Novel Extended Federated Authentication and Authorization Framework

Edward Rajah  
School of IT & Computing  
American University of Nigeria  
Yola, Nigeria  
edward.rajah@aun.edu.ng

Mathias Fonkam  
School of IT & Computing  
American University of Nigeria  
Yola, Nigeria  
mfonkam@aun.edu.ng

Narasimha Rao Vajjhala  
School of IT & Computing  
American University of Nigeria  
Yola, Nigeria  
narasimha.vajjhala@aun.edu.ng

Abstract—Authentication and authorization are two significant areas in the context of security management. Federated authentication and authorization specifications were primarily based on the Open Standard for Authorization (OAuth) specification. The OAuth specification addresses critical issues, including scalability and extensibility. However, vulnerabilities have been identified in several OAuth-based systems. The novel extension to the OAuth 2.0 framework that we are recommending addresses the issues with the security of the data transferred between two endpoints. Our proposed novel model is session-less without any requirement for third-party authentication and authorization securing the logging process over HTTP communication. In our proposed model, the need for redirection is eliminated through the setting up of dedicated authorization servers. This proposed novel extension should help developers, security analysts, and managers address a crucial vulnerability associated with federated authentication and authorization.

Keywords—security, authentication, authorization, novel, vulnerabilities.

I. INTRODUCTION

Smart objects are now a reality with the availability of network connection in several devices [1]. Also referred to as the Internet of Things (IoT), more than 20 billion devices are expected to be connected over the network by the year 2020 [1]. The magnitude of the devices connected over the network is also likely to bring to the fore new security and privacy challenges. Authorization and authentication have been key issues since the advent of networks of interconnected computers [2]. Granting users’ authorization to access resources has proven to be difficult considering the complexity involved. Cybergincips are finding ways to access resources they should not be privileged to access.

The weakest point identified in Internet software security is user authentication and resource authorization [2]. Authentication is defined as the process of confirmation of an attribute and identity claimed by an entity using credentials [1]. Authentication has evolved since the advent of the Internet and the World Wide Web. User authentication has improved starting from the use of usernames and passwords to the use of biometric – fingerprint and face detection and recognition. Authorization is the process of granting specific resource access permissions to authenticated entities, including users and devices as well as applications [1]. However, with the increasing complexity of technology, new cyber threats have emerged as the number of devices connected to the Internet has increased exponentially. As a result, security experts are changing their approach to security by actively focusing on industry-grade production-ready technology. This novel approach is an improvement to the existing Open Standard OAuth framework as it helps security experts, software developers, and companies address exploits identified during security tests in the lab.

Security breaches can still occur, and users lose control of their resources when these breaches occur. Security experts are aware of this and often address security issues through software patches. However, software patches are generally behind schedule than exploits identified and released into the public domain. Software patches solve identified security flaws but are susceptible to new exploits themselves. Hence, software patches are not viable for achieving robust security. Passwords have been used for more than five decades as the mode for authentication but there is near consensus on the need for a more secure and user-friendly replacement [3]. Previous access control mechanisms and frameworks, included the Discretionary Access Control (DAC), Role-Based Access Control (RBAC), and the Attribute-Based Access Control (ABAC) [1].

Federated authentication and authorization, also popularly known as “single sign-on” protocols can help reduce the problems with passwords [3, 4]. The use of federated authentication allows the central identity provider to authenticate the user through passwords are still needed to log into the central identity provider [3]. In the federated authentication system, the user is identified to a relying partner (RP) using an identity that is managed by an identity provider without exposing the user’s identity credentials. An example is logging on to a third-party site by using the Facebook credentials. In this case, Facebook is the identity provider. Some of the successful adaptions of the model include Facebook Connect, which was initially based on the OAuth specification [3, 5]. OAuth is a specification and framework that adapted the federated authentication model. Though this specification addresses the issues of scalability and extensibility, this framework is susceptible to several attacks [6]. One of the key problems of OAuth-based SSO systems is that most of these are often executed on abstract models because of which some of the implementation details are unintentionally ignored [7, 8]. The OAuth 2.0 framework is vulnerable to application impersonation.
attacks because of the use of multiple authorization flows as well as token types [8, 9].

Previous research has shown that the usage of OAuth is not clearly defined and there is a certain degree of ambiguity in the specification [6]. As a result, several RPs implement customized OAuth without adequate verification. The OAuth framework implementation has been vulnerable to phishing attacks, replay attacks as well as impersonation attacks [4, 6, 8-10]. In response to the growing need of improved user authentication over the Internet, specifications, and frameworks, such as HTTP Digest, OAuth 1.0, OAuth 2.0, Hawk, OpenID, and Bearer Token have been created [5]. However, each of these frameworks has inherent security flaws arising from their protocol specification. Most of the work done over the security of the OAuth specification deals with the various implementations of the specification rather than the standard itself [11]. Our paper focuses on some of the key aspects and flows in the OAuth 2.0 specification. We propose a novel extended authorization framework based on the OAuth 2.0 framework that addresses some of these vulnerabilities of the OAuth 2.0 framework.

II. BACKGROUND

A. OAuth 2.0 Specification

The OAuth protocol initially came up in 2007 after which it was adopted as a standard by IETF in 2010. OAuth 2.0 got several refinements over the initial OAuth 1.0 version which was mainly developed by the dedicated community [2, 12]. The updated OAuth 2.0 standard was formally accepted as a standard by IETF in 2012 [4]. The OAuth specification has a provision for an access token to be issued in an automated way [12]. OAuth 2.0 is one of the most popular federated authorization systems and is also the basis for the new single sign-on (SSO) standard OpenConnect [9-11]. OAuth 2.0 is mainly used for authorization though it is also now used by several enterprises for authentication [5, 10, 11]. One of the key features of standards and frameworks, such as the OAuth 2.0 framework is that we could use the social-login flows [13]. This would allow the user to login and access services using the existing accounts, for instance Google or Facebook, instead of creating a new account [13]. The user also has the option to authorize the application to access the data without sharing the credentials along with the option of revoking the access.

![Fig. 1. Abstract Protocol Flow for Two Parties](image)

OAuth 2.0 framework allows some aspects of the framework to the developer, including the choice of the access token, the storage of the access token and the decision on the implementation of the scope parameter [14]. The OAuth 2.0 protocol uses tokens consisting of the encoded access rights of the token owner along with a configurable lifespan. Two methods are used for communicating through a RESTful API, including the use of JSON web tokens and a special token format allowing a specific number of accesses [1]. Several services have opted for the use of multifactor authentication to provide enhanced security and prevent security breaches as a result of the loss of a single credential [1].

RESTful web services are lightweight and are quite useful for creating APIs for clients spread across the Internet though the communication is stateless [15]. The use of tokens to authenticate clients is one of the simpler, common, and reliable methods of authentication [15]. The client initiates this process by sending credentials to an authentication system which in the event of successful validation sends a token to the client. The client can use this token to identify itself and access the service. JSON Object Notation (JSON) tokens are compact tokens containing information that can be verified and trusted as the tokens are digitally signed by the issuer [15]. A JSON web token has three parts, namely, the header, the payload, and the signature [15]. The header has information about the type of the token as well as the hashing algorithm used by the token. The payload includes both the public and private keys used while the signature is used to verify the veracity and trustworthiness of the token [15].

OAuth2 involves four major roles, including the Resource Owner (RO), the Resource Server (RS), the Authorization Server (AS), and the Client [13]. The RO is an entity, including but not limited to an end user who can grant access to a protected resource. The RS can host the protected resources and is capable of accepting and responding to protected resource requests using access tokens. The AS can issue access tokens to the client after authenticating the RO and obtaining the authorization. OAuth 2.0 framework allows the developer to implement the AS and the RS on different machines [14] but these roles can be combined as well, for example, Google offers AS along with exposing the profile information. The token issuing mechanism allows the ROs to share their data hosted at the service providers with third-party applications [12]. The sharing of the token is not allowed without the explicit agreement of the RO [12].

The OAuth 2.0 specification allows the developer to choose from a range of access tokens [16]. Most of the developers usually choose between the Bearer Token and the JSON Web Token (JWT) [14]. The decision on the choice of the access token is usually influenced by three factors, including performance, ability to instantly revoke a token when the access is compromised, and the security of the access token [14]. The OAuth 2.0 specification allows two flows for RPs to obtain their tokens, namely, server-flow and client-flow [7]. Server-flow is ideal for web applications that receive their tokens from the server-side programming logic while client-flow is suitable for applications running on the client-side. For instance, in some of the client-side applications, the secret-key cannot be embedded and in such cases, the client-flow would be required to obtain the tokens.
As per the OAuth 2.0 framework, the authorization code allows the client to access a resource. However, if the authorization code is captured during the communication between the client and the resource owner, then this code can be used to launch a replay attack to gain access to the resource server (RS) [4]. The OAuth 2.0 framework is also vulnerable to a phishing attack in which the client generates a malicious client to retrieve the resource owner’s identity and password. After gaining access to the resource owner’s access information, the attacker can also gain access to the resource server [4].

The OAuth 2.0 framework was also vulnerable to the impersonation attack in which the attacker would first intercept the authorization code [4, 8]. The attacker would then use this code to impersonate a new session and gain access to the resource server [4]. The abstract protocol flow in Figure 1 describes the interaction between a client and a resource server. The resource server also acts as the authorization server granting access to protected resources on hosted on it. This configuration reduces the number of exchanges and parties involved in the authentication and authorization process, improving communication privacy.

The protocol flow for Fig. 1 is as follows:

- The client initiates a connection with the resource server requesting for access to data and/or information stored on the resource server.
- The resource server verifies the client and grants access to the protected resource via a generated access token. The access token comprises of a set of attributes that uniquely identify the client (like a fingerprint).
- The client captures the access token, includes it in the request URL and sends it to the resource server.
- The resource server validates the access token with the authenticated client username and system attributes/specifications.
- If valid, the resource server grants the client access to the protected resources on the server. If invalid, the client get redirected to http:127.0.0.1 by default or shown the authentication page.

The abstract protocol flow in Fig. 3 describes the interaction between a client, a resource owner, an authorization and a resource server. Unlike in Fig. 2, one more party is introduced, which is the resource owner. In this configuration, the resource owner responds to an authorization request from the client and grants the client access to communicate with the authorization server, which issues the access token for accessing protected resources on the resource server. Both the resource owner and authorization server are separate entities unlike in Fig. 1 where all roles are performed by the resource server. The configuration in Fig. 3 illustrates the implementation that is currently used in production/ enterprise environments and by companies such as Google, Facebook and Twitter.

The protocol flow for Fig. 3 is as follows:

- The client makes an authorization request, that is, initiates a connection with the resource server requesting for access to data and/or information stored on the resource server.
- The resource server either presents an authentication page of the third-party authorization server (recommended) or redirects the client to the authorization server itself for client credentials’ extraction and identification.
- The authorization server issues an access token to the client, which is captured by the client and used in subsequent requests made to the resource server.
- The resource server validates the client’s access token used in the request. The validation of the access token is achieved by comparing the verified username with that embedded inside the access token with the set of attributes that uniquely identify the client (like a fingerprint).
- If the access token is valid, the resource server grants the client access to protected resources on the server. If invalid, the client get redirected to http:127.0.0.1 by default or required to repeat the second step.
where approval is required to access the authorization server.

- Upon approval, the client is redirected to the authorization server with an authorization request to access resources on a resource server.
- The authorization server issues an access token to the client, which is captured by the client and used in subsequent requests made to the resource server.
- The authorization extracts the client’s credentials and issues an access token, which is captured by the client and used in subsequent requests to the resource server.
- The resource server validates the client’s access token in the request. The validation of the access token is achieved by comparing the verified username with that embedded inside the access token with the set of attributes that uniquely identify the client (like a fingerprint).

- If access token is valid, the resource server grants the client access to protected resources on the server. If invalid, the client gets redirected to http:127.0.0.1 by default or required to repeat the second step.

III. PROPOSED MODEL

The extended OAuth authentication framework that we are proposing is a sessionless protocol without any third-party authentication and authorization requirement. The login procedure is secured over the HTTP communication. The key difference between our extended framework and the OAuth 1.0 and OAuth 2.0 specifications is that the need for redirects and setup-dedicated authorization servers is removed.

The protocol flow for Fig. 4 is listed below:

- The client makes an authorization request, that is, initiates a connection with the resource server requesting for access to data and/or information stored on the resource server.
- The resource server either presents an authentication page of the third-party authorization server (recommended) or redirects the client to the authorization server itself for client credential’s extraction and identification.
- The authorization server issues a time constrained access token to the client, which is captured by the client and used in subsequent requests made to the resource server.
- The resource server validates the client’s access token used in the request. The validation of the access token is achieved by comparing the verified username with that embedded inside the access token, followed by the TTL value.
- If the access token is valid, the resource server grants the client access to protected resources on the server. If invalid, the client gets redirected to http:127.0.0.1 by default or required to repeat the second step.

Fig. 4. Refreshing an Expired Access Token

The protocol flow in Fig. 4 describes a configuration with an authorized session expiring. The illustration is a simple setup involving a client, an authorization and a resource server. In this configuration, the client’s authorization request is responded with an access token constrained by a time-to-live (TTL) value measured in seconds. The access token can be used for accessing protected resources on the resource server before the session’s TTL elapses.
The authorization flow in Fig. 5 illustrates the components involved in the identification, verification, and validation of the client requesting access to a protected resource on a resource server. The protocol flow of our novel extended specification is listed in Fig. 6. The end-user in the illustration refers to the entity using a program capable of making HTTP requests over a network. The client refers to the program used to perform the request and the user-agent refers to the program’s identifier such as engine type, version, architecture and so on. A malicious client cannot impersonate another client and obtain access to protected resources. This is because the server doesn’t keep its client’s credentials. In our proposed model, if an attacker manipulates the value of the token (authorization key), it will not cause the authorization server to grant access to protected resources - device fingerprint. Instead, the attacker will be redirected to the login page or to 127.0.0.1. Unlike Digest Authentication, replay attacks are impossible. The authenticated session is fingerprinted to the authenticating device using a series of parameters to create an immutable collection. The simulation of the model was done in the context of a “Writing Center” application. The testing and simulation of the model is shown in Fig. 7 and Fig. 8.

![Token Structure for the Proposed Extension](image1)

Once the server receives the request, the server validates the user credentials and returns a token. The token is then used for all subsequent requests to authenticate the user. However, unlike other tokens in other OAuth implementations, the token cannot be reused even if copied from one computer to another. Unlike OAuth2, in our proposed model neither the client identity nor the token are transmitted in the header and dependent on SSL certificates for protection.

![URL Structure in the Proposed Model](image2)

Access token credentials can be transmitted with/without TLS v1.2 because the token is encrypted. The transmission of authorization codes may be contained in the authorization key. In our proposed model, the end-user need not be redirected to a third party site to complete authentication. Our proposed model reduces phishing attacks. A CSRF attack cannot result because there isn't an authorization server. The entire authentication is sessionless and mandatory to exhibit such a property.

Given that the proposed framework works over the HTTP protocol, the initial transmission of user credentials is unencrypted. To address this issue, the username and password are tied to the device. In the case of hybrid-desktop applications such as Electron, the device’s UUID can be extracted paired with the registration credentials of the user. In the case of the browser, a collection of difficult to change values is used such as the computer’s GPU fingerprint, which is paired with the registration data sent to the server.

Assuming an attacker sniffs the packet and extracts the username and password, it still cannot be used. This is because the username and password are not paired with the attacker’s computer system. However, users with multiple devices can still use the same username and password on their devices by directly enrolling their devices via a link they will generate and visit on their other devices. Nevertheless, the use of an SSL certificate is still an added advantage.

Perhaps another precaution that implemented is the encryption of the username and password before transmission on the client-side before transmission. This technique also does not rely on the presence of an SSL certificate to secure the communication. However, the server has to know the encryption key that was used on the client-side to decrypt the data. The idea is to only secure the initial data transmission of user credentials with the server. Thus, an SSL certificate is still needed if subsequent communication requires confidentiality.

### IV. SUMMARY

In this paper, we proposed a novel extended OAuth authentication framework which would address some of the vulnerabilities of the OAuth federated authentication and authorization framework. The major vulnerability of the OAuth specification is the lack of a proper mechanism for protecting the token from eavesdroppers when the tokens are transmitted from the service provider to the client. Although there is a mechanism for verifying the request integrity, the confidentiality of the request itself is not guaranteed. The novel extension to the OAuth 2.0 framework that we are proposing addresses the issues with the security of the data transferred between two endpoints. Our proposed novel model is session-less without any requirement for third-party authentication and authorization securing the logging process over HTTP communication. In our proposed model, the need for redirection is eliminated through the setting up of dedicated authorization servers. Future research work could explore further on standardizing this process by testing it in different environments and conditions. We had tested this novel extension framework in a Writing Center application that was developed, and the model was validated. However, further testing and validation of this model are required to explore any new vulnerability.
V. REFERENCES


