisms within the proposed context, using OAuth 2.0 to enforce data access control. Some references [16, 2, 17, 21] highlight the importance of using the FIWARE GE related to security but do not explore the vulnerabilities arising from the poor implementation of OAuth 2.0.

The purpose of this paper is to create a system that allows detection and analysis of three categories of vulnerabilities in solutions that use OAuth 2.0 (CSRF, AS Mix-Up, and 307 Redirect), running over HTTP/HTTPS in an on-premise development environment, focused on managing IoT devices. As a secondary objective, it is intended to create an alerting system for found vulnerabilities.

2. Background

This section presents the main concepts to support and conceptualize this paper.

2.1. Oauth 2.0

OAuth 2.0 is a protocol created to solve the user's problem of passing their credentials to another application. In

other words, it was created to facilitate the development of applications that access data in other applications so that a common user can use the delegation of access. The OAuth 2.0 protocol standardizes how to obtain a token of access transparently for the user without managing the access tokens [32, 33].

There are four main actors in an OAuth 2.0 system: (1) clients; (2) resource owners; (3) Authorization Servers (AS); and (4) resource servers. OAuth 2.0 specifies multiple grant types for different use cases and a framework for creating new grant types. The most common grant types are authorization code grant, implicit grant, resource owner password credentials grant, and client credentials grant. The authorization code grant, detailed in Fig. 1, shows that when the user authorizes a client to access their data on a Resource Server, the client first redirects the user's browser to the AS. The user authenticates to the AS, providing' the login data, and then is directed back to the client with an AS-generated authorization code. After obtaining this code, the client accesses the AS endpoint responsible for generating the token and receives one token to be used as a credential to access user data in RS.

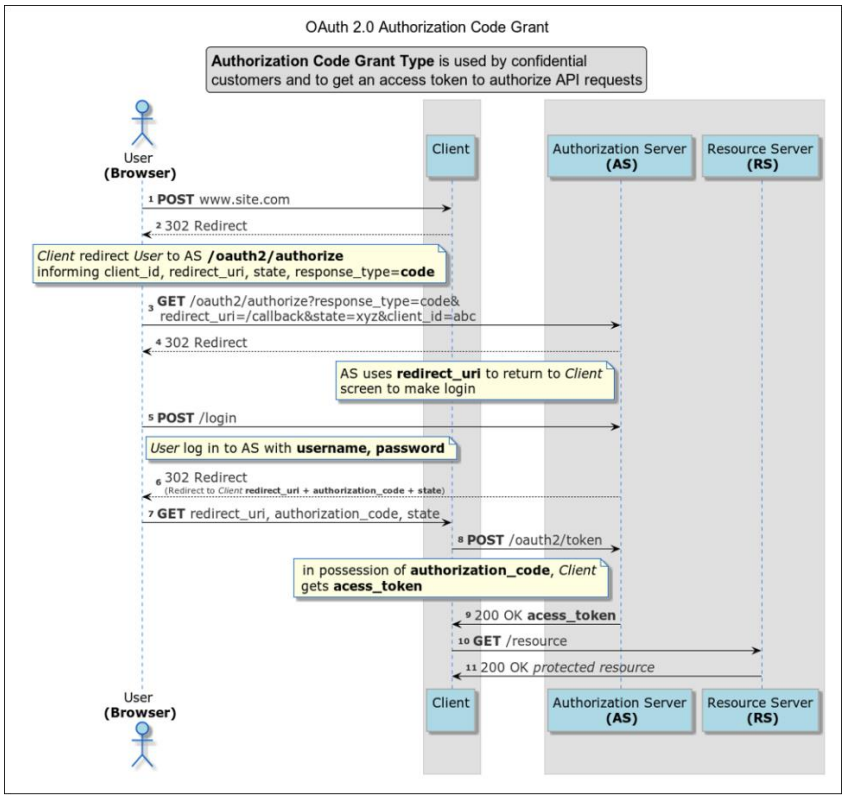

Fig.1. OAuth 2.0 authorization code grant

2.2. FIWARE

FIWARE is a platform that composes a framework of open-source components that can be assembled among themselves or with third-party platform components to accelerate the development of intelligent solutions [35]. Fig. 2 shows the interaction among some components of a FIWARE architecture based on the components used by the architecture proposed by this paper.

Component (1) is an Identity Manager (IdM) called Keyrock (GE) [26]. Using Keyrock, in conjunction with other security components such as Policy Enforcement Point (PEP) and Policy Decision Point (PDP), allows you to add OAuth 2.0-based authentication and authorization to your services and applications. Keyrock is essentially an AS OAuth 2.0 and supports authorization and authentication for the entire platform and client applications in the FIWARE infrastructure.

Component (2) is a Policy Enforcement Point (PEP) called Wilma (GE) [36]. Whenever a user tries to gain access to the resource behind the PEP proxy, the PEP Wilma describes the user attributes for PDP Keyrock to request a security decision and enforce a decision (allow or deny).

Component (3) of the architecture is a Context Broker (GE) called Orion. This is the central and mandatory component of FIWARE. It allows you to manage context information in a highly decentralized and large-scaled way. Orion Context Broker implements Publish/Subscribe Context Broker (GE), providing an interface Next Generation Service Interfaces (NGSI).

Component (4) of the architecture represents an IoT Agent (GE) called Backend Device Management (IDAS) (GE), which facilitates the interface with the Internet of Things (IoT), robots, and third-party systems to collect valuable context information or trigger actions in response to context updates.

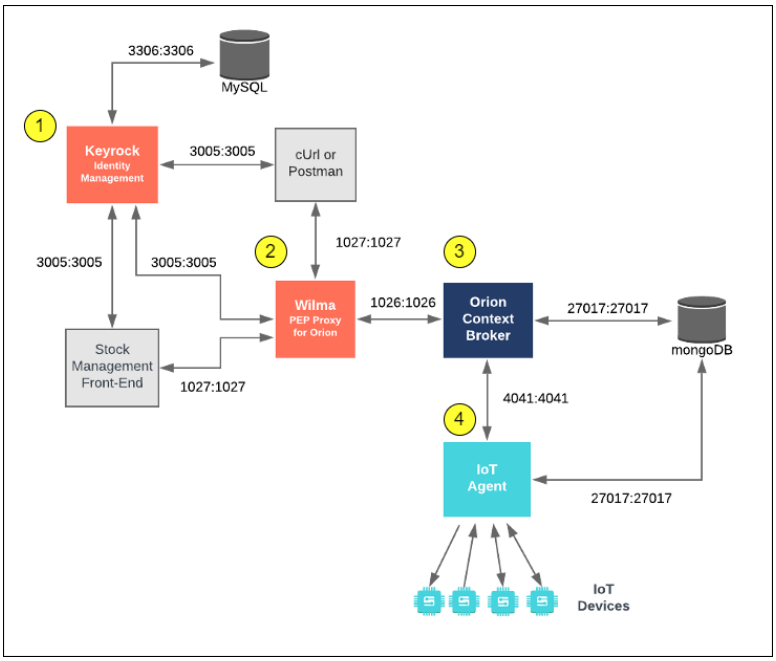

Fig.2. FIWARE architecture components

2.3. Oauth 2.0 Attacks

The OAuth 2.0 protocol has some documents with security considerations, like RFC 6819, 6749, and 8252 [32, 33, 37]. In addition to the reasons mentioned in the Introduction section, the choice of the analyzed vulnerabilities followed the following criteria: (1) CSRF is one of the most common attacks on the Internet and is also listed in the OWASP Top Ten [38]; and (2) as the work of [27] points out, the AS Mix-Up and 307 Redirect vulnerabilities are relatively recent, and many systems are still vulnerable to their effects. Next, are detailed the functioning of the three vulnerabilities/attacks and their possible mitigation schemes.

- CSRF: this attack happens in the context of continuous interaction between a web browser and a destination site. The attack involves a malicious website that uses the target browser to initiate an attacker's choice request to the target website. This can cause the destination site to perform actions without the user's participation [32, 33]. In particular, if the target user is currently connected to the site, the web browser sends cookies for the user containing a token authentication and the attacker's request. This way, the website processes the malicious request initiated by the user. [24, 32, 33] recommend that the client and the AS should use the parameter state in their requests to mitigate this attack.
- AS Mix-Up: in this attack, the attacker confuses the client about which AS the user chose to acquire an authentication code or token at the beginning of the authorization process. Then, that can be used to impersonate the user or access the user's data. This attack happens to OAuth 2.0 authorization code grant and OAuth 2.0 implicit grant type. To carry out the attack, the attacker manipulates the user's first request in such a way that the client thinks the user wants to use an identity managed by an attacker's AS (A-AS), while the user wants to use his identity managed by an honest AS (H-AS). As a result, the client sends the authorization code or the access token issued by the H-AS to the attacker. The attacker can then use this information to log in to the client with the user's identity or access the protected resources on the H-RS associated with the H-AS. To mitigate this vulnerability, an Internet Engineering Task Force (IETF) [39] was created that specifies precisely how an AS can (and should) include its issuer identifier (parameter iss) in the authorization response. The AS must add a parameter to identify the issuing AS (iss) in the responses, and clients must extract the value of this parameter from the responses if it is present. Clients must also compare the extracted and URL-decoded value with the authorization server's issuer identifier where the authorization request was sent [40].
- 307 Redirect: in this attack, the attacker (running a client) discovers the user's credentials when he logs in to an AS that uses the wrong HTTP redirect status code [27]. In RFC 6749 and 8252 [32, 33], the HTTP status code 302 is used for this purpose, but any other method is allowed. However, when HTTP status code 307 is used for redirection, the user's agent sends the form data (user credentials) via HTTP POST to the client, as this status code does not require the user's agent to rewrite the POST request into a GET request (discarding the form data in the body of the POST request). This way, if the relying party is malicious, it can use the credentials to impersonate the user in the AS. To mitigate this problem, HTTP status code 303 [39] should be used.

2.4. Related Works

This subsection analyzes some works that address issues related to the implementation of OAuth 2.0 and its vulnerabilities in environments that use the HTTP/HTTPS protocol in different scenarios.

The work of [27] shows that OAuth 2.0 is a widely deployed authorization protocol and serves as the basis for OpenID Connect. OpenID Connect behaves as an authentication layer and allows clients to verify user identity based on an Authorization Server (AS), in addition to providing an essential structure for obtaining user profile data. This work is based on abstract models, and the authors focus mainly on the analysis of specifications. In this way, refined implementation details are ignored. Leaving out of scope the analysis regarding security issues caused by practical implementation failures. Another issue is that, in general, developers are unaccustomed to use formal modeling concepts at work.

Work [25] shows the use of OAuth 2.0 by many applications aimed at FinTechs for authorization purposes. It shows that some of the developments do not correctly implement the OAuth 2.0 specifications. This creates critical vulnerabilities that end up having negative impacts on the companies. The authors focus on OAuth 2.0 implementation vulnerabilities made by developers. Developers often misinterpret the requirements needed for secure implementation. However, these vulnerabilities are common, and the authors show the possible flows where these are found but do not propose a test scenario with real or fictitious data.

The authors of the paper [34] shows that OAuth 2.0 provides an open framework for authorizing users on the web, and the standard enumerates mandatory security protections for various attacks. However, many modalities of this standard allow these protections to be implemented optionally. They analyzed the extent to which a particularly dangerous vulnerability such as CSRF exists in real-world deployments. The authors scanned an average of 10,000 domains. They found that 25% of the sites using OAuth 2.0 are suspected to be vulnerable to CSRF attacks. In this same work, the authors analyzed CSRF attacks in various domains, discovered and reported attacks in OAuth 2.0 implementations, however, they did not delve into other types of known vulnerabilities in OAuth 2.0, like AS Mix-Up Attack. Another limitation of the work [34] is that the authors did not address the weaknesses found in smart environments, such as the FIWARE environment.

In [41] the authors shows the importance of OAuth 2.0 security. This subject has been analyzed theoretically, using formal methods and practically looking at OAuth 2.0 implementations. Considering that OAuth 2.0 is widely adopted by Identity Providers (IdPs) around the world, the work of [41] covers only a few points of real-world OAuth 2.0 implementations that are potentially vulnerable to attacks from Partial Redirection URI Manipulation (PRURIM) and do not look at smart solutions nor for other OAuth 2.0 vulnerability types such as AS Mix-Up Attack.

The increasing visibility of OAuth 2.0-related vulnerabilities and its large-scale real-world implementations motivate the authors' work [42]. They argue that many studies rely on manual analysis to discover new vulnerabilities in OAuth 2.0 implementations or run automated tests for a limited set of specific and previously known vulnerabilities.

The work of [16] reports a study to discover the feasibility and suitability of the FIWARE platform for smart cities. This study is conducted in stages and generates a permission system using FIWARE's built-in security mechanism in its implementation. Security is a key issue in Smart cities applications, so it is an eliminatory criterion if it cannot be met. However, the author stated that in his work that no performance or load tests were performed. In addition, it also did not provide a method that allows the analysis and detection of OAuth 2.0 vulnerabilities. Another limitation of the work is that the authors did not work with the HTTPS protocol.

The work of [17] shows that Secure Electronic Identification (eID) is one of the main enablers of data protection, privacy, and online fraud prevention, especially in new application areas such as smart cities, where embedding real identities into trusted infrastructures has enormous potential. However, the authors admitted that the proposed model could be improved by supporting compatibility with other known standards, such as OpenID Connect. Also, integrating clients that support JSON Web Token (JWT) or other authentication standards can be very useful. Another limitation of the work is that the authors did not work with the HTTPS protocol.

Table 1 compares the various proposals that serve as the basis for this paper to clarify the contribution offered.

Table 1. Comparison of related work

Papers	A	B	C	D	E	F	G
Felt et al. (2016)	No	1	Client, AS and RS	Formal	2	Yes	HTTP/HTTPS
Gocher e Bahtiyar (2019)	No	2	AS, Client	Formal	2	No	HTTP/HTTPS
Shernan et al. (2013)	No	1	AS, Client	Practice	2	Yes	HTTP/HTTPS
Li et al. (2018)	No	1	Client, AS and RS	Formal	2	Yes	HTTP/HTTPS
Ronghai et al. (2016)	No	1	AS, Client	Practice	2	Yes	HTTP/HTTPS
Salhofer (2018)	Yes	-	AS, Client	Practice	1	No	HTTP
Alonso et al. (2019)	Yes	-	AS, Client	Practice	1	No	HTTP
This paper	Yes	2	Client, AS and RS	Practice	1	Yes	HTTP/HTTPS

Table 1 criteria:

- Uses smart environments: the first point to be analyzed is the environment in which the work is implemented, in this case, intelligent environments are used, enabled by IoT components or other environments.
- Focus of OAuth 2.0 Vulnerability Analyzes: the works compared bring two possible focuses of the analysis of vulnerabilities in OAuth 2.0: (1) specification, an analysis made based on the protocol documentation; and (2) implementation, based on the details of the developer's implementation, where fine details can be ignored.
- Focus on Vulnerable OAuth 2.0 components (Client; AS; RS): each component composes the OAuth 2.0 protocol is susceptible to vulnerabilities. This topic looks at which components the related works focused their analysis on.
- Analysis methodology: the methodology is an essential point for the analysis. Formal analysis is very efficient and capable of finding unknown vulnerabilities. However, practical analysis is closer to implementation and can analyze fine details that the developer often overlooks.
- Data (Simulation; Real): data can be real or simulated. Some works use real data in their analysis and get real-world answers. In an environment with simulated data, it is possible to assess the poor implementation of the protocol and assess potential problems beforehand. This paper uses an environment with simulated data, generated from a component (Simulator) of the proposed architecture, to show developers possible gaps that may occur if they forget some detail in the implementation.
- Method for analyzing and detecting OAuth 2.0 vulnerabilities: Not all related works have proposed a method for analyzing and detecting OAuth 2.0 vulnerabilities. In this context, a method can assess possible vulnerabilities in architecture through a system or framework. Through a method, it is possible to anticipate possible unsafe implementations.
- Communication protocol (HTTP; HTTPS): some works did not use the HTTPS protocol in their research. The works that used smart environments only did their analysis in unsecured HTTP environments. This point is crucial as many OAuth 2.0 vulnerabilities are addressed when HTTPS is used. Thus, as performed in our proposal, an analysis of exposure that affects HTTPS grows in importance and can be instrumental in generalizing to other environments. Among the differentials of our proposal concerning related works is the analysis of OAuth 2.0 vulnerabilities (CSRF, AS Mix-Up, and 307 Redirect) with HTTP and HTTPS communication protocols in a smart environment composed of FIWARE components.

3. Conceptual Architecture of the Proposed System

This paper proposes to create a system for detecting and analyzing three categories of OAuth 2.0 vulnerabilities, running over the HTTP/HTTPS protocol in an on-premise development environment focused on managing smart environments. Fig. 3 illustrates the proposed system development.

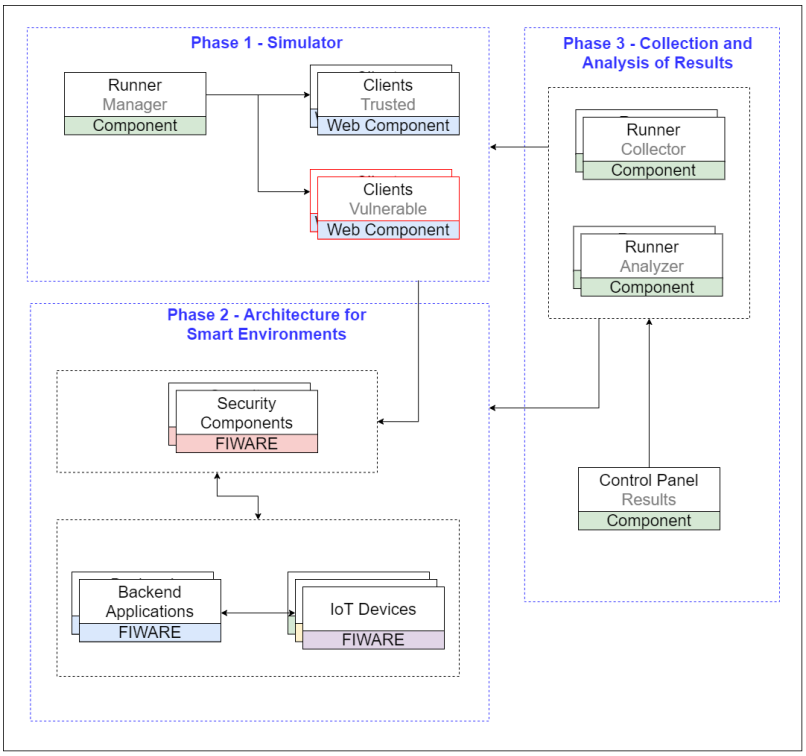

Fig.3. Conceptual architecture of the proposed system

Phase 1 is responsible for generating mass for future or real-time collection and analysis of results. It is possible to create trusted and vulnerable clients for analysis. At Phase 2, a test environment is created consisting of components responsible for generating data for analysis and conclusions. Phase 3 is responsible for the collection and analysis of data from the experiment.

3.1. Architectural Structure

For validation of the proposed scenarios, two architectures were created to test different scenarios. The communication of the first architecture is based on the HTTP protocol. Here you can test the OAuth 2.0 flow Authorization Code and simulate the vulnerabilities presented in this paper: CSRF, 307 Redirect Attack e AS Mix-Up. The communication of the second architecture is based on the HTTPS protocol. In this architecture, it is possible to show that, even in an end-to-end protected communication environment, it is possible to simulate the AS Mix-Up vulnerability when the OAuth 2.0 implementation does not follow the mandatory recommendations by the IETF [39].

These two architectures allow two types of executions: : (1) synchronous and (2) asynchronous mode. In synchronous mode an user can check the requests and responses executed by a specific flow. In this mode, the system allows the user to choose which stream to analyze: a normal OAuth 2.0 stream, a malicious AS Mix-Up stream or a CSRF attack. In asynchronous mode, the system randomly applies, every 10 seconds, one of two vulnerabilities (AS Mix-Up or CSRF) in a normal OAuth 2.0 stream. It is a way of testing the accuracy of the system in detecting vulnerabilities, since the user is not sure if the stream being sent is normal, AS Mix-UP or CSRF.

To create a virtualized network we use the Docker platform, the Java programming language and the Spring Boot framework to implement the system components.

The HTTP architecture is split into two networks as shown in Fig. 4: (1.a) local network where the OAuth Manager component is installed, and (1.b) Docker network that contains the components participating in the test simulation.

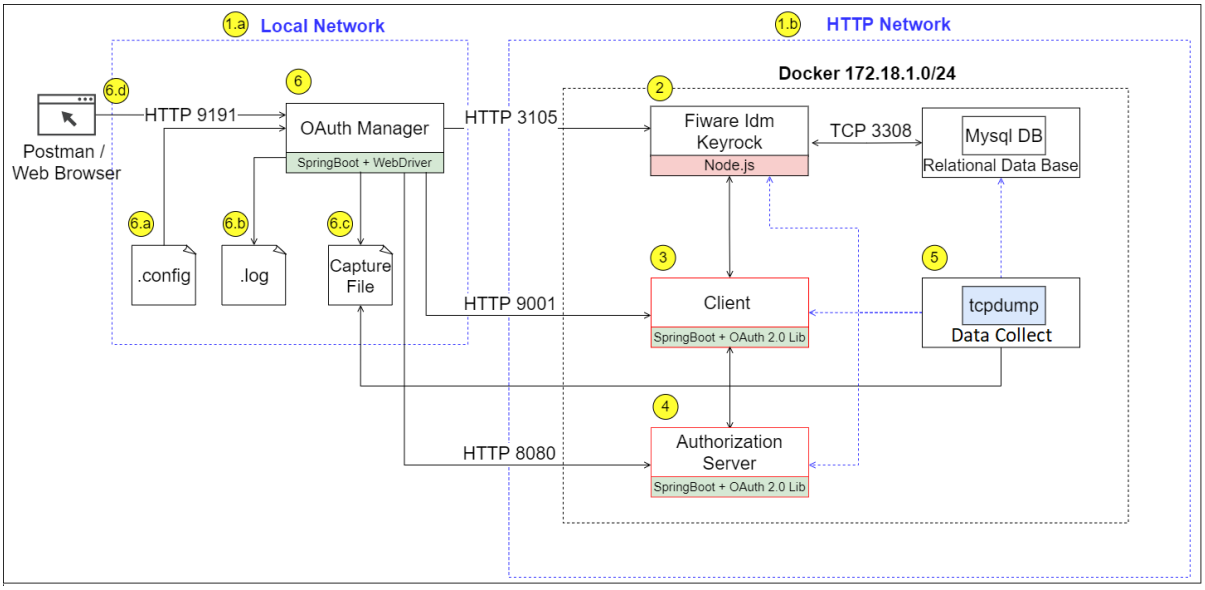

Fig.4. HTTP experiment architecture

The Docker network used in this application is composed of the following components:

- Keyrock: an AS implementation for FIWARE, created in node.js. Its image is available on DockerHub. It has a dependency on a relational database, in this case, MySQL, responsible for keeping all the necessary data to guarantee the authentication and authorization of registered users.
- Client: a Java Spring Boot application capable of simulating an OAuth 2.0 client, that is, it is responsible for maintaining a connection with a previously registered AS.
- AS Vulnerable: this component plays the role of an AS with some vulnerabilities (CSRF, AS Mix-Up, and 307 Redirect) analyzed in this paper.
- TCPDump: sniffer responsible for collecting all interaction between components within the Docker network.
- OAuth Manager: The only local network component. It is responsible for all the configuration logic, analysis, and data capture. During analysis, data is available in real-time for viewing through a control panel which points out possible vulnerabilities and how to mitigate them (Fig. 5).

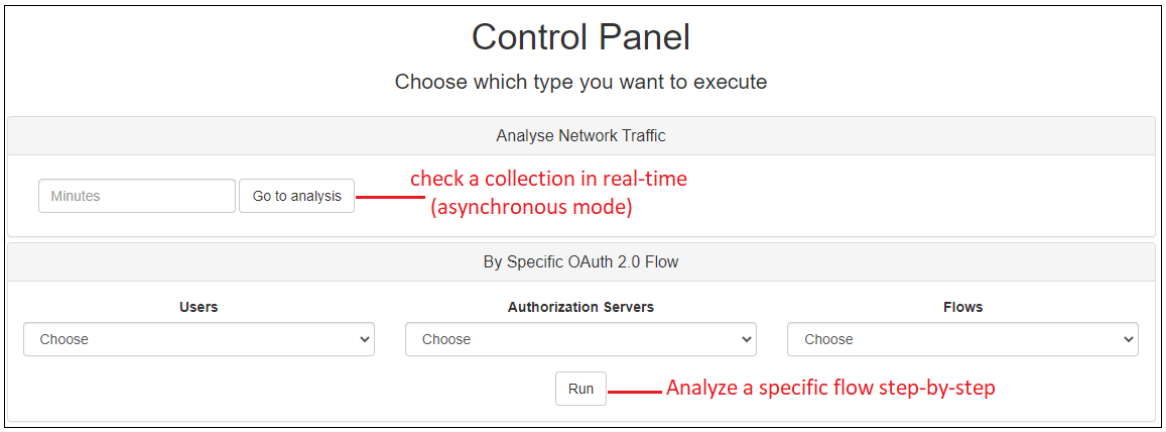

Fig.5. Control panel

As shown in Fig. 6, the architecture of the HTTPS experiment is divided into two networks: (1.a) local network where the OAuth Manager component is installed, and (1.b) Docker network that contains the components participating in the test simulation.

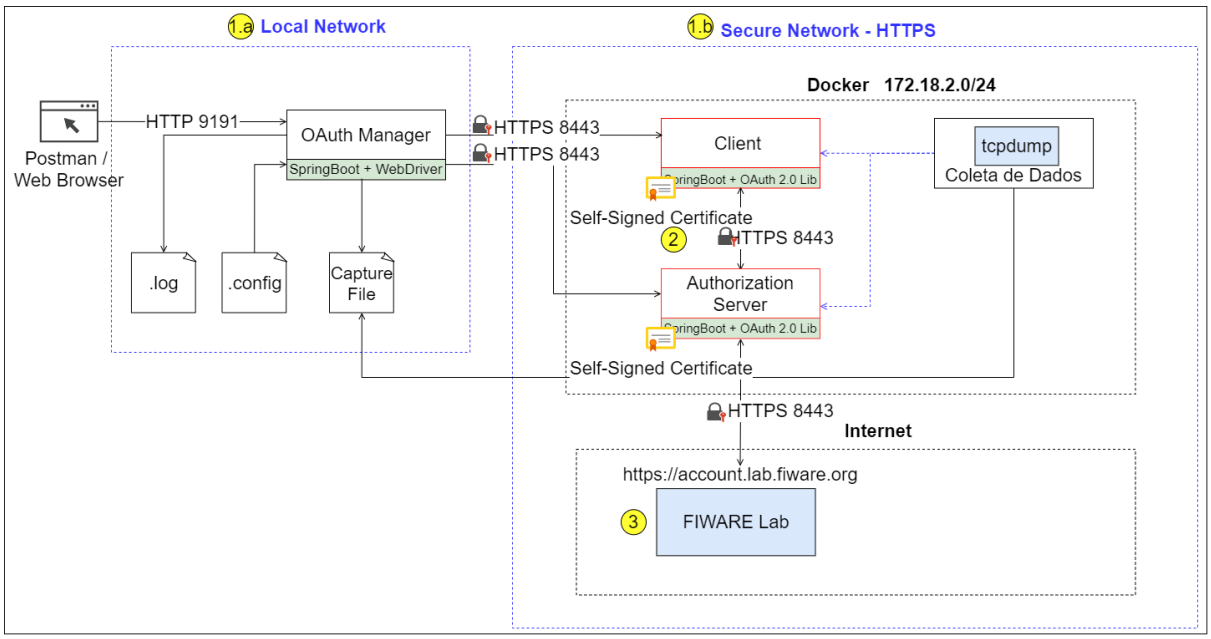

Fig.6. HTTPS experiment architecture

The local network is the same as used in the HTTP architecture. However, in the network created in Docker, all participants talk through the HTTPS protocol. The Docker network created for this application is composed of the following components:

- Vulnerable Client and AS using HTTPS: a Java Spring Boot application capable of simulating an OAuth 2.0 client and AS.
- FIWARE Lab: has the client registration for testing. In this experiment, the client and the AS are kept in the same application. However, they talk through secure endpoints using the HTTPS protocol. To ensure end-to-end security, HTTPS support is enabled in the application. For this, the key tool from JAVA is to generate a self-signed certificate in PKCS12 format.

3.2. Test Scenario Results

The purpose of the proposed application is to detect and analyze three categories of vulnerabilities (CSRF, AS Mix-Up, and 307 Redirect) in solutions that use OAuth 2.0 and run HTTP/HTTPS in an on-premise development environment focused on managing IoT devices. At this moment, all tests are done synchronously using the proposed system. It is important to plot percentages that indicate how vulnerable systems are to the attacks discussed in this paper. This allows the prioritization of security controls according to the environment used.

The work of [34] has tested 5.6 million URLs and has shown that 25% of them (1.4 million) were vulnerable to the CSRF attack.Within this set of affected sites, the authors identified that 20% (280,000) of these vulnerable addresses were connected to a Facebook API, demonstrating the extent that this flaw can reach. The data raised by [34] also showed, in an additional analysis with 180 vulnerable URLs, that 81% of them used the HTTPS protocol. More recent studies, such as [42] in 2016, identified 61% (405) of sites vulnerable to CSRF (among the top 500 Chinese and American sites), while the survey carried out by [41] presented a percentage of 39% for this same threat.

According [41], analysis of the 1.000 top-ranked websites showed 137 supporting Google sign-in, with 15% (21) of them vulnerable to Mix-up attack.

Finally, the work of [43] showed that within a set of shortened URLs, 84% of them responded with 30X redirection codes, while in URL shortening services this percentage has achieved 20%.

Table 2 shows the scenarios of experiments, results of synchronous tests, and vulnerability percentages.

Table 2. Synchronous tests experiments and results

ID Test scenario	User	Authorization Server	Attack	Communication Protocol	Result
\#1	Alice	AS Vulnerável	CSRF	HTTP	Vulnerable
\#2	Charlie	Keyrock	CSRF	HTTP	Not Vulnerable
\#3	Gil	FIWARE Lab	CSRF	HTTPS	Vulnerable
\#4	Alice	AS Vulnerável	AS Mix-UP	HTTP	Vulnerable
\#5	Bob	Keyrock	AS Mix-UP	HTTP	Vulnerable
\#6	Gil	FIWARE Lab	AS Mix-UP	HTTPS	Vulnerable
\#7	Alice	AS Vulnerável	307 Redirect	HTTP	Vulnerable
\#8	Charlie	Keyrock	307 Redirect	HTTP	Not Vulnerable
\#9	Gil	FIWARE Lab	307 Redirect	HTTPS	Not Vulnerable
CSRF: vulnerability between 25% - 61% of sites [34, 41, 42] As Mix-UP: 15% vulnerability [41] 30X redirect: 84% vulnerability in shortned URL and 20% in shortned services [43]					

All scenarios are executed sequentially. The last column (Results) of Table 2 has a binary proposal and considers the difficulty of exploiting the threat in two ways: (1) "Vulnerability" means that the system is entirely flawed; while (2) "Not vulnerable" means that the system is 100% safe.

The system proposed in this paper works through the control panel (Fig. 5), which allows the choice and execution of the specific OAuth 2.0 flow and the step-by-step analysis of the Authorization Code Grant. For analysis purposes, nine test scenarios were run involving three users, three AS, three vulnerability categories, and two different environments: HTTP e HTTPS. The execution of the first three scenarios represents the attack with the vulnerability.

In scenario #1, the user Alice chooses the Vulnerable AS to obtain the access token through the HTTP protocol. This AS is not prepared for a CSRF attack, so the vulnerability is found in this scenario.

In scenario #2, user Charlie chooses Keyrock to get the access token via HTTP protocol. The test detects that this AS is prepared for a CSRF attack and does not return the authorization code when the client does not inform the state parameter in the redirect URL.

In scenario #3, user Gil chooses FIWARE Lab to obtain the access token through HTTPS protocol. This AS is not prepared for a CSRF attack. Therefore, the finding is that this is a vulnerable scenario. In these first three scenarios, it is clear that the system works and indicates that Keyrock is prepared for this vulnerability category. On the other hand, vulnerable AS and FIWARE Lab are prone to CSRF attacks. The attack on Vulnerable AS was already expected (since it was created for that). In FIWARE Lab, the vulnerability returns the authorization code even when the client does not inform the state parameter In the redirect URL.

The following three scenarios run for the attack on the AS Mix-Up vulnerability. In scenario #4, the user Alice chooses the Vulnerable AS to obtain the access token through the HTTP protocol. This AS is not prepared for an AS Mix-Up attack, so this scenario is vulnerable.

In scenario #5, user Bob chooses Keyrock to get the access token through the HTTP protocol. As this AS is not prepared for an AS Mix-Up attack, the scenario is marked as vulnerable.

In scenario #6, user Gil chooses FIWARE Lab to obtain the access token through HTTPS protocol. This AS is not prepared for an AS Mix-Up attack, so this scenario is also vulnerable.

Note that vulnerabilities are found in scenarios #4, #5, and #6 that perform the AS Mix-Up attack. Susceptibility to Vulnerable AS was expected because this AS was created without this protection. Keyrock and FIWARE Lab Authorization Servers have shown that they are vulnerable to this attack as they do not implement the necessary precautions. Following the recommendations [21], they should include their respective issuer identifiers (parameter iss) in the authorization response, and clients should extract the value of this parameter from the responses when it is present. With this measure, the client guarantees that it makes requests to the same AS that started the flow. Another important point is that in scenario #6, where FIWARE Lab is used in an end-to-end HTTPS network, even if the data travels in a

secure environment, the flow is prone to vulnerability if the recommendations are not followed.

The last three scenarios run for the 307 Redirect vulnerability attack. In scenario #7, the user Alice chooses the Vulnerable AS to obtain the access token through the HTTP protocol. This AS is not prepared for a 307 Redirect attack and is marked as vulnerable.

In scenario #8, user Charlie chooses Keyrock to get the access token via HTTP protocol. The test detects that this AS is prepared for a 307 Redirect attack and does not return HTTP code 307 in redirect URLs.

In scenario #9, user Gil chooses FIWARE Lab to obtain the access token through HTTPS protocol. The test detects that this AS is prepared for a 307 Redirect attack and does not return HTTP code 307 in redirect URLs.

In the last three scenarios, it turns out that the only AS that is prone to the 307 Redirect attack is the Vulnerable AS (created for this). Both Keyrock and FIWARE Lab are prepared for this vulnerability category and do not return HTTP 307 code in their redirect URL. After running the synchronous tests, it was possible to identify that CSRF and AS Mix-Up vulnerabilities occur even in an environment that transfers its data in a secure protocol with end-to-end security such as HTTPS.

The proposed system also alerts about the vulnerabilities found in the HTTP architecture in real-time. The scenario is executed asynchronously, for this, the system proposed in this paper is used through the Control Panel, and the chosen execution option is Analyze Network Traffic. In this run, it is possible to analyze network traffic in real-time to find potential vulnerabilities. Data capture only covers the components present in the HTTP architecture. The proposed system does not capture all HTTPS architecture data. In fact, the system captures data within a Docker environment. As FIWARE Lab is an external system and is exposed on the Internet, its data is neither captured nor displayed in Analyze Network Traffic.

Table 3 shows the results of the asynchronous test. It is possible to observe the number of executed OAuth 2.0 Authorization Code Grant flows, how many of these flows suffered CSRF attacks, and how many suffered AS Mix-Up attacks.

Table 3. Asynchronous test results

Option Analyze Network Traffic	
Execution time	10 minutes
OAuth 2.0 Authorization Code Grant	59 no attacks
CSRF Attack	17 attacks
AS Mix-Up Attack	20 attacks

It is worth mentioning that the number of attacks 307 Redirect is not present in Table 3. The proposed system does not perform this attack in a parameterized way. Identification of this attack is verified in the communication URLs among the components of the architecture.

The information composed in the results is obtained from the file of log of application control. This log file makes it possible to overview everything that happened in a specific scenario delimiting the beginning and end of execution. The Control Panel makes it possible to find vulnerabilities online and offers links to possible fixes that mitigate each of the analyzed vulnerabilities. Based on the results, it is possible to understand that the proposed system, and with minor adjustments, it can be used in other environments that use OAuth 2.0.

Both synchronous and asynchronous tests proved that the proposed system is able to identify the three exploited attacks efficiently and suggest security measures with 100% effectiveness.

4. Conclusions

Based on the fact that some security implementations suggested in the OAuth 2.0 standard are optional and often fail to be implemented, in this paper we present the proposal of an application to detect and handle three specific vulnerabilities of the OAuth 2.0 protocol: CSRF, AS Mix-UP and 307 Redirect.

As a proof of concept, we chose to carry out experiments on a smart platform, focused on the use of smart devices Since these environments are often experimental, they often lack the implementation of all recommended security measures. This is one of the differentials of the work, since few vulnerability studies of the OAuth 2.0 protocol are carried out in these smart environments. Thus, the proposed system was implemented in FIWARE, a platform aimed at the use of smart IoT devices, which has a series of ready-made APIs and a set of applications (GE) focused on security.

In addition to real-time detection of vulnerabilities, performed by analyzing real-time network traffic, the proposed system suggested ways of mitigating each threat studied in both the HTTP and HTTPS protocols. The results obtained were quite satisfactory, as the system was able to analyze, identify and treat vulnerabilities in nine different scenarios. It is worth noting that the chosen scenarios were quite comprehensive as they analyzed the effectiveness of the system at least at three different points: (1) in the AS; (2) in the client; and (3) in the RS. In 10 minutes, the real-time tests identified 96 vulnerabilities, and in all scenarios both the identification and the suggested solution worked with 100% efficiency. In fact, the result of the work was extended and identified flaws that can be committed in the FIWARE implementation, such as vulnerabilities in the Keyrock component, for example. These flaws can cause security problems when using

OAuth 2.0, even when the HTTPS protocol is used.

In addition to the intelligent environment used, another differential of the proposal, in relation to the referenced research, was the practical implementation of the experiment, whose source code is available at https://github.com/gilsonsf/oauth-fiware.

The fact that the system analyzes only three vulnerabilities can be considered a limitation of the system and can be improved in further research.

Acknowledgments

The technical support of the partnership between Huawei and Universidade de São Paulo (USP).

References

[1] (FI-PPP), F. I. P.-P. P. FI-PPP: FutureInternet Public Partnership. 2019. Disponível em: <https://ec.europa.eu/digital-single-market/en/future-internet-public-private-partnership>.

[2] PONOMAREV, K.; NISSENBAUM, O. Attribute-based encryption with authentication provider in fiware platform. In: IEEE. 2018 Dynamics of Systems, Mechanisms and Machines (Dynamics). [S.l.], 2018. p. 1–5.

[3] ELRAWY, M. F.; AWAD, A. I.; HAMED, H. F. Intrusion detection systems for iot-based smart environments: a survey. Journal of Cloud Computing, Springer, v. 7, n. 1, p. 1–20, 2018.

[4] ULLO, S. L.; SINHA, G. R. Advances in smart environment monitoring systems using iot and sensors. Sensors, Multidisciplinary Digital Publishing Institute, v. 20, n. 11, p. 3113, 2020.

[5] SETHI, P.; SARANGI, S. R. Internet of things: architectures, protocols, and applications. Journal of Electrical and Computer Engineering, Hindawi, v. 2017, 2017.

[6] KARIE, N. M. et al. A review of security standards and frameworks for iot-based smart environments. IEEE Access, IEEE,2021.

[7] SIRIWARDENA, P. Advanced API security: OAuth 2.0 and beyond. [S.l.]: Springer, 2020.

[8] OFOEDA, J.; BOATENG, R.; EFFAH, J. Application programming interface (api) research: A review of the past to inform the future. International Journal of Enterprise Information Systems (IJEIS), IGI Global, v. 15, n. 3, p. 76–95, 2019.

[9] GMBH, T. O. B. P. Open Banking Project. 2020. Disponível em: <https://www.openbankingproject.com>.

[10] CRUZ-PIRIS, L.; RIVERA, D.; VEGA-BARBAS, M. Methodology for massive configuration of oauth 2.0 tokens in large iot scenarios. In: IEEE. 2020 16th International Conference on Intelligent Environments (IE). [S.l.], 2020. p. 5–12.

[11] EMERSON, S. et al. An oauth based authentication mechanism for iot networks. In: IEEE. 2015 International Conference on Information and Communication Technology Convergence (ICTC). [S.l.], 2015. p. 1072–1074.

[12] OH, S.-R.; KIM, Y.-G. Afaas: Authorization framework as a service for internet of things based on interoperable oauth. International Journal of Distributed Sensor Networks, SAGE Publications Sage UK: London, England, v. 16, n. 2, p. 1550147720906388, 2020.

[13] RASYID, M. U. H. A.; MUBARROK, M. H.; HASIM, J. A. N. Implementation of environmental monitoring based on kaa iot platform. Bulletin of Electrical Engineering and Informatics, v. 9, n. 6, p. 2578–2587, 2020.

[14] FERRY, N. et al. Enact: Development, operation, and quality assurance of trustworthy smart iot systems. In: SPRINGER. International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. [S.l.], 2018. p. 112–127.

[15] BADII, C. et al. Smart city iot platform respecting gdpr privacy and security aspects. IEEE Access, IEEE, v. 8, p. 23601–23623, 2020.

[16] SALHOFER, P.; JOANNEUM, F. Evaluating the fiware platform: A case-study on implementing smart application with fiware. In: Proceedings of the 51st Hawaii International Conference on System Sciences. [S.l.: s.n.], 2018. v. 9, p. 5797–5805.

[17] ALONSO, Á. et al. An identity framework for providing access to fiware oauth 2.0-based services according to the eidas european regulation. IEEE Access, IEEE, v. 7, p. 88435–88449, 2019.

[18] CABRINI, F. H. et al. Enabling the industrial internet of things to cloud continuum in a real city environment. Sensors, MDPI, v. 21, n. 22, p. 7707, 2021.

[19] CABRINI, F. H. et al. Helix multi-layered: a context broker federation for an efficient cloud-to-things continuum. Annals of Telecommunications, Springer, p. 1–13, 2022.

[20] JUNIOR, N. F. et al. Performance evaluation of publish-subscribe systems in iot using energy-efficient and context-aware secure messages. Journal of Cloud Computing, SpringerOpen, v. 11, n. 1, p. 1–17, 2022.

[21] JUNIOR, N. F. et al. Lightweight and secure publish-subscribe system for cloud-connected ultra low power iot devices. Journal of Communication and Information Systems, v. 36, n. 1, p. 100–113, 2021.

[22] AL-MASRI, E. et al. Investigating messaging protocols for the internet of things (iot). IEEE Access, IEEE, v. 8, p. 94880–94911, 2020.

[23] USLU, B. Ç.; OKAY, E.; DURSUN, E. Analysis of factors affecting iot-based smart hospital design. Journal of Cloud Computing, SpringerOpen, v. 9, n. 1, p. 1–23, 2020.

[24] JUNIOR, N. F. et al. Iot6sec: reliability model for internet of things security focused on anomalous measurements identification with energy analysis. Wireless Networks, Springer, v. 25, n. 4, p. 1533–1556, 2019.

[25] GÖÇER, B. D.; BAHTIYAR, ¸ S. An authorization framework with oauth for fintech servers. In: IEEE. 2019 4th International Conference on Computer Science and Engineering (UBMK). [S.l.], 2019. p. 536–541.

[26] ACADEMY, F. Keyrock Identity Management. 2021. Disponível em: <https://fiwareacademy.readthedocs.io/en/latest/security/keyrock/index.html>.

[27] FETT, D.; KÜSTERS, R.; SCHMITZ, G. A comprehensive formal security analysis of oauth 2.0. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. [S.l.: s.n.], 2016. p. 1204–1215.

[28] SILVA, A. et al. Energy-efficient node position identification through payoff matrix and variability analysis. Telecommunication

Systems, Springer, v. 65, p. 459–477, 2017.
[29] JUNIOR, N. F. et al. Privacy-preserving cloud-connected iot data using context-aware and end-to-end secure messages. Procedia Computer Science, Elsevier, v. 191, p. 25–32, 2021.
[30] SILVA, A. et al. Grouping detection and forecasting security controls using unrestricted cooperative bargains. Computer Communications, Elsevier, v. 146, p. 155–173, 2019.
[31] CABRINI, F. H. et al. Helix sandbox: An open platform to fast prototype smart environments applications. In: IEEE. 2019 IEEE 1st Sustainable Cities Latin America Conference (SCLA). [S.l.], 2019. p. 1–6.
[32] HARDT, D. RFC 6749: The oauth 2.0 authorization framework. Internet Engineering Task Force (IETF), v. 10, n. 10.17487, 2017.
[33] DENNISS, W.; BRADLEY, J. RFC 8252: Oauth 2.0 for native apps. Internet Engineering Task Force (IETF), 2017.
[34] SHERNAN, E. et al. More guidelines than rules: Csrf vulnerabilities from noncompliant oauth 2.0 implementations. In: SPRINGER. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. [S.l.], 2015. p. 239–260.
[35] ACADEMY, F. FIWARE Academy Docs. 2021. Disponível em: <https://fiware-academy.readthedocs.io/en/latest/>.
[36] ACADEMY, F. PEP-Proxy Wilma. 2021. Disponível em: <https://fiware-pep-proxy.readthedocs.io/en/latest/>.
[37] LODDERSTEDT, T.; MCGLOIN, M.; HUNT, P. OAuth 2.0 threat model and security considerations. [S.l.], 2013.
[38] ACADEMY, F. OASIS eXtensible Access Control Markup Language (XACML). 2021. Disponível em: <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html>.
[39] LODDERSTEDT, T. et al. Oauth 2.0 security best current practice. IETF Web Authorization Protocol, Tech. Rep. draft-ietf-oauth-security-topics-16, 2020.
[40] SELHAUSEN, K. zu; FETT, D. Oauth 2.0 authorization server issuer identifier in authorization response. IETF Web Authorization Protocol, Tech. Rep. draft-ietf-oauth-security-topics-16, 2021.
[41] LI, W.; MITCHELL, C. J.; CHEN, T. Your code is my code: Exploiting a common weakness in oauth 2.0 implementations. In: SPRINGER. Cambridge International Workshop on Security Protocols. [S.l.], 2018. p. 24–41.
[42] YANG, R. et al. Model-based security testing: An empirical study on oauth 2.0 implementations. In: Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. [S.l.: s.n.], 2016. p. 651–662.
[43] DEVESA, J. et al. An efficient security solution for dealing with shortened url analysis. In: WOSIS. [S.l.: s.n.], 2011. p. 70–79.

Authors' Profiles

Gilson da Silva Francisco holds a degree in Information Systems - São Judas Tadeu Universit (2007) and a Master's degree in Computer Engineering - IPT (2021).

He is a Software Engineer with 15 years of experience in analysis, development, and systems architecture.

Anderson Aparecido Alves da Silva holds a PhD in Computer Engineering (USP 2016) and has completed 2 post-docs in engineering. He worked for 23 years in the private sector and is currently a researcher at IPT and university professor. His line of research is about information security.

Marcelo Teixeira de Azevedo holds a Bachelor's Degree in Computer Science from Universidade Santa Cecília and a Post-Doctorate, Doctorate and Master's Degree in Electrical Engineering from the Polytechnic School of the University of São Paulo. He is currently an undergraduate professor at Mackenzie.

Eduardo Takeo Ueda is a Mathematician (UNESP) and Computer Engineer (UNIVESP). He holds a PhD in Electrical Engineering from USP), a Master's in Computer Science (USP) and a specialization in Health Informatics (UNIFESP). He is currently Professor in the Master of Applied Computer Science at the Institute of Technological Research of the State of São Paulo (IPT).

Adilson E. Guelfi holds a PhD in Electrical Engineering from Poli-USP. Adilson served as technical manager of LSI-TECH and currently holds the position of Dean of Research and Graduate Studies at UNOESTE (SP).

Jose Jesus Perez Alcazar holds a degree in Systems Engineering and Computation - Universidad de Los Andes (1983), a Master's degree in Computer Science from the UFMG (1988) and a PhD in Informatics from the PUC of Rio de Janeiro (1995). He is currently professor-doctor at the University of São Paulo.

How to cite this paper: Gilson da Silva Francisco, Anderson Aparecido Alves da Silva, Marcelo Teixeira de Azevedo, Eduardo Takeo Ueda, Adilson Eduardo Guelfi, Jose Jesus Perez Alcazar, "Vulnerability Detection in Intelligent Environments Authenticated by the OAuth 2.0 Protocol over HTTP/HTTPS", International Journal of Computer Network and Information Security(IJCNIS), Vol.16, No.2, pp.1-13, 2024. DOI:10.5815/ijcnis.2024.02.01