

Securing Web Applications with OpenID Connect, OAuth 2.0 and Two-Factor Authentication

Richard Madden¹, William Farrelly¹, Kevin Curran²

¹Computer Science Department

Letterkenny Institute of Technology, Letterkenny, Ireland

²School of Computing, Engineering and Intelligent Systems

Ulster University, Derry, N. Ireland

Abstract

Usability and security are vitally important as the Internet is now classified as the new normal, and sometimes the only, medium for contact between people, private businesses, governments, and other organizations, identity management becomes a vital component of online communications. To access any online resources today requires end-users to register their identity with a Service Provider (SP) that provides and operates the service or services. The registration process involves end-users supplying a variety of personal data about themselves. This data is then stored with the Service Provider. Identity and Access Management (IAM) is the digital format for the management and control of user information, but what does that mean? Identity & Access, Management covers a large range of technologies because it is a framework of policies and technologies for ensuring that the proper people in an enterprise have the appropriate access to technology resources [1]. Single Sign-On, Public Key Infrastructure, Active Directory, Kerberos, Governance, Enterprise Directory, Privileged Access Management are some of the main components in the IAM space. There are many technologies available today that offer single sign-on (Active Directory, Enterprise Directory, Kerberos, SAML, OpenID Connect, WS-FED, WS-TRUST), and although each one offers that single sign-on experience in terms of security however OpenID Connect is now seen as the go-forward solution because it is a decentralized standard, meaning it is not controlled by any individual or set of individuals, website or service provider [2]. We explore the standards of the OpenID Connect (OIDC) protocol that was introduced by the OpenID Foundation. OIDC is built on a version introduced earlier called OAuth 2.0. The reason behind this came from the fact that the OAuth protocol was being used for user authentication and authorisation, however, OAuth was only ever an authorisation framework. Now with OpenID Connect, it is an almost comprehensive protocol for the implementation of user identification and authorisation. The easiest way to understand the difference between each protocol is, OIDC issues access tokens (for authentication) and OAuth 2.0

issues id tokens (for authorisation). The first part of this paper introduces the OpenID Connect protocol (which takes care of authentication and authorisation), while later parts discuss Multi-Factor Authentication that will showcase problems that both the OpenID Connect (OIDC) protocol and Multi-Factor authentication can solve. Finally, a study of current implementations, how to adopt the OpenID Connect (OIDC) standard into a web application, while also enabling two-factor authentication will be explored.

Keywords: OpenID, Authentication, OAuth, security,

1. Introduction

Any account used online in today's world is under attack from sophisticated actors looking to steal their identity. Therefore, it is essential to secure anything that requires an end-user to enter a credential to gain access, but it is also important to find the balance between a nice user-friendly login process and security. Today, most organizations have migrated most services to the cloud, and because of this, the requirement from an end-user is to create multiple accounts to access these resources. What then ends up happening is the same password will be used across these platforms, meaning when an account is compromised the sophisticated actor may get access to a wide range of resources. Obviously, in an enterprise it's important to provide training to end-users around password management, how to identify a genuine service, etc, however, an application needs to have a safe and secure authentication method and authorisation method to protect end-user data. Implementing the OpenID Connect protocol, OAuth 2.0 Authorization Code Flow to manage authentication and authorisation, and then followed up with a Two-Factor authentication challenge will do exactly that. Identity and Access Management (IAM) refers to the ability to manage user identities and their access to IT resources such as systems, applications, files, and networks. Identity and Access Management solutions have been a critical aspect of IT infrastructure for

many years now as they help to make work happen. However, many IT admins have come to discover that traditional Identity and Access Management solutions are struggling to manage the complexity of modern networks. As a result, IT organizations are now looking for new approaches to Identity and Access Management. Historically, the most popular IAM platform has been Microsoft Active Directory (AD). Active Directory is an on-prem IAM platform that was designed for on-prem, Windows-based environments. When AD was introduced in 1999, most IT networks were on-prem and Windows-based. Windows has remained the most popular enterprise operating system ever since. So it's no surprise that Active Directory is now ubiquitous. However, IT networks started to change as MAC systems, Linux servers, web applications, alternative storage solutions, Google Apps, AWS, and the Cloud came to market in the mid-2000s. Solutions such as these were not Windows-based, nor were they on-prem. Consequently, Active Directory implementations began to struggle, and to this day still do. Of course, IT organizations could patch Active Directory with third-party add-ons such as identity bridges, web applications single sign-on, privileged Identity Management, and more to mitigate some of these traditional challenges. However, the trouble with this approach is that it adds significant cost and complexity, not to mention that modern IT organizations would rather shift their identity management infrastructure to the cloud.

The good news is that there are next-generation cloud IAM platforms available today, one of those being Okta which is introduced as part of this project. This is effectively Active Directory reimaged for modern networks. Think of this as a virtual IT resource, without the help of costly third-party add-ons and anything on-prem. This type of platform allows user management, secure connections to their systems, applications, files, and networks regardless of the platform, provider, protocol, or location. The discussion around identity ownership and validation/verification is shared, meaning it is not up to just one entity (be that public or private) to have exclusive possession or control. For example, if you look at the Republic of Ireland the Government can use Personal Public Service Number (PPSN) to identify individuals living on the island in Ireland, and of course, other countries around the globe will have something similar. In the United States, they can use Social Security Numbers (SSN) to identify their citizens. If you look at a private firm they may use one framework for identifying their clients and then a separate mechanism to identify employees. Also in many cases, a single person can have many identities. For example, physical identity can be their driving license, passport, employee card, or even a student card.

Similarly, a single person can have multiple online identities. Someone may have an electronic bank account, an email address, or an address with a social network. Every such identification refers to the same person but is retained in separate institutions. Although the search for a common, unified identity continues, OpenID Connect seeks to establish an integrated identity authentication system across multiple public and private institutions. OpenID Connect (OIDC) offers a secure, easy, and efficient method for individuals to represent themselves to a wide range of applications and service providers using one or more of their identities held by a trustworthy Identity Provider (IDP). OpenID Connect has received a good endorsement and widespread acceptance from the world's leading high tech firms, such as Google, Amazon, and Facebook, since its launch in 2014 (see Figure 1).

OpenID Connect takes the pressure off-web application developers to expend additional manpower and resources thinking about how to manage user data stores that house credentials because if OIDC is being considered as the IAM stack for the web application the likelihood is the OIDC provider that the web application is going to integrate with for its authentication will be external to the web application meaning the ownership is on the external party to manage those user credentials. With this type of authentication mechanism, what then happens is that authentication is assigned to those trustworthy organizations who have specifically developed committed, purpose-built IDP. In turn, for users who can provide or who choose to use their Google, Twitter, or Facebook identity to access the resource in question, authentication becomes very convenient [3]. This paper will aim to demonstrate how OpenID Connect will soon be the cornerstone of Identity and Access Management. This research explores the OpenID Connect (OIDC) protocol, OAuth 2.0, and adding multi-factor authentication to any web application. Researching these three components will show how it can help not only large scale organizations but small to medium enterprises achieve that ideal Identity & Access Management world. We explain in detail the OpenID Connect standard as drawn up by OpenID Foundation. This will also include its key features and any potential security and/or even privacy issues. Potential improvements to the OIDC standard are briefly touched on and comparisons between Security Assertion Markup Language (SAML) 2.0 and OpenID Connect are explored. To date, there has not been a body of academic work that explores all three protocols in an effort to provide authentication and authorisation of end users. There are papers available that look at using the OAuth 2.0 Authorization Framework for authentication (which is incorrect and against spec)

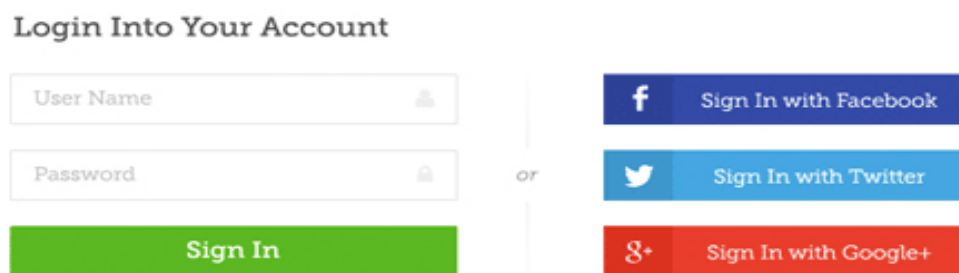


Figure 1: Use Twitter, Google, or Facebook Identity

and then talk about implementing MFA if they had more time. Any diagram that is not referenced to an external source is my own work. Threat actors are now rampant across the internet looking to steal user identities with specifically crafted social engineering attacks and credential harvesters, so now more than ever it's important that any business, organization, or enterprise implement the correct single sign-on solution but specifically also enabling a solution with two-factor authentication. Of course, two-factor authentication also has its weaknesses, for example, some applications that enforce 2FA allow the user to input a code sent via SMS to their mobile device, and although this is still verifying your identity it is open to abuse. Rather than being created by the system itself, the code is transmitted via SMS. This provides the opportunity for an interception of the code. There is also a possibility of SIM swapping, whereby a sophisticated actor obtains a SIM card with the phone number of the victim. The SMS messages sent to the victim will then be intercepted by the sophisticated actor giving them the authentication code. For this reason, it is important when implementing 2FA SMS should not be allowed nor should the code be sent to the user's email address. User passwords will still get compromised, however, that second identity challenge safeguards not only the user's data but also the data belonging to the organization/enterprise in question. This research will focus on the correct way to implement OIDC, OAuth 2.0, and two-factor authentication to help with protecting data from threat actors.

The purpose of this research is to examine the OpenID Connect protocol (its only purpose is to handle user authentications) and as part of OIDC, the OAuth 2.0 Authorization Code Flow is enforced (i.e. the OpenID Connect component in charge of authorisation), and finally, the research explores how to integrate an OIDC Provider with a Cloud Identity Provider to enable secure user authentication and authorisation.

2. OpenID

Users, roles, and access are some of the terms that has been heard concerning Identity and Access

Management. Identity is how one is represented online, sometimes through a social log-in, work email address, your personal email address, or even an application. So, an identity trying to gain access to a protected resource securely is what Identity and Access Management strive to provide. Some of the challenges that Identity and Access management need to address are things like password management, alleviating identity silos, and securing APIs. Also being able to have cross-domain federation so users can use a single identity across multiple systems. But to secure endpoints it's important the correct technologies are used and then enforce strong authentication and authorisation policies as seen in Figure 2. There is also account management and provisioning that IAM systems provide, so things like creating profiles, setting up security questions, access control based on role or attributes mapped in the directory. From the outset Identity and Access Management looks like it's simply about access but it is not if you look at the bigger picture. It is about ensuring that end-users or employees have a great user experience and this is balanced out with security.

2.1. OpenID Connect (OIDC)

The OpenID Connect protocol is explained in detail in this section. While OpenID Connect is built based on OAuth 2.0, it has addressed some of the areas that were missing in OAuth 2.0. OpenID Connect is a basic layer of identity that strengthens the protocol of OAuth 2.0 to enable an end user's identity to be checked by applications [4]. This is possible by simply authenticating the end-user against the Authorization Server/Host, and in addition to that simple but yet important information about the user can be provided through this authentication mechanism.

OpenID Connect can be used by web browsers, mobile applications, and browser-based clients to obtain information about authenticated sessions and end-users. OpenID Connect is scalable and extensible [5]. It encourages participants to use optional features such as OpenID Provider discovery, end-user identity data protection, and, where applicable, session management.

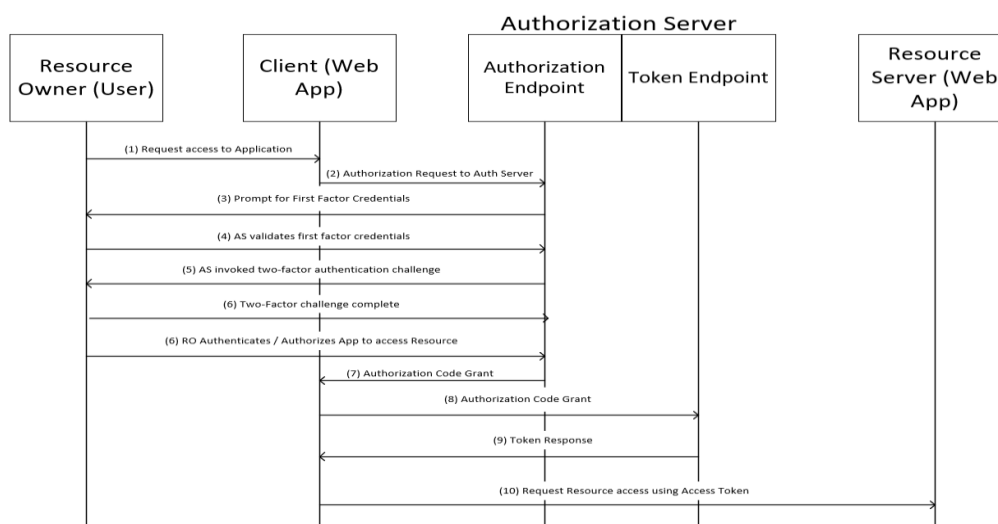


Figure 2: Enforcing Strong Authentication and Authorization Flows

OpenID 2.0 is the precursor for OpenID Connect, and like OpenID 2.0, OpenID Connect executes many of the same functions with Application Programming Interfaces (APIs) that allow integrations with many applications both mobile applications and native applications. Furthermore, OpenID Connect also allows for expanded methods for effective cryptography and signing. However, OpenID Connect is fully implemented into the OAuth 2.0 protocol itself, unlike OpenID 2.0 and its incorporation with OAuth 1.0, which involved an expansion. With OpenID Connect, three separate entities are needed to perform end-user authentication. These are 1. The end-user or the resource owner; 2. A client looking for access to the end-users resource and 3. OpenID provider performs user authentication to the client (Resource Owner / Authorization Server) [6]. Figure 3 shows the relationship between the entities. In some scenarios the Authorization Server and Resource Server reside on separate entities, this relationship is portrayed in Figure 4.

2.2. OpenID Foundation

The standards around OpenID Connect came from the OpenID Foundation. This foundation is the founder, publisher, keeper, and even guardian of the OpenID principles. It also maintains the OpenID culture and technology and nurtures them. The OpenID Foundation is a non-profit international standardization organization of individuals and companies committed to enabling, promoting, and protecting OpenID technologies [7]. The OpenID Foundation requires OpenID Connect adopters and implementers to approve their implementations to ensure compliance and to globalize with implementations. The qualification process of the foundation

uses the novel self-certification and compatibility method to the test suites established by the foundation. Once certification is achieved, implementations are entitled to publicly show the certification label "OpenID Certified" [7]. This is necessary to achieve user 'buy-in' and to promote confidence in the OIDC protocol and its implementations.

2.3. OAuth 2.0 Authorization Framework

OpenID Connect is based on the OAuth 2.0 Authorization Framework. This framework then allows applications to gain restricted/limited access to an HTTP service. This can be done by initiating a permission contact process between the resource owner and the HTTP service on behalf of the resource owner, or by encouraging the third-party application to gain consent on its behalf [8]. OAuth 2.0 was developed to help implement proper security control between client to server authentication. Before OAuth 2.0 if a request was made from a client to a protected resource (resides on a server) it was done so using credentials provide by the resource owner. This was the result of highlighting a large security flaw (some mentioned) in the conventional client to server authentication frameworks because to allow third-party applications to access the resource meant the owner of the resource had to share its creds with those third-party applications [9]:

- Third-party apps are expected to save the login credentials of the resource owner for future use. Usually, that's either a password in plain text or a hash. Also, considering the inherent safety vulnerabilities in passwords, servers must support password authentication.

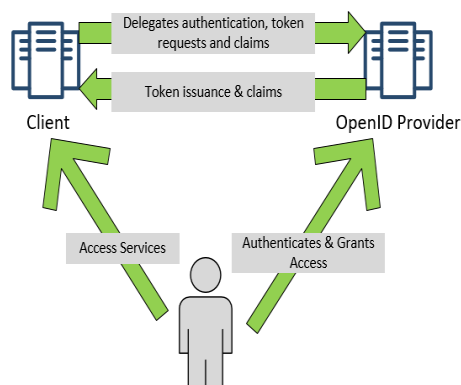


Figure 3: User, Client, and OpenID Provider relationship

- Third-party apps are equipped with grossly granular access to the properties of the property holders. This limits the potential of resource owners to restrict the length or accessibility to a small subset of resources.
- Without revoking access to all third parties, the resource owner cannot revoke access to a single third party. And only by modifying the password of the third party will this be done.

OAuth then came into play and broke up these elements by adding an authorisation layer and distinguishing the client's function from that of the resource owner. This meant the client must request access to the protected resource. With this new layer, the client will be issued an access token. With this token, it has a scope, expiration time, and other attributes. These tokens are provided to these third-party clients/applications with approval from the resource owner by an authorisation server. To access the secure services managed by a resource provider, the client uses that access token [9]. To provide some clarity, a resource owner (end-user) allows a video recording service (client) access to their protected videos which are hosted on a video share (resource endpoint) without ever providing any username and password to the recording service. So, with OAuth, the user invokes an authentication request with any authorisation server that is trusted by the video recording service (client). This service can then issue the video recording service (client) an access token.

2.4. OpenID Connect Principal Function

OpenID Connect provides that Identity Layer to OAuth. OpenID Connect is a simple identity layer that sits on top of the OAuth 2.0 protocol. It enables clients to verify the identity of the End-User based on the authentication performed by an Authorization Server [4].

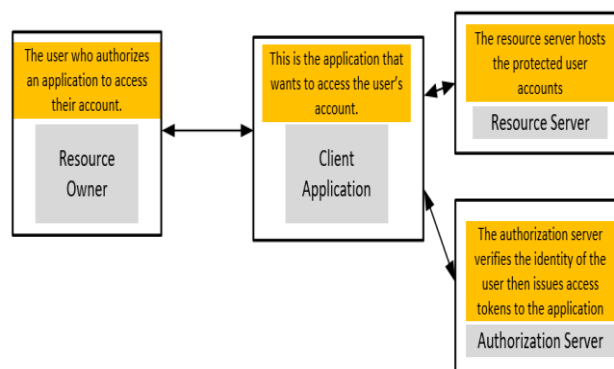


Figure 4: Parties who separate both functions

OpenID Connect also obtains basic profile information about the end-user using REST APIs. OpenID Connect is needed because even though OAuth provides authorisation, it does not provide authentication. With OAuth, the user authenticated and proved they were present to the Authorization Server, but the sole purpose of this was to create and grant an access token to the client application.

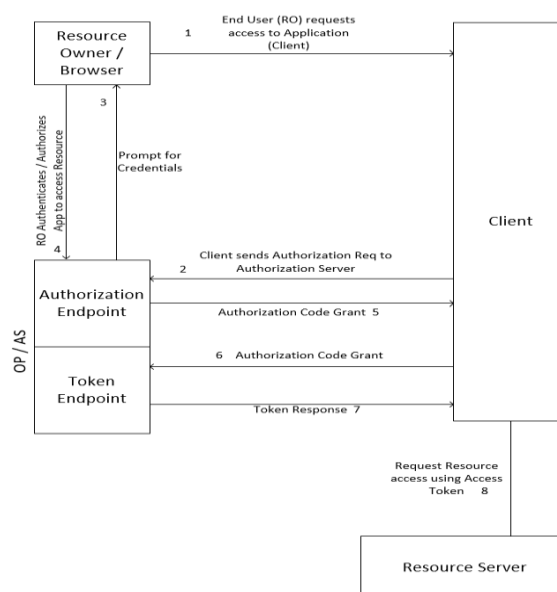


Figure 5: OpenID Connect Authentication Flow

For clients to leverage this extension in the Authorization request they simply include the OpenID scope value [4]. What is then returned is something called an ID Token. This ID Token is ID token is encoded as a JSON Web Token or JWT. Often referred to as OpenID Providers (OPs) are OAuth 2.0 Authentication Servers that implement OpenID Connect. Relying Parties (RPs) are quite often referred to as OAuth 2.0 Clients using OpenID

Connect. Figure 5 shows the OpenID Connect protocol in a simple form.

2.5. Authentication – What it means in the OIDC world.

Authentication in OpenID Connect means challenging the end-user to confirm their identity (currently not logged in to the application) or it can mean that a determination is made that the end-user is already logged and no authentication is necessary (the user has a valid session). Upon successful authentication, an Identity Token (JSON Web Token) is sent from the server to the client. The identity token is similar to an ID card or passport. It contains several required attributes or claims about that end-user but also how the end-user was authenticated. These claims are Subject, Issuing Authority, Audience, Issue Date, and Expiration Date.

- The Subject is a unique identifier assigned to a user by the Identity provider, for example, a username.
- The Issuing Authority is the Identity provider that issued the token.
- The Audience identifies the Relying Party who can use this token.
- The Issue and Expiration Date is the date and time the token was issued and will expire.
- Several optional claims help the relying party validate the ID token, such as Authentication time which shows the time the End-User was authenticated, and Nonce values which mitigate replay attacks.
- The token may also contain additional requested claims about the subject such as name and email address.

With OpenID Connect there are three avenues authentication can follow which determine how the tokens will be sent back to the client [4] - Authorization Code Flow, Implicit Flow and Hybrid Flow. The Identity token is encoded as a JSON Web Token or JWT. In this token are what is called claims and they form part of its payload. Like an access token, the Identity token is also digitally signed using JSON Web Signature to achieve integrity and non-repudiation. The Header, Payload, and Signature are combined into a JWT and are encrypted with JSON Web Encryption for confidentiality [4]. OpenID Connect uses scopes to retrieve information in the ID token.

2.6. Scopes

The OpenID Connect specification contains four standard predefined scopes (Profile, Email, Address,

Phone) which are used to supply the client application with consented user details. The OpenID scope is a mandatory scope to specify that OpenID Connect is required. For example, the scope “profile”, requests access to the End-Users default profile claims. In the initial authentication request, the client application can request scopes or claims to be returned in the Identity Token. Alternatively, they can be requested using an access token through a REST API call to the UserInfo endpoint.

The OpenID Connect Identity Provider has many End Points with which the End-User and Client Application interact. These are the Authorization Endpoint, the Token Endpoint, and the UserInfo endpoint [10]. The Authorization endpoint is where the End-User is asked to authenticate and grant the client application consent to access their identity, and any other required information such as email, or address. This extra information is called UserInfo claims. Once consent is given, this endpoint passes back an Authorization code. This is the endpoint in which the End-User indirectly interacts with the Identity Provider through a user agent, for example, a browser.

The token endpoint authenticates the client application. It also exchanges the authorisation code from the Authorization endpoint, for an ID token, an access token, and an optional refresh token. The UserInfo Endpoint is an OAuth 2.0 protected resource that is used by the Identity Provider to return consented user information or claims to the client application, provided that a valid access token is presented.

Claims are values containing end-user information, but also information about the OpenID Connect service itself. Example claims can be last name, telephoneNumber or first name. The party responsible for issuing these claims are the OpenID Connect providers. These are bundled up into security tokens called ID Tokens. These tokens are issued based on trust relationships that are built using federated technologies.

2.7. JSON Web Tokens

JSON Web Tokens (JWT) are an open, industry-standard RFC 7519 method for representing claims securely between two parties [4]. So think of them like JSON payloads that one party will send to another and the receiving party will be able to make sure that the payload that they received was effectively sent by another party whom they are federated. JWT is just for authorisation not authentication. Authentication is about taking in a username and a password and authenticating to make sure that the username and password is correct. Authorisation is making sure that the user that sends request to the endpoint/server/application is the same user that actually logged in during the authentication

process. At the beginning, there is the user authentication challenge with email and password. This is sent in a POST request to the server. The server then creates a JSON web token. This JSON web token is encoded and signed using a secret key (This secret key is known by the application and the authorisation server. This is generated/provided during the integration process). So the authorisation server will know if it has been tampered with. The JSON web token is then sent back to the client. This JSON web token has all the information about the user in its

payload (claims). The client is then going to POST a request to the authorisation server with that JSON web token with the authenticated end users claims

(payload). The authorisation server verifies that this JSON web token has not been changed/tampered with since it was signed [11]. Using the jet.io website here is sample JSON web token as seen in Figure 6 .

On the left there is an encoded version of a JSON web token and on the right is the decoded version of that JSON web token showing the three different parts, header, payload and signature. The header determines the algorithm to encode and decode. The payload which has all the information (claims) and lastly the signature which allows verification of that token.

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyMTM4NjIzMjMlLCJuYV11IjoIiGVGVzdGluZyBvc2VyIiwiaWF0IjoxNTE2MjM5MDIyfQ.EDBBCBw3ZuMJZIH1R1Tr7axeFDwwvIFreKajWFrDvvs
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "213862323",
  "name": "Testing User",
  "iat": 1516239822
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  THASDINASF____-23912-38
) ☒ secret base64 encoded
```

Figure 6: The jwt.io website

This verify signature as seen in Figure 6 is the most important part because it is going to verify that the JWT was not altered. It does that by taking the first two portions of the token separated them with a period and base64 encodes them, it then takes the algorithm that is defined in the header and it uses that algorithm and the secret to encrypt the token. So, if this token was to be tampered with past this point the signature will no longer match. When the client or when the server gets the JSON web token the header and payload will be decoded, then combined together and hashed again with the algorithm defined in the header. The last portion of the key is then validated against what has just been decoded versus what was just received and if it does match then it has not been tampered with.

3. OpenID Connect (OIDC) & MFA Design

We document the implementation and integration of a web application to an OpenID Provider (OP) using the Authorization Grant Flow. The web application itself will be hosted on an external-facing Linux host but built using NodeJS and Jhipster.

JHipster is an open-source application generator used for web application development and microservices using Angular or React and the Spring Framework. As part of this application build, you can specify OpenID Connect / OAuth 2.0 as the authentication and authorisation protocol. This web application will be hosted on an external-facing public cloud server over HTTPS as seen in Figure 7 . The web application will then be integrated with Okta using the OpenID Connect protocol. Okta will act as the OpenID Provider and multi-factor authentication will also be enabled in the Okta tenant enabling that two-factor authentication challenge for the end-user base. The application will have two types of user roles, Administrators and Users. Post a successful authentication to this web app the administrators will have full access allowing them to see servers resources, logs, etc., however, a standard user will not have this level of view. As part of the configuration in the Okta tenant groups can be created, users can then be added to these groups, and post a successful authentication the id_token sent back to the application will include the user's group. The web application will then decode the token and based on the group found in the token it then decides what the user can see and do in the web application.

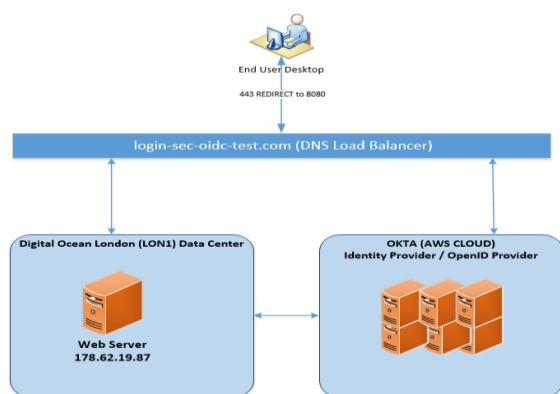


Figure 7: Application Infrastructure

Access control to the web application can be achieved by the administrator of the Okta Tenant who can create the users manually. They also bulk upload leveraging the Okta API using curl or postman, using API keys for authentication and set the tenant to allow any user to register through the web app front end. With any of these options the user upon the first login, if they are found to not have a device registered for the two-factor authentication, as part of that login process they will need to do so to gain access to the application. Any new user registrations can also be auto-provisioned so they are added to the user group. This allows granular control.

3.1. Identity Providers and OpenID Providers

An identity provider (IDP) is a system entity that creates, maintains, and manages identities and information for principals while also providing authentication services to service providers and relying party applications in a federated or distributed network. Identity providers offer end-user authentication as a service. In reality, it is software that implements the identity provider part of the security access markup language (SAML for short) or OpenID Connect Federation protocols. Popular identity providers that reside online are Google, Instagram, and Amazon Web Services (AWS). Whereas for corporate or enterprise usage some popular IDP's are Azure AD, Okta, and PingIdentity. An OpenID Provider (OP), on the other hand, is an intermediary between an identity provider and applications wishing to authenticate end-users. An OpenID Provider (OP) is something that allows applications to integrate with identity providers to provide single sign-on for its end-user base. It also allows an application to integrate with multiple Identity Providers. Most vendors now provide IDaaS solutions (Identity as a Service) so one option would be outsourcing all end-user account management and

security to a third-party vendor. Some may find this simple and also cost-effective. Depending on the size of the company/enterprise/corporation they may decide to stand up an Identity Provider that can be managed in-house by their internal security team. With most Identity Provider Software nowadays they come bundled with adapters that have many OpenID Connect capabilities.

3.2. OpenID Connect - Single Sign-On

Single sign-on is an authentication method that allows end-users to log into multiple resources and applications with one set of credentials. More often than not end-users will only need to authenticate once per session unless they switch browsers, clear cookies, or invoke an authentication request from a new device. In the world we now live in end users expect this type of single sign-on experience from all web-based applications. This does come with its challenges and hurdles however with OpenID Connect and OAuth 2.0 they have set the Single Sign-On baseline as being a viable standard for a World Wide Web. In OpenID Connect, session management is accomplished using secure cookies. End-user sessions start when the Relying Party receives and then validates the Identity Token for the end-user. From that moment the session is maintained using the cookie. Figure 8 illustrates an end-user OpenID Connect Single Sign-On.

3.3. Integrations of the OIDC protocol

OpenID Connects strength lies through its widespread acceptance as an authentication standard and this allowed implemented improvements to help with strengthened security and standardization. All the tools required for an OIDC implementation are listed in this section. This will cover the interaction between Relying Parties, OpenID Providers, and End Users. Attributes used The OpenID specification specifies the attributes used by both the OpenID Providers and the Relying Parties. As with any Single Sign-On integration, some OIDC providers will require dynamic, out of band and static configurations. Others will have the luxury of using pre-configured installations. For an integration that does not use a pre-configured relationship between an OpenID Provider and a Relying Party, the specifications around OpenID Connect Discovery and OpenID Connect Dynamic Client Registration must be followed. When a Relying Party integrates into an OpenID Provider they may have certain configuration settings/features specified as mandatory that must be implemented on the Relying Party side. But there will be also some marked as optional.

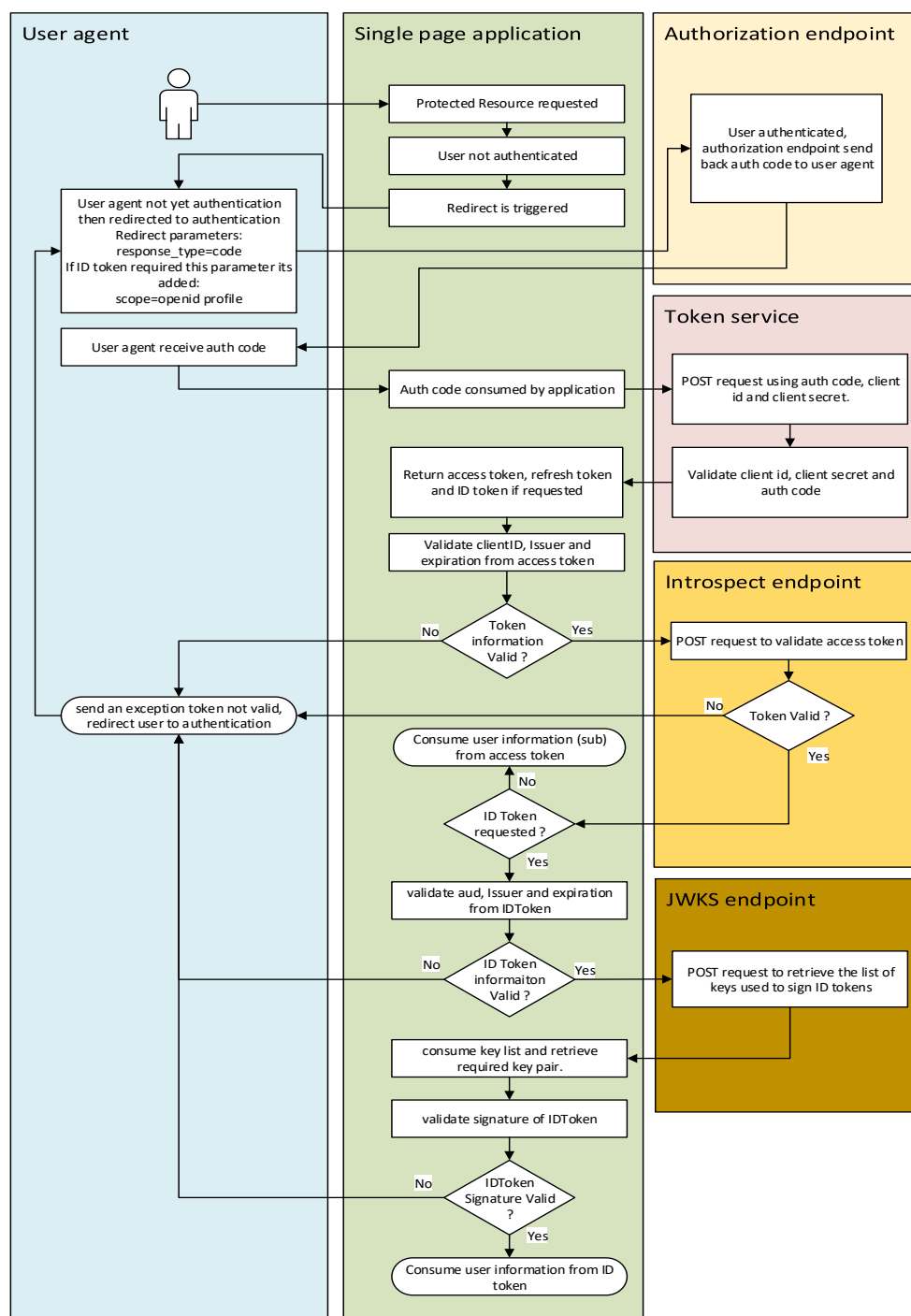


Figure 8: OpenID Connect Single Sign-On Flowchart

The OpenID Connect protocol specifies that the distinction between required and optional features is critical because it is what determines what security features must be enabled versus what security features can be used by the relying party if desired. Also, much analysis into the OpenID Connect protocol has shown that whilst the specification is logically strong, the lack of alignment with the specification and deployment flaws is the biggest factor of security breaches. Most of this will come down to what type of OAuth Client is in play, for

example, a client that would be classed as confidential would be the best use of the Authorization Code Flow. Authorization Code Grant Type Flow would be as follows. An end-user browses to an application that is integrated via OAuth with an OpenID Provider.

The Application identifies the end-user and redirects them back to the Authorization Endpoint. An example redirects within a browser to an Authorization Endpoint would look similar to the following. In the following example, there is a nonce

parameter value specified. However, this is optional. It is a string that is used to associate the Client session with an ID Token and to mitigate replay attacks [12]. Scope, response_type, client_id, redirect_uri and state values are required. The end-user is then authenticated and the OpenID Provider would return the requested authorisation code and state back to the redirect URI.

Once the application has the Authorization Code, it can then be used to get an access token. This HTTP POST needs to be authenticated using the client_id and client_secret obtained during the integration with the OpenID Provider. A sample Token Endpoint would look like <https://sample.com/as/token.oauth2>. The following parameters are required within the request: client_id, client_secret, grant_type, code and redirect_uri. Access tokens should then be validated against the Introspect Endpoint. Authentication is required by presenting the client_id and client_secret. The Introspection Endpoint will respond with a JSON with an 'active' flag which is a Boolean value of whether or not the presented token is currently active. The value will be "true" if the token has been validated successfully. ID Tokens (received when using OpenID scope) will also require validation by the application. Once validated user information can be consumed from the ID Token (email claim) and the application can authorize the user to access resources. The validation of the ID token includes evaluating both the payload and the digital signature [13].

3.4. OpenID Connect overview

Figure 9 illustrates how the sign-on process will look from a technical standpoint for this project. Using the OpenID Connect protocol for user authentication is far more superior to most other offerings on the market today. The Resource Owner in this scenario will be the end-user trying to access the application. The client is the configuration stored in the server where the web app resides. This holds all of the information to initiate the SSO request. Okta's Authorization Server will act as a token endpoint and the resource server is where the web application will be hosted. The application itself will run on port 8080 over HTTPS, however, iptables are used to create a rule to direct traffic from 443 to 8080. This allows for a nice user experience as no ports will reside in the URL string.

3.5. OIDC system protocol flow

Figure 10 describes each step taken on how an end-user will be authenticated by the Okta service. This would not be possible without the OpenID Connect integration from web application to the Okta

identity service. It also shows the two-factor authentication challenge prior to any application authorisation. The application decodes the token and looks for the users ROLE attribute that is in the id_token. This is used to determine what the user can or cannot do.

The user accesses a protected resource and is redirected to Okta for authentication.

- Okta authenticates the user and generates an authorisation code from the authorisation endpoint after validating the client ID and client credentials.
- The application makes a backend call to the token endpoint of Okta with the short-lived authorisation code.
- Okta issues Access Token and ID Token. ID Token provides user attributes.
- These tokens are validated and decoded by the spring library and user attributes are sent in a JSON format response.
- The application reads those user attributes and manages the session of the authenticated user.

3.6. OIDC User Story

The use cases/user stories talk - about how users communicate with the third-party Identity provider (Okta) to gain access to any application integrated with them to prove single sign-on.

Stake Holders: End-Users, Secure Web Servers, Third-Party Identity Provider, Identity Repository

Brief Description: This user story explains how after a user has successfully authenticated to the web application integrated with Okta, from that point as long as the user has an active session they will successfully be able to access other applications that are integrated with the same Okta tenant without any authentication challenge.

Goal: Obtain an id_token, access token, and refresh token.

Successful Result: The end-user can access the protected resources required.

Prerequisites:

- The user has an account created in the Okta user repository
- Application is running and accessible
- Application is integrated and protected by Okta

Initiation: End-user accesses the web application from their machine.

Standard Flow of events:

- End-user goes to the web application and clicks Logon.

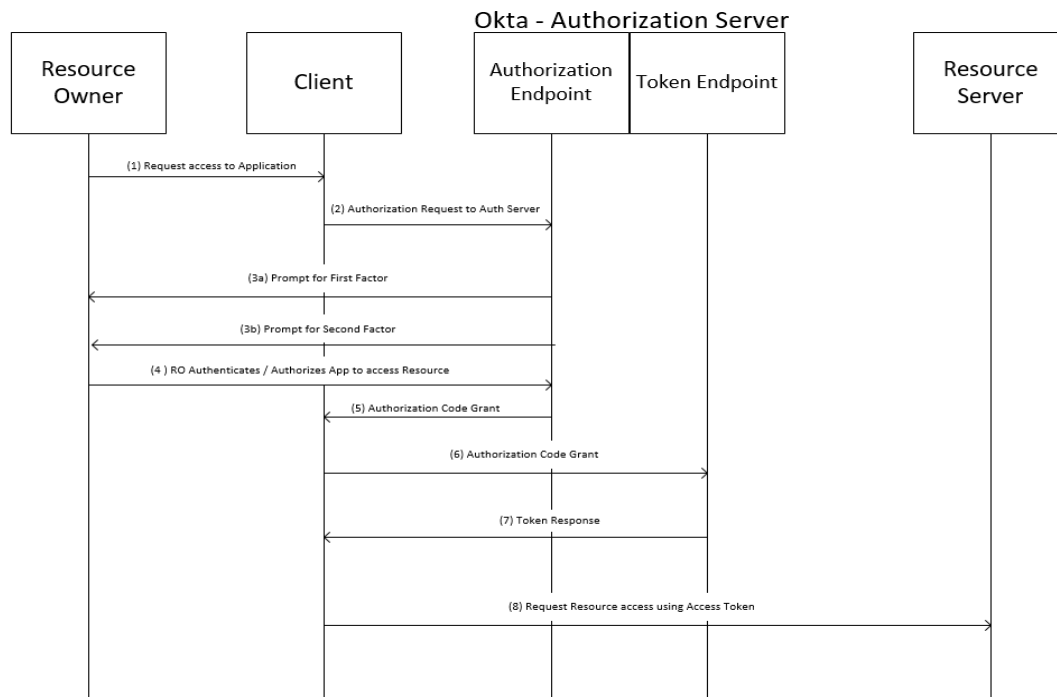


Figure 9: High-level end-user authentication flow

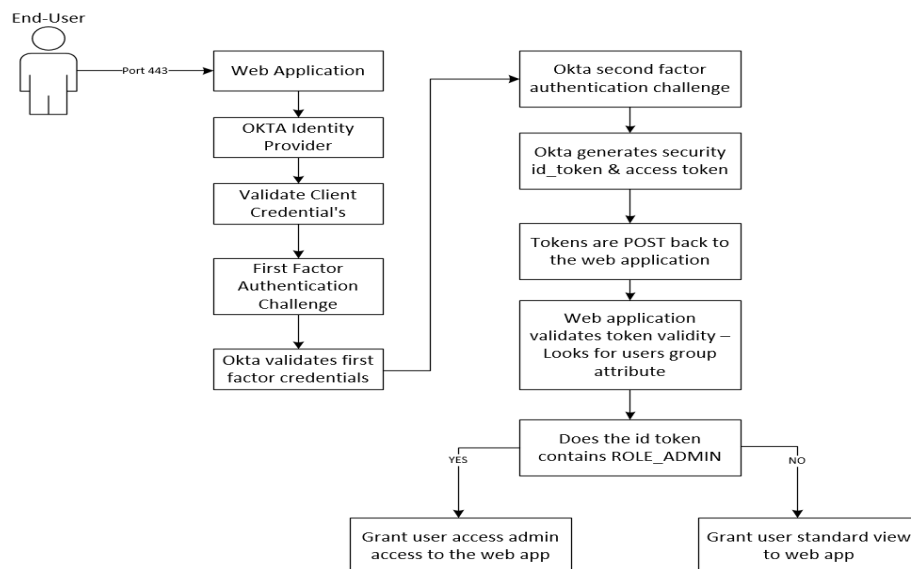


Figure 10: OIDC High-Level Authentication Flow

- ii. The application redirects the end-user to the Okta email and password login form.
- iii. The user enters their credentials. These credentials are verified by Okta.
- iv. If the first factor was a success the user is then challenged for a second-factor authentication.
- v. End-User then completes the second-factor authentication challenge.
- vi. Okta validates the second factor and then posts the id token and access token to the application.
- vii. The application then looks at the id token attributes, based on the user group (Admin versus User) the correct view then loads.

Assumptions: The user provides valid credentials.

Alternate Flow: Bad credentials will not be accepted. End-user will be redirected back to the web application front end.

Post Condition: The end-user was able to access the protected resource correctly, no errors encountered.

3.7. OpenID Connect Best Practices

In light of the versatility and usability of the OpenID standard, several areas around security must be addressed with OpenID Connect Client profiles. These profiles cover browser-based (user-agent), web applications, desktop, passive and native applications (fat clients). This section will look at profile types and the threats that are associated with them and what attributes can be used to further secure an OpenID Connect implementation and or integration.

Client authentication will only occur if the authorisation server has established with web app clients, better means of client authentication other than a client password are encouraged. The protection of client passwords as well as other credentials associated with the client must be protected. The client secret that is issued by the authorisation server to the client must not be shared with any other clients. If client authentication using client secret is not possible the authorisation server must look into other ways of confirming client identity, for example, private key validation. The client can first prepare the keys and allow the Authorization server access to the public key. The client then signs the JSON with the private key before it is transported to the authorisation server. Upon receipt, the authorisation server token endpoint will extract the assertion from the client and then verify the JSON using the public key. Another option is mutual TLS client authentication. With this validation, both the client and the token endpoint must use mutual TLS. Any token that contains information/attributes/credentials belonging to an end-user or entity must be kept confidential in transportation and storage, and only exchanged with the authorisation server, the client, and the resource server. All transportation must be done over HTTPS when using OpenID Connect or OAuth 2.0 flows. Implicit Grant flow requires the access token to be transported in the Uniform Resource Identifier Fragment leaving this open to exposure to unwanted parties. Authorization Servers have to ensure that unauthorized parties are unable to generate, modify, or guess access tokens to produce valid access tokens. Clients should use minimal scopes when requesting access tokens. When choosing how to honour the requested scope, the authorisation server should take into account the client identity and may issue an access token with fewer rights than requested. Authorization codes should be sent over a protected connection, and all redirect URI should make use of HTTPS over TLS. As authorisation codes are distributed via browser redirects, they may theoretically be exposed via web browser history and headers of the HTTP referrer. These codes must be short-lived and used only once. If an authorisation server sees multiple requests to exchange a code for a token all tokens should then be revoked for that

client.

The authorisation code grant type allows the client to specify a redirect URI through the use of the `redirect_uri` parameter. This opens up an attack for sophisticated actors to manipulate the redirect URI parameter which will then force the authorisation server to redirect the end-user to the bad actors specified URI. A bad actor can setup and integrate a client to kick off an authorisation flow. The bad actor can then initiate an authentication request from their browser to the authorisation server. Doing this will allow them to obtain the authorisation URI from the genuine client and then replace that with their redirect URI which they control. This allows the bad actor to manipulate the end-user which in turn will allow access to the client. After the authentication request reaches the authorisation server the end-user will be asked to present a credential to confirm their identity. Once validated the end user will then be redirected to an endpoint that is controlled by the bad actor with the authorisation code. With this authorisation code now in the control of the bad actor, it will allow them to post this code to the client using the original redirection URI provided by the client. The client will then take this code and provide an access token to the bad actor allowing them to access the end-users protected resource. This type of attack can be avoided if the authorisation server ensures that the redirect URI used to retrieve the authorisation code is identical to the redirection URI gave when the authorisation code is exchanged for an access token. For any client that is public-facing, a redirect URI must be provided. Any redirect URI is that is presented in the request must be verified against the redirect URI that is configured at the authorisation server.

Older integration often uses the Resource Owner Password Credentials Grant type. With this type of integration, the credentials used (i.e. username and password) can obtain the access token. The risk factor is reduced because the client does not store these credentials however if these credentials were to be known by a bad actor they would have full access to the token exchange. It's important that if this grant type has to be used, the authorisation server should narrow down the scope and the lifetime of the tokens issued. All tokens (Access, Refresh, and ID), passwords, secrets, authorisation codes and client credentials must not be transported in clear text. Any information in the JSON Web Token must not include any sensitive information about the client of the resource owner in plain text. Nothing should be transmitted over insecure channels or stored in an unsecured manner.

4. Implementation

One of the many requirements of this study is to answer the question is the solution secure. Anything to do with single sign-on or the overall IAM stack

always circles back to security because without it you are leaving your endpoint/application open to a worldwide web where users, data, digital assets, etc will all become compromised. In today's world, it is not just about TLS/SSL between endpoints because at times that is not even safe [14]. Sophisticated actors have found a way to use SSL to their benefit., meaning it is no longer enough to trust that encrypted data transmission is completely secure since we have the CIAA principles, so it's important to understand that when implementing OpenID Connect it's done correctly using the correct flow for your endpoint and also using the tokens as they were intended. So when it comes down to designing a secure IAM solution it requires careful thought and planning. OpenID is the fastest, easiest, and most secure way to sign in to websites [15]. Nothing is unbreakable however there are no major security flaws with OIDC, the only flaw that will occur is using this protocol incorrectly, or not implementing using the OIDC specification. To avoid man-in-the-middle attacks, for any request sent to authorisation and token endpoints, the authorisation server must use TLS with server authentication. The client must verify the TLS certificate of the Authorization Server as specified by the Public Key Infrastructure Specification and in compliance with its server identity authentication specifications. If a bearer token was to become compromised by a hacker it could lead to them having access to multiple resources. So these tokens must be shielded from exposure in the possession and transportation to avoid misuse by having a comprehensive client secret. Doing so will allow the client/authorisation server to know when the token has been tampered with [16]. Configurations of clients must ensure tokens are not exposed to unwanted parties during

transit or storage, since they can be used to obtain access to secure services. TLS must always be used when making requests with bearer tokens, if not this exposes the token to multiple attacks if compromised. Token servers SHOULD issue short-lived (one hour or less) bearer tokens, particularly when issuing tokens to clients that run within a web browser or other environments where information leakage may occur. Using short-lived bearer tokens can reduce the impact of them being leaked [17]. URLs (or cookies) should not contain bearer tokens, instead, HTTP message headers should contain the tokens. If bearer tokens are passed in page URLs, attackers might be able to steal them from the history data, logs, or other unsecured locations and use replay attacks [17].

The rest of this section goes through some aspects of the setup of the web application, the integration with the third-party identity provider (Okta) using OpenID Connect, test user account setup, and authentication policy creation. The web application will be built on a Nodejs server using an open-source application generator called JHipster. The application is configured to run over HTTPS on port 8080, however, an iptables rule on the server will redirect all 443 traffic to port 8080, with this rule takes away the requirement to include the port in the web app URL. The main function of this application is to have two roles, Admin and standard users. During the implementation the Okta tenant will be configured to also have these two groups, users will then be added to these groups, and post a successful authentication the group the user is part of will be added in the id_token. The web application will then provide the correct level of rights for that authentication session based on that claim. The authentication flow can be seen in Figure 11.

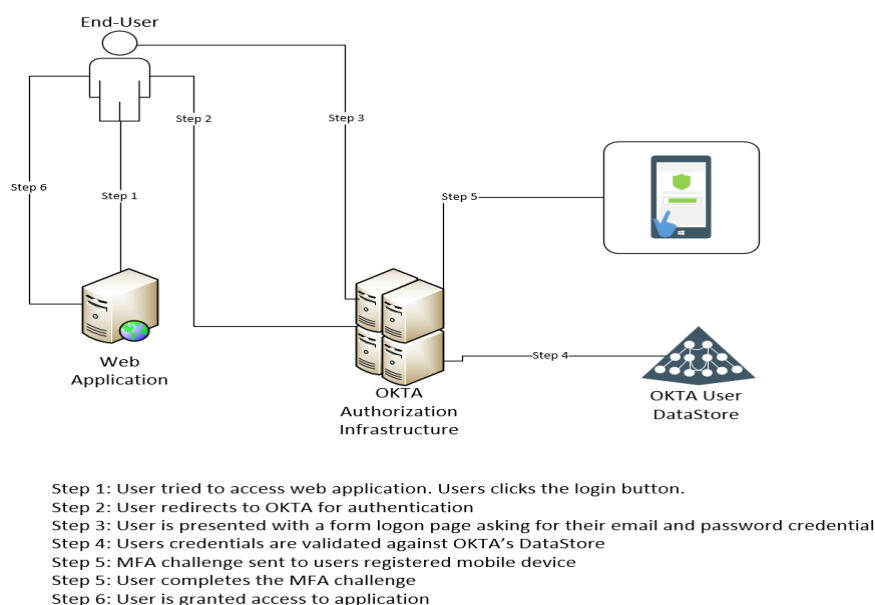


Figure 11: End to End authentication flow

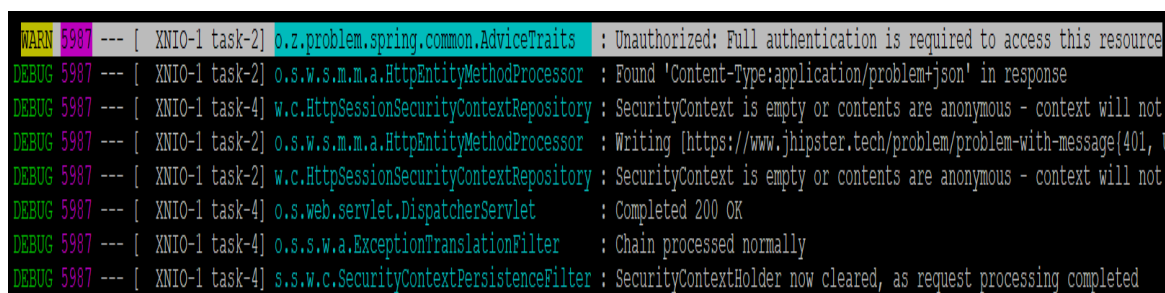
The hosting provider we used was DigitalOcean. The domain was purchased from namecheap. SSH keys were created. Apache was installed on Ubuntu. The main purpose of SSL is to safeguard sensitive information through the use of encryption when in transit across the World Wide Web. The web app was built using JHipster which is a free and open-source, multi-platform, free to download application generator used to develop web apps. The technologies used are based on microservices using Angular / React and the SpringBoot Framework.

At this point, the application has been installed, however before it can run the OAuth 2.0 / OpenID Connect configuration needs to be complete. By default, JHipster creates a Docker container, and in that container resides an opensource identity and access management application called Keycloak. Our web application is integrated with a third-party Identity Provider (Okta). As part of single sign-on for this web application end, users need to have a credential to perform the first factor (email and password). For the second-factor authentication, they need to bind something to their identity. Account creation/provisioning will be handled in the Okta tenant, this can be done using self-service registration (allows any user to register for the app) or manual provisioning (controlled by the application administrator). Self-service is disabled. It is controlled by the Okta tenant. For added security MFA is used to add an extra layer of security for this web application. With Okta, it allows multiple types of MFA for the end-users: Okta Verify, SMS Authentication, Google Authenticator, Symantec VIP, on-Prem MFA, RSA SecureID, or Email Authentication.

5. Evaluation

The purpose of this testing is to demonstrate how the framework has been implemented serves its purpose, which is to protect the user's data by strong authentication mechanisms followed by granular authorisation using the ID Token. All testing carried out is done to satisfy the research question. It will also show the benefit of MFA and how it can easily protect users' data.

The app runs in development mode so all events/transactions are printed to the console. This allows viewing of the authentication tokens, access tokens, and id tokens. During the configuration of the web app, OIDC/OAuth authentication was selected as the authentication mechanism. This means all local logins that come as standard with this JHipster build are redundant. During the implementation section test user account sso.help21@gmail.com was set up and then assigned to the Admin Group in the Okta Tenant. This same account will now be used to login into the application. After the user has completed the two-factor authentication sequence the access token and id token will be printed out to the console allowing us to view the claims. As this user was added to the admin group they will have full admin rights in the web application. The following steps will confirm the same. Using chrome browser navigate to <https://login-sec-oidc-test.com/>. The following was printed to the console log. Please note the line highlighted in Figure 12 which indicates the user access the web application does not have a valid authentication session. The user is then redirected to the Okta login form as seen in Figure 13.



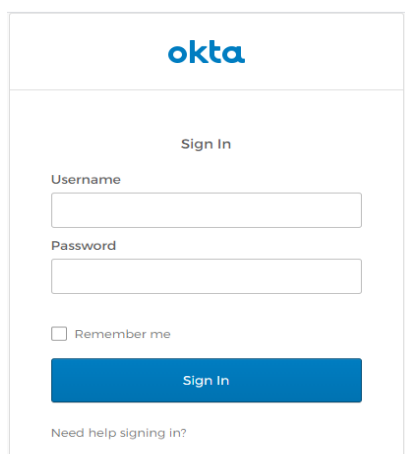
```

WARN 5987 --- [ XNIO-1 task-2] o.z.problem.spring.common.AdviceTraits : Unauthorized: Full authentication is required to access this resource
DEBUG 5987 --- [ XNIO-1 task-2] o.s.w.s.m.a.HttpEntityMethodProcessor : Found 'Content-Type:application/problem+json' in response
DEBUG 5987 --- [ XNIO-1 task-4] w.c.HttpSessionSecurityContextRepository : SecurityContext is empty or contents are anonymous - context will not
DEBUG 5987 --- [ XNIO-1 task-2] o.s.w.s.m.a.HttpEntityMethodProcessor : Writing [https://www.jhipster.tech/problem/problem-with-message[401, t
DEBUG 5987 --- [ XNIO-1 task-2] w.c.HttpSessionSecurityContextRepository : SecurityContext is empty or contents are anonymous - context will not
DEBUG 5987 --- [ XNIO-1 task-4] o.s.web.servlet.DispatcherServlet : Completed 200 OK
DEBUG 5987 --- [ XNIO-1 task-4] o.s.s.w.a.ExceptionTranslationFilter : Chain processed normally
DEBUG 5987 --- [ XNIO-1 task-4] s.s.w.c.SecurityContextPersistenceFilter : SecurityContextHolder now cleared, as request processing completed
  
```

Figure 12: Console log output - no valid session

The console log shows the redirect to the OpenID provider seen. A URL is constructed by the application, this indicates the start of the authorisation flow. The end-user then enters their email address and password as seen in Figure 14. Okta will verify the user's credentials against their datastore. Upon a successful first factor authentication, the end-user will be challenged for the second factor as seen in Figure 15. For the second factor challenge, this specific user tied their identity using the Google Authenticator app. At this

point, the user has completed the login challenges, and the application now has the authorisation code that will obtain the access token. The application will POST the request to the authorisations server endpoint with the parameters grant_type which lets the endpoint know to use the Authorization Code grant type; Code which is the code that was given in the redirect and redirect_uri which is the same uri given when requesting the code. These parameters in the request will be validated by the token endpoint. If all checks out, an access token will be generated and



Okta

Sign In

Username

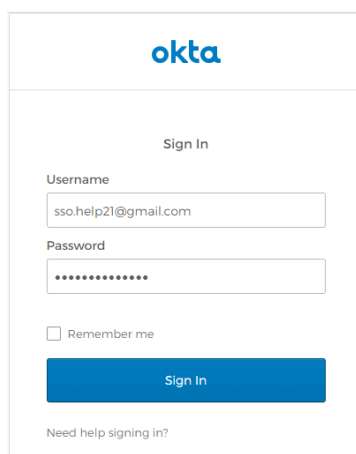
Password

☐ Remember me

Sign In

Need help signing in?

Figure 13: Okta login Form



Okta

Sign In

Username

sso.help21@gmail.com

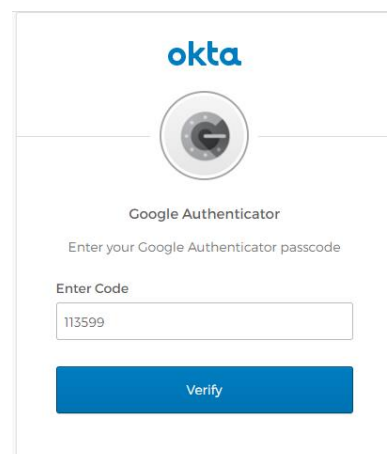
Password

☐ Remember me

Sign In

Need help signing in?

Figure 14: First-factor challenge



Okta

Google Authenticator

Enter your Google Authenticator passcode

Enter Code

113599

Verify

Figure 15: Second-factor challenge

```
2021-01-07 17:43:59.031 DEBUG 5987 --- [ XNIO-1 task-3] .s.o.c.w.OAuth2LoginAuthenticationFilter : Authentication success. Up
dating SecurityContextHolder to contain: org.springframework.security.oauth2.client.authentication.OAuth2AuthenticationToken@9
ae09bd6: Principal: Name: [00u15ant1pdW1znwx5d6], Granted Authorities: [[ROLE_USER, SCOPE_email, SCOPE_openid, SCOPE_profile]]
, User Attributes: [{at hash=VyB-y8bLhxMBCHikLBwgAg, sub=00u15ant1pdW1znwx5d6, zoneinfo=America/Los_Angeles, ver=1, email veri
fied=true, amr=["mfa","pwd","otp"], iss=https://dev-7201054.okta.com/oauth2/default, groups=["Everyone","ROLE ADMIN"], preferre
d username=sso.help21@gmail.com, locale=en-US, given_name=Richard, nonce=sCbPShRmThbPCLk67ahFKVvg1Ugt1jRHkmUvmWzJVqW, aud=[00
a1561xxNb1s0fnf5d6], updated at=2020-11-18T22:08:26Z, idp=00u15anp9nlovSNSL5d6, auth_time=2021-01-07T17:43:50Z, name=Richard M
adden, exp=2021-01-07T18:43:58Z, family_name=Madden, iat=2021-01-07T17:43:58Z, email=sso.help21@gmail.com, jti=ID.vz2oi-HH8UDc
R5jCabHqUpzwpz19mgHdwMXoFikTq38}], Credentials: [PROTECTED]; Authenticated: true; Details: org.springframework.security.web.au
thentication.WebAuthenticationDetails@fffe9938: RemoteIpAddress: 165.225.196.226; SessionId: 5xtmLe56eaGlpz2AXgDlKoQdCRR3Bnnvo
40hg3dl; Granted Authorities: ROLE_ADMIN
```

Figure 16: ID Token issued by the Authorization Server

returned in the reply. Point to note, the authentication has been a success and the authorisation server will now issue an id token to the application. This id token contains the user's attributes, including the user's ROLE. The granular authorisation should be performed from the claims in this token and not the access token.

Inspecting the token it has the user's email address, location, full name, and groups associated with them. As you can see in Figure 16, this user has been assigned to the admin group, meaning they will have full administration rights to this application. The authorisation server will now redirect the user back to the redirect URI that was specified in the construction of the URL when the authorisation code flow was initiated and the user has full admin privileges.

The results of these tests show that when OpenID Connect, OAuth 2.0 Authorization Code Flow, and Multi-Factor Authentication is integrated and used in the correct way users and the endpoint are protected. In the tests carried out, the exchange of the access token for the id token can be seen after successful authentication. Then based on the claims passed to the web application in the id token the web application decides as to what type of granular authorization is to be performed. In this test, it was a

simple admin vs user scenario. In the final test, email spoofing was performed and successfully captured a user's credentials. With these credentials now compromised it allowed the hacker to attempt a login to the web app, however, because Multi-factor authentication was in play this protected the user and their data. When it comes to authentication it's important to be in the mind-set of something you know and something you have. That way if something you know is compromised (password) the something you have will act as your safety net. And it's the same for authorization, when developing a web application you can't just assume that if the user passed the authentication process then they are safe and they should have access. If you think of a real-world scenario between a consultant, doctor, and patient who all have access to a system. In this system patients can log in and view medical records but what if that patient is waiting for results on a particular blood test or scan and the consultant and doctor do not want to allow the patient to see these records until after the consultation. This is where granular authorization comes into play because all three parties can still log in using their credentials but by using the claims in the id token to control what level of access the user has/what is presented to them on screen.

6. Conclusion

In this paper covered numerous Identity & Access Management protocols and solutions that are used within many organizations today but often used in the incorrect way. When it comes to security for applications, one will hear the term SSO but what does that mean. If one picks it apart SSO is for authentication only and not authorization because it allows users to log in and have access to whatever is inside the endpoint. But Identity & Access Management covers authentication and authorization so when it comes to designing a web application instead of just thinking about SSO think about the whole Identity & Access Management framework for the application. The reason OpenID Connect, OAuth 2.0, and Multifactor authentication were chosen for this project is to showcase the importance of implementing IAM the correct way. We show that OpenID Connect is an SSO solution but OAuth 2.0 is not, however many feel it is because authorization could also be used for authentication. This goes fully against the OAuth 2.0 specifications which state OAuth 2.0 is perfect for conveying authorization decisions between web applications and REST endpoints. OAuth 2.0 can be found in a broad range of implementations, including offering user authentication mechanisms. And because of that developers, service providers, IT professionals, and security professionals have been led to wrongly believe it is an authentication protocol. In addition to OAuth 2.0, it is a rich authorization framework with many flows. For example, in a scenario where a scheduled job leverages an API that exports/imports data to a database this is a machine-to-machine authorization and this type of integration could leverage the Client Credential Flow. Another example is a native application. If you decompile a native application it would clearly show the Client Secret (cannot securely store the secret) and it may also use custom URL schemes that log redirects allowing untrusted applications to receive the Authorization Code. So again OAuth 2.0 provides a version of the Authorization Code Flow but it uses something called Proof Key for Code Exchange (PKCE). The main focus of this project was protecting web applications/endpoints/users and that is why these three protocols were chosen. Building the web application was done using a free open-source application generator called JHipster. This application generator was chosen because not only does it provide front-end and back-end services but it also has a built-in authentication (OIDC) and authorization (OAuth 2.0) module making it very easy to achieve IAM compliance. Initial tests show the importance of the access token and the id token when making authorization decisions. In the last test, the importance of multifactor comes into play where, even with a compromised account, the sophisticated

actor cannot gain access to the web application. Of course, as with any security implementation, it must be done following documented policies and standards. For OpenID Connect following the OpenID[.]net specification for the implementation will ensure the correct implementation methods are used for the endpoint/web application and as an added security layer Multi-Factor Authentication will help safeguard the end-users.

7. References

- [1] Wiki, (2020). Identity management, https://en.wikipedia.org/wiki/Identity_management. (Access Date: 12 July 2020).
- [2] OpenID. Benefits of OpenID. 2020. [Online]. Available: <https://openid.net/individuals/>. (Access Date: 12 July 2020).
- [3] Chen, S. (2012). Signing Me onto Your Accounts through Facebook and Google. <https://ieeexplore.ieee.org/abstract/document/6234424>. (Access Date: 2 February 2021).
- [4] Hardt, (2012). The OAuth 2.0 Authorization Framework, <https://tools.ietf.org/html/rfc6749#section-10.15>. (Access Date: 12 April 2020).
- [5] OpenID. (2014). Welcome to OpenID Connect. <https://openid.net/connect/>. (Access Date: 2 February 2021).
- [6] Bhati, K. (2019). Authentication and Authorization to Secure API's. <https://nl.devoteam.com/en/blog-post/authentication-authorization-secure-apis/>. (Access Date: 2 February 2021).
- [7] O Foundation. (2015). OpenID Foundation. <https://openid.net/foundation/#:~:text=The%20OpenID%20Foundation%20is%20a,developers%2C%20vendors%2C%20and%20users>. (Access Date: 14 September 2020).
- [8] Hunt, P. (2020). OAuth 2.0 Threat Model and Security Considerations. <https://datatracker.ietf.org/doc/rfc6819/>. (Access Date: 3 June 2021).
- [9] J. Bradley (2020). Web Authorization Protocol. <https://tools.ietf.org/id/draft-ietf-oauth-security-topics13.txt>. (Access Date: 2 June 2021).
- [10] Connect2id. (2020). Standard OAuth 2.0 / OpenID Connect endpoints. <https://connect2id.com/products/server/docs/api>. (Access Date: 2 July 2021).
- [11] Auth0. (2020). JWT.IO. <https://jwt.io/>. (Access Date: 1 July 2021).
- [12] OpenID. (2014). OpenID Connect Core. https://openid.net/specs/openid-connect-core-1_0.html#CodeFlowSteps. (Access Date: 4 July 2021).
- [13] Ping identity. (2020). OAuth 2.0 Developers Guide. <https://www.pingidentity.com/developer/en/resources/oauth-2-0-developers-guide.html>. (Access Date: 1 July 2021).

[14] Gartner. (2020). Radware. <https://www.gartner.com/imagesrv/media-products/pdf/radware/Radware-1-2Y7FR0I.pdf>. (Access Date: 2 August 2021).

[15] OpenID.net, "Benefits of OpenID," 2020. [Online]. Available: <https://openid.net/individuals/>. (Access Date: 2 August 2021).

[16] Urban. C. (2019). Formal analysis of Facebook Connect. <https://caterinaurban.github.io/pdf/sofsem2011.pdf>. (Access Date: 2 August 2021).

[17] Hardt J. (2012). OAuth 2.0 Bearer Token Usage. <https://tools.ietf.org/html/rfc6750#page-10>. (Access Date: 1 September 2021).

[18] OKTA, (2017). Security Considerations. <https://www.oauth.com/oauth2-servers/authorization/securityconsiderations/>. (Access Date: 12 September 2020).

[19] Shopper, S. (2008). What is a CSR. <https://www.sslshopper.com/what-is-a-csr-certificate-signing-request.html>. (Access Date: 11 September 2020).