Future of Identity and Access Management: The OpenID Connect Protocol

---

A Thesis Presented to

The Faculty of the Department of

Information and Logistics Technology

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Information System Security

By

Omer Ofleh

August 2018

ABSTRACT

As the Internet becomes the standard, and often the only, mechanism for interactions between individuals, private companies, governments and other organizations, digital identity management is exceedingly a critical component of this online communication and commerce. Identity and Access Management (IAM) is the management and control of information about users in a digital format. This information may include mechanisms to verify the identity of the users (authentication) and ensuring approved access to resources (authorization). In addition, IAM maintains descriptive details about users and provides portability of this information between disparate systems.

This thesis explores the OpenID Connect (OIDC) standard introduced by the OpenID Foundation. Built on an earlier standard known as OAuth 2, the OIDC standard, also referred to as a protocol, specifies a near-complete procedure to provide authentication and authorization of users across the Internet. The first part of the thesis introduces the protocol while later parts explore the problems it purports to solve. Finally, a study of current implementations, the adoption of the OIDC standard by industry and possible ways to improve upon the standard are explored.

Table of Contents

List of Figures

1. Introduction

This thesis explores the most recent developments in Identity and Access Management (IAM) on the Internet and what lies ahead for IAM. Specifically, it explores the OpenID Connect (OIDC) standard and how it largely solves the problem of IAM across heterogenous networks such as the Internet. The thesis first describes the OIDC standard in detail as created by the OpenID Foundation, including its core functionality and important security and privacy concerns. Next, implementations of the standard are explored with major successful ones referenced as examples. The evolution of and potential improvements to the OIDC standard are also considered. In conclusion, light is shed on how and why the OIDC is so widely adopted and the fastest growing IAM protocol in practice today.

By definition, the issue of identity ownership and verification is distributed. No one entity, public or private, has a complete monopoly on all identities. Government agencies in the United States may use Social Security numbers (SSN) to identify individuals living in the country. Countries around the world may have similar or slightly different ways of identifying their citizens. Private companies may use one mechanism to identify their customers and a separate one to identify their employees. A single person may have several "identities" depending on when and where they are. A person's physical identity may be verified using a driver license and an SSN. A similar method of verification may be a passport, student identity card or an employee badge.

Similarly, several online identities may exist for a single person. One could have an online bank account, an email account or a social network account. Each of these identities points to the same person but is held at different entities (Society, 2016). While the search for a single, universal identity continues, OpenID Connect aims to create an interoperable mechanism for identity

verification across many entities, both public and private. OpenID Connect provides a trustworthy, simple and effective mechanism for individuals to identify themselves to many application and service providers using one or more of their identities stored at a trusted identity provider (IdP).

Since its unveiling in 2014, OpenID Connect has had a strong backing and wide adoption from the largest global high technology companies such as Google, Facebook and Deutsche Telekom.

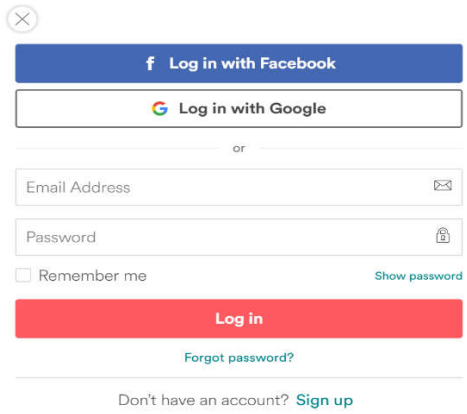

Figure 1- Use Google or Facebook Identity

OpenID Connect reduces users' frequent use of the arcane and increasingly insecure paradigm of username/password and frees application and service provides from managing user identities and only act as OpenID Connect consumers. This is feasible authentication is delegated to larger and more trustworthy enterprises that created dedicated, purpose-built IdPs. For exam2ple, users are able to provide their existing Google or Facebook identities (Wang, 2012) to any vendor that has chosen to accept OpenID Connect as an authentication protocol. Not only is this convenient for users, it also leverages the security, scalability and disaster recovery infrastructure provided by these global enterprises. This thesis will attempt to show why OIDC is the future of Identity and Access Management in the foreseeable future.

2. What is OpenID Connect (OIDC

In this section, the OpenID Connect protocol is described in detail. Starting with early attempts to create a comprehensive standard for IAM, OIDC is shown as the evolution and most successful of such attempts. In addition, the foundation that created and maintains OIDC is described. Later in this section, the core functionality of OIDC is described in detail.

OIDC is a simple identity layer that augments the OAuth 2.0 protocol to allow Clients (applications) to verify the identity of an end-user (Hardt, 2012). It accomplishes this by authenticating the end-user against an Authorization Server. In addition, it is capable of providing basic and meaningful profile information about the end-user in an interoperable and easy to implement manner.

Web applications, mobile apps, and even browser-based (JavaScript) clients are able to use OpenID Connect to request and receive information about authenticated sessions and end-users. The OIDC stand is as flexible as it is extensible. It allows participants to use optional features such as discovery of OpenID Providers, encryption of end-user identity data, and session management when appropriate.

Like its predecessor OpenID 2.0, OpenID Connect performs many of the same tasks with user-friendly APIs that can be used by native and mobile applications. In addition, OpenID Connect stipulates optional and extended mechanisms for robust encryption and signing. However, unlike OpenID 2.0 and its integration with OAuth 1.0 which required an extension, OpenID Connect is thoroughly integrated in the OAuth 2.0 protocol itself.

OpenID Connect essentially brings together the three different parties necessary to authenticate a user:

a) The User or Resource Owner

b) The Client trying to access the User's resource

c) The OpenID provider able to authenticate the User to the Client

The following diagram illustrates the relationship between the three parties.

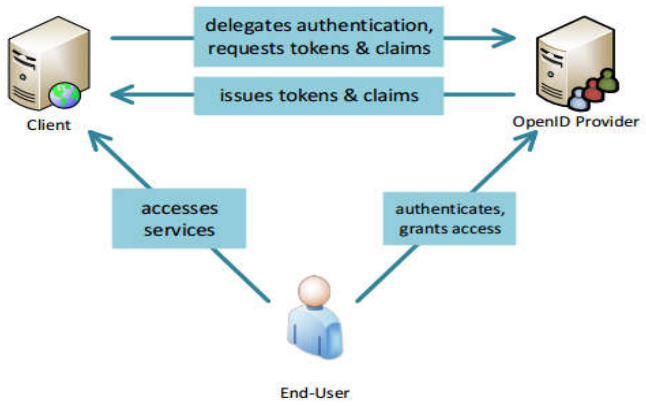

Figure 2- Relationship between Client, User and OpenID Provider

If the Resource Server is separate from the Authorization Server, the relationship is depicted below.

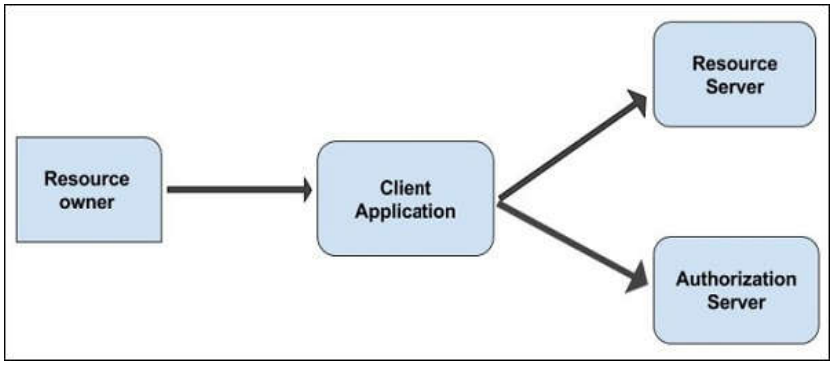

Figure 3- Relationship between parties with a separate Authorization Server

2.1 The OpenID Foundation

OpenID Foundation is the founder and publisher of the OpenID Connect standard. The OpenID Foundation (OIDF) is the guardian and keeper of the OpenID standards. It also protects and nurtures the OpenID community and technologies. OIDF is a non-profit international standardization organization of individuals and companies committed to enabling, promoting and protecting OpenID technologies (Foundation, 2014). From its formation in June 2007, the foundation has served as a public trust organization representing a large community of developers, organizations, and users. OIDF provides the infrastructure necessary to promote and support the community's expanded adoption of OpenID. This includes the management of intellectual property and brand marks as well as encouraging the growth and global participation in the adoption of OpenID.

The OIDF also enables adopters and implementors of OpenID Connect to certify their implementations to ensure conformance and to encourage interoperability among implementations. The foundation's certification process uses the novel approach of self-certification and conformance to test suites developed by the foundation. Once certified, implementations are allowed to proudly display the "OpenID Certified" certification mark.

2.2 The OAuth 2.0 Authorization Framework

The OAuth 2.0 authorization framework, upon which OpenID Connect is built, enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by initiating an approval interaction workflow between the resource owner

and the HTTP service, or by enabling the third-party application to obtain access on its own behalf.

OAuth2 came about as a result of serious flaws and fundamental limitations in traditional client-server authentication models. In such models, a request is made by a client to access a restricted resource on a server by authenticating with the server using the resource owner's credentials. To provide third-party applications access to this restricted resource, the owner of the resource shares its credentials with the third party. This is an inherently flawed paradigm (Lodderstedt, 2013) which, in addition to its limitations, raises several concerns:

- Third-party applications are must store the resource owner's login credentials for future use. This is typically either a password in clear text or a hash of the password. Furthermore, servers must support password authentication, despite the inherent security weaknesses in passwords.
- A coarsely granular access to the resource owner's resources is granted to third-party applications. This limits the resource owners' ability to restrict the duration or access to a limited subset of resources.
- The owner of the resource cannot revoke access to a single third party without revoking access to all third parties. And this can only be achieved by changing the third party's password.
- If a single third-party application is compromised, this leads to the compromise of the end-user's password and all of the data protected by that password.

OAuth addresses these issues by introducing an authorization layer and separating the role of the client from that of the resource owner. In OAuth, the client requests access to resources controlled

by the resource owner and hosted by the resource server and is issued a different set of credentials than those of the resource owner.

Instead of using the resource owner's credentials to access protected resources, the client obtains an access token -- a string denoting a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

As an example, an end-user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo-sharing service (resource server), without sharing her username and password with the printing service. Instead, with OAuth, the end-user authenticates directly with another server (authorization server) trusted by the photo-sharing service which issues the printing service delegation-specific credentials (access token).

The OAuth standard and specification are designed to be used with HTTP protocol. It has not been designed for and the use of OAuth over any protocol, other than HTTP, is not part of the standard.

2.3 OpenID Connect Core functionality

OpenID Connect implements authentication as an extension on top of the OAuth 2.0 authorization process. To indicate the use of this extension, Clients simply include the *openid* scope value in their Authorization Request (Hardt, 2012). Information about the authentication performed is returned in a JSON Web Token (JWT) called an ID Token (see Terminology). OAuth 2.0 Authentication Servers implementing OpenID Connect are also referred to as OpenID Providers (OPs). OAuth 2.0 Clients using OpenID Connect are also referred to as Relying Parties (RPs).

In its simplest form, the OpenID Connect protocol follows the steps below:

1. The RP (Client) sends a request to the OpenID Provider (OP).

2. The OP authenticates the end-user and obtains authorization.
3. The OP responds with an ID Token and usually an Access Token.
4. The RP can send a request with the Access Token to the User-Info Endpoint.
5. The User-Info Endpoint returns Claims about the end-user.

These steps are illustrated in the following diagram:

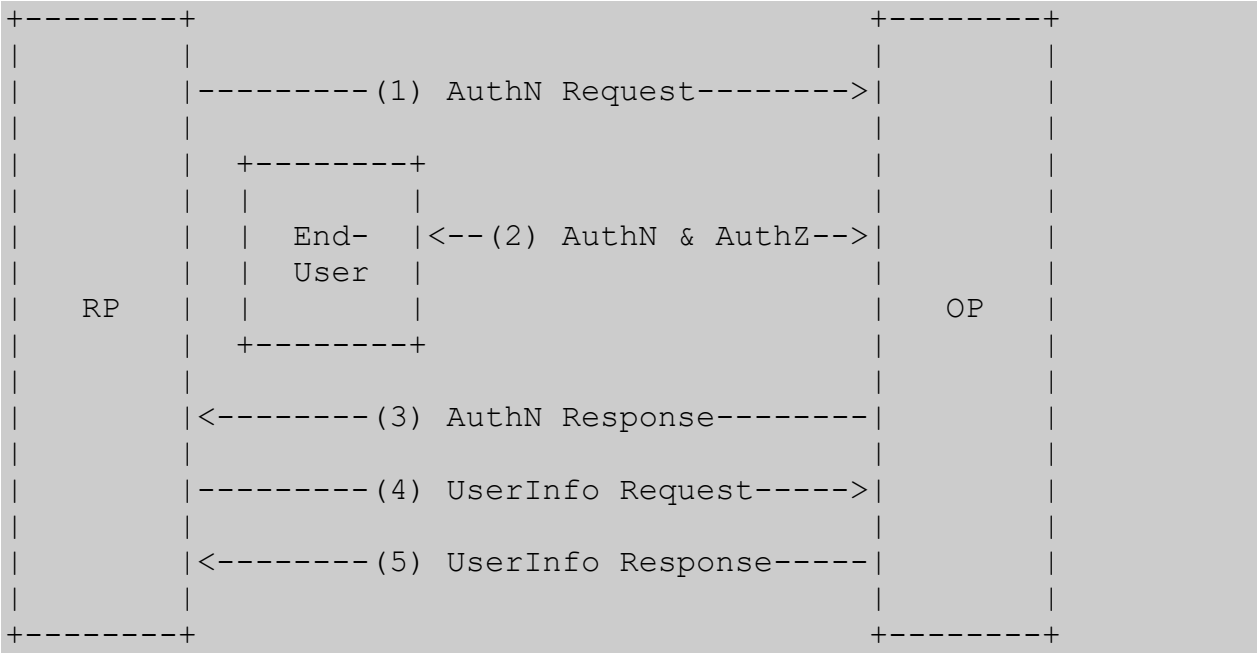

Figure 4- Authentication steps in OpenID Connect

2.3.1 Authentication

OpenID Connect performs authentication to log in the end-user or to determine that the end-user is already logged in. OpenID Connect returns the result of the Authentication performed by the Server to the Client in a secure manner so that the Client can rely on it (Hardt, 2012). For this reason, the Client is called Relying Party (RP) in this case. The secure manner of this communication is a fundamental aspect of OpenID Connect and will be explored further.

A successful authentication results in an ID Token being returned. This token contains information about the end-user and their authorizations in the form of Claims. The claims include such information as the Issuer of the token, the Subject Identifier (the end-user), when the token was issued and when it expires among other pieces of information.

Authentication can follow one of three paths:

1) Authorization Code Flow
2) Implicit Flow
3) Hybrid Flow

The authentication flows determine how the ID Token and Access Token are returned to the Client.

2.3.2 Identity Token (ID Token)

The primary extension that OpenID Connect makes to OAuth 2.0 to enable end-users to be Authenticated is the ID Token data structure (Hardt, 2012). The ID Token is a security token that contains Claims about the Authentication of an end-user by an Authorization Server when using a Client, and potentially other requested Claims. The ID Token is represented as a JSON Web Token (JWT) .

ID Tokens must be signed using JSON Web Signature (JWS) and optionally both signed and then encrypted using JWS and JSON Web Encryption (JWE) respectively. This provides authentication, integrity, non-repudiation, and, if needed, confidentiality (Hardt, 2012). If the ID Token is encrypted, it must be signed then encrypted, with the result being a nested JWT.

A sample of the claims available in the ID Token is shown in the table below. ID Tokens may contain other claims, however, any claims that are not understood or expected must be ignored.

Furthermore, in addition to verifying the signatures and cryptography within the ID Token, the token must be validated using the claims contained in it.

Claim	Type	Description
iss	Required	Issuer Identifier for the Issuer of the response.
sub	Required	Subject Identifier. A locally unique identifier for the end-user.
aud	Required	Audience(s) that this ID Token is intended for.
iat/exp	Required	Time when JWT was issued and time after which it must not be accepted.
nonce	Required	Associates a Client session with an ID Token to mitigate replay attacks.
auth_time	Optional	Time when the end-user authentication occurred.
acr	Optional	Authentication Context Class Reference.
amr	Optional	Authentication Methods References.
azp	Optional	Authorized party - the party to which the ID Token was issued.

2.3.3 Claims

OpenID Connect is known as a claims-based authentication standard. A claim is simply a statement a subject (e.g. an end-user) makes about itself or another subject. Examples of familiar claims are first-name, role and email address. Claims are issued by a provider, and they are given one or more values and then packaged in security tokens, e.g. ID Token, that are issued by an *issuer*, commonly known as a *security token service* (STS) (Hardt, 2012). This mechanism of issuing, presenting and validating tokens creates a multi-party trust relationship based on proven technologies such as federation and encryption (see Terminology).

The following is an example of the set of Claims (the JWT Claims Set) in an ID Token:

```
{
   "iss": "https://subdomain.sample.com",
   "sub": "558a5460320",
   "aud": "s6gkDtyut3",
   "nonce": "n-0Sk6_WzLE2Mj",
   "exp": 1519632560,
   "iat": 1519633560,
   "auth_time": 1311280969,
   "acr": "urn:mkce:incom:itp:sliver"
  }
```

Figure 5- Claims in an ID Token as seen in JSON format

2.3.4 JSON Web Tokens

JSON Web Token (JWT) is a compact claims representation format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. JWTs encode claims to be transmitted as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted. JWTs are always represented using the JWS Compact Serialization or the JWE Compact Serialization. The suggested pronunciation of JWT is the same as the English word "jot" (Hardt, 2012).

JWTs represent a set of claims as a JSON object that is encoded in a JWS and/or JWE structure (Barnes, 2014). This JSON object is the JWT Claims Set. A JSON object consists of zero or more name/value pairs (or members), where the names are strings and the values are arbitrary JSON values. These members are the claims represented by the JWT. This JSON object may contain whitespace and/or line breaks before or after any JSON values or structural characters.

The member names within the JWT Claims Set are referred to as Claim Names. The corresponding values are referred to as Claim Values.

The JWT Claims Set represents a JSON object whose members are the claims conveyed by the JWT. The claim names within a JWT Claims Set must be unique; A JWT with duplicate Claim Names must be rejected or a JSON parser that returns only the lexically last duplicate member name may be used.

The set of claims that a JWT must contain to be considered valid is context dependent and is outside the scope of the OpenID Connect standard. Specific applications of JWTs will require implementations to understand and process some claims in particular ways. However, in the absence of such requirements, all claims that are not understood by implementations must be ignored.

2.3.5 Identity Providers and OpenID Providers

An Identity Provider (IdP) is a server that maintains and can provide identity information to other entities. For example, Google is an Identity Provider. If you log in to a site using your Google account, then a Google server will send your identity information to that site. There are many identity providers not the least of which are corporate LDAP directories (Wilton, 2014). The largest identity providers are now online search or social media providers such as LinkedIn, Yahoo! and Facebook.

An OpenID Provider (OP), on the other hand, is an intermediary between an identity provider and applications wishing to authenticate end-users. OpenID providers allow applications to be shielded

from dealing directly with an identity provider and easily integrate with several identity providers at once.

There are three different ways an OpenID Provider can work with an Identity Provider to serve identities, each varying in complexity.

One approach is to outsource account management and security to a third party. For many companies, this approach may be simpler and more cost effective. Many vendors provide "Identity as a Service" that enables organizations to quickly implement OpenID.

A second way OpenID Connect may be used is when organizations choose to become the Provider themselves. These companies use one of the existing off-the-shelf libraries, plugins or software packages that feature OpenID capabilities out of the box (Hardt, 2012).

The third and most challenging way is for an organization to implement the OpenID Connect standard in-house and provide custom support for OpenID in its software and account management tools. This approach affords organizations a greater degree of control over the user experience but is also the riskiest as it requires deep expertise in web security.

2.3.6 Single Sign-on and Session Management

The ability to enter an end-user's credentials once and access many different resources without re-authentication is now universally accepted and expected in corporate intranets and extranets. Single Sign-on provides this experience for users within trusted networks. Expanding this experience to the wider Internet has been more challenging. OpenID Connect and its predecessor, OpenID 2.0, are the only viable standard for a decentralized, Internet-wide Single Sign-on experience.

Single Sign-on, or session management, in OpenID Connect is achieved by using secure cookies. A session for an end-user at the RP begins when the RP validates the end-user's ID Token after authentication. This session is maintained by a cookie (named *opbs* in the diagram below but may have any name). At the start of the session, the OP server will send the *obps* cookie to the RP. All the RPs sharing the same browser session will have the same *obps* cookie and the OP server will store the cookie in a database for the particular browser session.

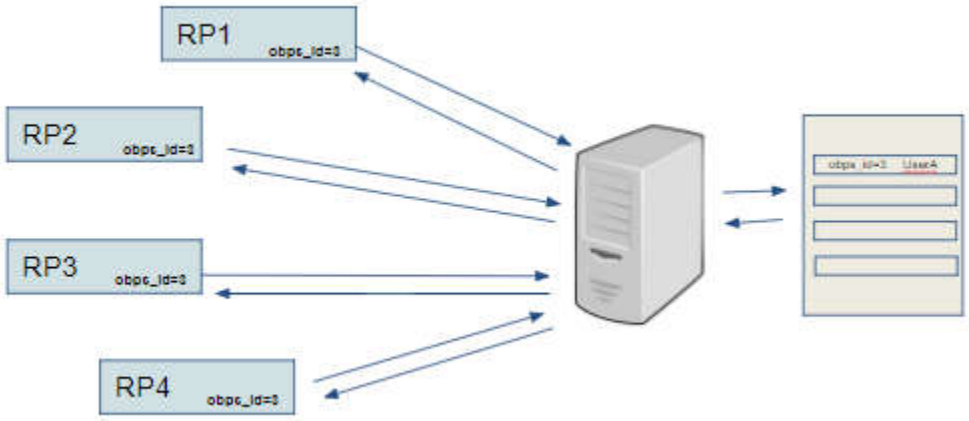

Figure 6- Single Sign-On using Secure Cookies

When the OP supports session management, it must return a Session State as an additional parameter in the Authentication Response. The RP then needs to store the value of this session state. A polling mechanism between RP and OP starts after RP sends the session state value in the authentication response. The RP polls the OP at an interval suitable for the RP application sending the session state with each polling instance. When OP receives a polling request, it accesses the *obps* cookie value at the server and recalculates the session state (Fett, 2016).

In OpenID Connect Session management, when an end-user logs out of the RP, the RP will send a logout request to the logout endpoint of the OP. The OP server will then remove the end-user session value from the database and will terminate the session of the browser.

When the session is terminated, the *obps* cookie value will be changed. When another RP within the same browser session sends a poll message with the session state value, the OP compares this session state with the recalculated value after the logout and declare they are not equal. When an RP terminates the browser session, then OP will send 'not equal' as a response to the post messages of other RPs. At this point, other RPs should re-authenticate the end-user as the single sign-on session has ended.

3. Anatomy of an OIDC Implementation

The strength of OIDC lies in its wide adoption as a standard. This allowed both early improvements in the standard and increased security and standardization. And since OIDC is built on earlier standards and protocols, it is instructive to highlight where it has improved upon these earlier standards. In this section, the ingredients of an OIDC implementation are described. Specifically, the relationship between Relying Parties, OpenID Providers and End-users is explored in greater detail.

The OpenID specification defines features used by both Relying Parties and OpenID Providers. It is expected that some OpenID Providers will require static, out-of-band configuration of RPs using them, whereas others will support dynamic usage by RPs without a pre-established relationship between them. Some OpenID Connect installations can use a pre-configured set of OpenID Providers and/or Relying Parties. In those cases, it might not

be necessary to support dynamic discovery of information about identities or services or dynamic registration of Clients.

In cases when installations choose to support unanticipated interactions between Relying Parties and OpenID Providers that do not have pre-configured relationships, they should accomplish this by implementing the facilities defined in the OpenID Connect Discovery 1.0 and OpenID Connect Dynamic Client Registration 1.0 specifications.

In general, it is up to Relying Parties which features they use when interacting with OpenID Providers. However, some choices are dictated by the nature of their OAuth Client, such as whether it is a Confidential Client, capable of keeping secrets, in which case the Authorization Code Flow may be appropriate, or whether it is a Public Client, for instance, a User Agent (Browser) Based Application or a statically registered Native Application, in which case the Implicit Flow may be appropriate (Li, 2013).

When using OpenID Connect features, those listed as being required or are described with a "must" are mandatory to implement, when used by a Relying Party. Likewise, those features that are described as "optional" need not be used or supported unless they provide value in the particular application context. Finally, when interacting with OpenID Providers that support Discovery, the OP's Discovery document can be used to dynamically determine which OP features are available for use by the RP.

The distinction between required features and optional ones in the OIDC protocol is crucial because it insists that any security related features must be supported while those secondary for security but necessary for convenience may be omitted in an implementation. The

distinction is clearly illustrated in the standard and is one of the sources of implementation deficiencies leading to lower security. Furthermore, much research into OIDC security has shown that while the standard is theoretically sound, the greatest source of security breaches has come from lack of adherence to the standard and implementation defects.

As an example, when using the Authorization Code or Hybrid flows, an ID Token is returned from the Token Endpoint in response to a Token Request using an Authorization Code (Hardt, 2012). Some implementations may choose to encode state about the ID Token to be returned in the Authorization Code value. Others may use the Authorization Code value as an index into a database storing this state. While the actual implementation details have been left to individual standard users, it is important to follow the protocol's emphasis on both required and optional features.

Another example is the *nonce* parameter value which needs to include per-session state and be difficult to guess for attackers. One method to achieve this for Web Server Clients is to store a cryptographically random value as an HTTP-Only session cookie and use a cryptographic hash of the value as the nonce parameter. In that case, the nonce in the returned ID Token is compared to the hash of the session cookie to detect ID Token replay by third parties. A related method applicable to JavaScript Clients is to store the cryptographically random value in HTML5 local storage and use a cryptographic hash of this value. The two methods described show the flexibility of OIDC standard in different contexts but the fact that a nonce is required ensures the security of the protocol and is not in optional under any scenario.

Finally, when response parameters are returned in the Redirection URI fragment value, the Client needs to have the User Agent parse the fragment encoded values and pass them to on to the Client's processing logic for consumption. User Agents that have direct access to cryptographic APIs may be able to be self-contained, for instance, with all Client code being written in JavaScript. However, if the Client does not run entirely in the User Agent, one way to achieve this is to post them to a Web Server Client for validation.

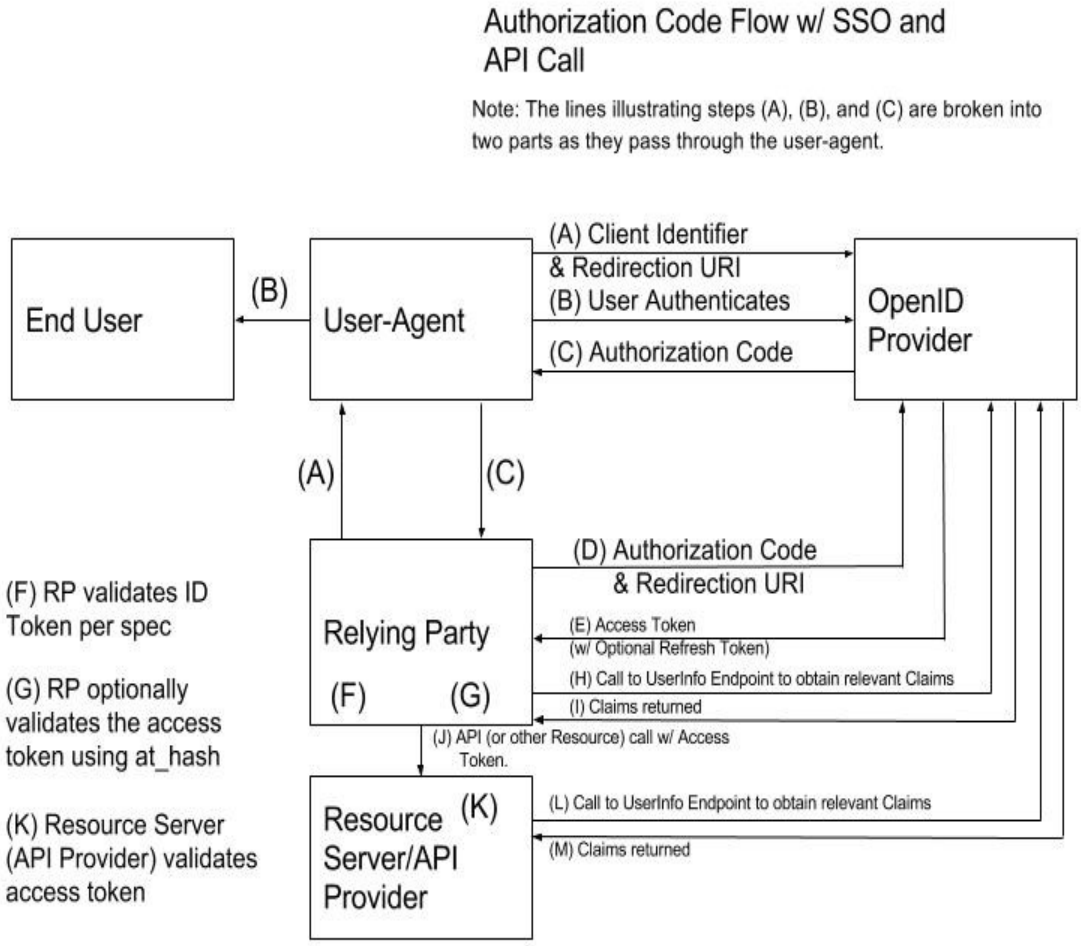

Figure 7- Authorization Flow in OpenID Connect

4. Security Concerns

Considering the flexibility and extensibility of the OpenID standard, there are many areas the security of which must be considered. The basic client profiles to consider for security are: user-agent (browser) based applications, web applications and desktop or native applications. In this section, the potential threats that exist for each client profile is described. Furthermore, the required attributes of a secure OIDC implementation are detailed and emphasized.

4.1 Client Authentication

The authorization server establishes client credentials with web application clients for the purpose of client authentication. The authorization server is encouraged to consider stronger client authentication means than a client password. Web application clients must ensure confidentiality of client passwords and other client credentials.

The authorization server must not issue client passwords or other client credentials to native application or user-agent-based application clients for the purpose of client authentication (Sun, 2012). The authorization server may issue a client password or other credentials for a specific installation of a native application client on a specific device (Hardt, 2012).

When client authentication is not possible, the authorization server should employ other means to validate the client's identity – for example, by requiring the registration of the client redirection URI or enlisting the resource owner to confirm identity. A valid redirection URI is not sufficient to verify the client's identity when asking for resource owner authorization but can be used to prevent delivering credentials to a counterfeit client after obtaining resource owner authorization. The authorization server must consider the security implications of interacting with

unauthenticated clients and take measures to limit the potential exposure of other credentials (e.g., refresh tokens) issued to such clients.

4.2 Client Impersonation

A malicious client may impersonate another client and obtain access to protected resources if the impersonated client fails to, or is unable to, keep its client credentials confidential (Dhamija, 2008).

The authorization server must authenticate the client whenever possible. If the authorization server cannot authenticate the client due to the client's nature, the authorization server must require the registration of any redirection URI used for receiving authorization responses and should utilize other means to protect resource owners from such potentially malicious clients. For example, the authorization server can engage the resource owner to assist in identifying the client and its origin.

The authorization server should enforce explicit resource owner authentication and provide the resource owner with information about the client and the requested authorization scope and lifetime. It is up to the resource owner to review the information in the context of the current client and to authorize or deny the request (Richer, 2015).

The authorization server should not process repeated authorization requests automatically (without active resource owner interaction) without authenticating the client or relying on other measures to ensure that the repeated request comes from the original client and not an impersonator.

4.3 Access Tokens

Access token credentials (as well as any confidential access token attributes) must be kept confidential in transit and storage, and only shared among the authorization server, the resource servers the access token is valid for, and the client to whom the access token is issued (Google,

2018). Access token credentials must only be transmitted using TLS as described in with server authentication as defined by HTTP over TLS standard.

When using the implicit grant type, the access token is transmitted in the URI fragment, which can expose it to unauthorized parties. The authorization server must ensure that access tokens cannot be generated, modified, or guessed to produce valid access tokens by unauthorized parties.

The client should request access tokens with the minimal scope necessary. The authorization server should take the client identity into account when choosing how to honor the requested scope and may issue an access token with less rights than requested. This specification does not provide any methods for the resource server to ensure that an access token presented to it by a given client was issued to that client by the authorization server.

4.4 Refresh Tokens

Authorization servers may issue refresh tokens to web application clients and native application clients. Refresh tokens must be kept confidential in transit and storage and shared only among the authorization server and the client to whom the refresh tokens were issued. The authorization server must maintain the binding between a refresh token and the client to whom it was issued. Refresh tokens must only be transmitted using TLS with server authentication as defined by the HTTP over TLS standard (Yang, 2016).

The authorization server must verify the binding between the refresh token and client identity whenever the client identity can be authenticated. When client authentication is not possible, the authorization server should deploy other means to detect refresh token abuse. For example, the authorization server could employ refresh token rotation in which a new refresh token is issued with every access token refresh response. The previous refresh token is invalidated but retained

by the authorization server. If a refresh token is compromised and subsequently used by both the attacker and the legitimate client, one of them will present an invalidated refresh token, which will inform the authorization server of the breach.

The authorization server must ensure that refresh tokens cannot be generated, modified, or guessed to produce valid refresh tokens by unauthorized parties.

4.5 Authorization Codes

The transmission of authorization codes should be made over a secure channel, and the client should require the use of TLS with its redirection URI if the URI identifies a network resource. Since authorization codes are transmitted via user-agent redirections, they could potentially be disclosed through user-agent history and HTTP referrer headers.

Authorization codes operate as plaintext bearer credentials, used to verify that the resource owner who granted authorization at the authorization server is the same resource owner returning to the client to complete the process. Therefore, if the client relies on the authorization code for its own resource owner authentication, the client redirection endpoint must require the use of TLS.

Authorization codes must be short lived and single-use. If the authorization server observes multiple attempts to exchange an authorization code for an access token, the authorization server should attempt to revoke all access tokens already granted based on the compromised authorization code. If the client can be authenticated, the authorization servers must authenticate the client and ensure that the authorization code was issued to the same client.

4.6 Code Redirection

When requesting authorization using the authorization code grant type, the client can specify a redirection URI via the *redirect_uri* parameter. If an attacker can manipulate the value of the redirection URI, it can cause the authorization server to redirect the resource owner user-agent to a URI under the control of the attacker with the authorization code (Fett, 2016).

An attacker can create an account at a legitimate client and initiate the authorization flow. When the attacker's user-agent is sent to the authorization server to grant access, the attacker grabs the authorization URI provided by the legitimate client and replaces the client's redirection URI with a URI under the control of the attacker. The attacker then tricks the victim into following the manipulated link to authorize access to the legitimate client.

Once at the authorization server, the victim is prompted with a normal, valid request on behalf of a legitimate and trusted client and authorizes the request. The victim is then redirected to an endpoint under the control of the attacker with the authorization code. The attacker completes the authorization flow by sending the authorization code to the client using the original redirection URI provided by the client. The client exchanges the authorization code with an access token and links it to the attacker's client account, which can now gain access to the protected resources authorized by the victim (via the client).

In order to prevent such an attack, the authorization server must ensure that the redirection URI used to obtain the authorization code is identical to the redirection URI provided when exchanging the authorization code for an access token. The authorization server must require public clients and should require confidential clients to register their redirection URIs. If a redirection URI is provided in the request, the authorization server must validate it against the registered value.

4.7 User Credentials

The resource owner password credentials grant type is often used for legacy or migration reasons. It reduces the overall risk of storing usernames and passwords by the client but does not eliminate the need to expose highly privileged credentials to the client. This grant type carries a higher risk than other grant types because it maintains the password anti-pattern this protocol seeks to avoid.

The client could abuse the password, or the password could unintentionally be disclosed to an attacker (e.g., via log files or other records kept by the client). Additionally, because the resource owner does not have control over the authorization process (the resource owner's involvement ends when it hands over its credentials to the client), the client can obtain access tokens with a broader scope than desired by the resource owner. The authorization server should consider the scope and lifetime of access tokens issued via this grant type.

The authorization server and client should minimize use of this grant type and utilize other grant types whenever possible.

4.8 Endpoint Legitimacy

Access tokens, refresh tokens, resource owner passwords, and client credentials must not be transmitted in the clear. Authorization codes should not be transmitted in the clear.

The "state" and "scope" parameters should not include sensitive client or resource owner information in plain text, as they can be transmitted over insecure channels or stored insecurely.

4.9 Request Confidentiality

In order to prevent man-in-the-middle attacks, the authorization server must require the use of TLS with server authentication as defined by the HTTP over TLS standard for any request sent to the

authorization and token endpoints. The client must validate the authorization server's TLS certificate as defined by Public Key Infrastructure standard and in accordance with its requirements for server identity authentication.

4.10 Brute-force Credentials-Guessing

The authorization server must prevent attackers from guessing access tokens, authorization codes, refresh tokens, resource owner passwords, and client credentials. The probability of an attacker guessing generated tokens (and other credentials not intended for handling by end-users) must be less than or equal to $2^{(-128)}$ and should be less than or equal to $2^{(-160)}$.

The authorization server must utilize other means to protect credentials intended for end-user usage.

4.11 Phishing Attacks

Wide deployment of this and similar protocols may cause end-users to become inured to the practice of being redirected to websites where they are asked to enter their passwords. If end-users are not careful to verify the authenticity of these websites before entering their credentials, it will be possible for attackers to exploit this practice to steal resource owners' passwords.

Service providers should attempt to educate end-users about the risks phishing attacks pose and should provide mechanisms that make it easy for end-users to confirm the authenticity of their sites. Client developers should consider the security implications of how they interact with the user-agent (e.g., external, embedded), and the ability of the end-user to verify the authenticity of the authorization server.

To reduce the risk of phishing attacks, the authorization servers must require the use of TLS on every endpoint used for end-user interaction.

4.12 Cross-Site Request Forgery

Cross-site request forgery (CSRF) is an exploit in which an attacker causes the user-agent of a victim end-user to follow a malicious URI (e.g., provided to the user-agent as a misleading link, image, or redirection) to a trusting server (usually established via the presence of a valid session cookie).

A CSRF attack against the client's redirection URI allows an attacker to inject its own authorization code or access token, which can result in the client using an access token associated with the attacker's protected resources rather than the victim's (e.g., save the victim's bank account information to a protected resource controlled by the attacker).

The client must implement CSRF protection for its redirection URI. This is typically accomplished by requiring any request sent to the redirection URI endpoint to include a value that binds the request to the user-agent's authenticated state (e.g., a hash of the session cookie used to authenticate the user-agent). The client should utilize the "state" request parameter to deliver this value to the authorization server when making an authorization request.

Once authorization has been obtained from the end-user, the authorization server redirects the end-user's user-agent back to the client with the required binding value contained in the "state"

parameter. The binding value enables the client to verify the validity of the request by matching the binding value to the user-agent's authenticated state. The binding value used for CSRF protection must contain a non-guessable value, and the user-agent's authenticated state (e.g.,

session cookie, HTML5 local storage) must be kept in a location accessible only to the client and the user-agent (i.e., protected by same-origin policy).

A CSRF attack against the authorization server's authorization endpoint can result in an attacker obtaining end-user authorization for a malicious client without involving or alerting the end-user.

The authorization server must implement CSRF protection for its authorization endpoint and ensure that a malicious client cannot obtain authorization without the awareness and explicit consent of the resource owner.

4.13 Clickjacking

In a clickjacking attack, an attacker registers a legitimate client and then constructs a malicious site in which it loads the authorization server's authorization endpoint web page in a transparent iframe overlaid on top of a set of dummy buttons, which are carefully constructed to be placed directly under important buttons on the authorization page (Kiani, 2011). When an end-user clicks a misleading visible button, the end-user is actually clicking an invisible button on the authorization page (such as an "Authorize" button). This allows an attacker to trick a resource owner into granting its client access without the end-user's knowledge.

To prevent this form of attack, native applications should use external browsers instead of embedding browsers within the application when requesting end-user authorization. For most newer browsers, avoidance of iframes can be enforced by the authorization server using the (non-standard) "x-frame-options" header. This header can have two values, "deny" and "sameorigin", which will block any framing, or framing by sites with a different origin, respectively. For older browsers, JavaScript frame-busting techniques can be used but may not be effective in all browsers.

4.14 Code Injection and Input Validation

A code injection attack occurs when an input or otherwise external variable is used by an application unsanitized and causes modification to the application logic. This may allow an attacker to gain access to the application device or its data, cause denial of service, or introduce a wide range of malicious side-effects.

The authorization server and client must sanitize (and validate when possible) any value received -- in particular, the value of the "state" and "redirect_uri" parameters.

4.15 Open redirectors

The authorization server, authorization endpoint, and client redirection endpoint can be improperly configured and operate as open redirectors. An open redirector is an endpoint using a parameter to automatically redirect a user-agent to the location specified by the parameter value without any validation (Hardt, 2012).

Open redirectors can be used in phishing attacks, or by an attacker to get end-users to visit malicious sites by using the URI authority component of a familiar and trusted destination. In addition, if the authorization server allows the client to register only part of the redirection URI, an attacker can use an open redirector operated by the client to construct a redirection URI that will pass the authorization server validation but will send the authorization code or access token to an endpoint under the control of the attacker.

4.16 Bearer Token Security

The possession of a bearer token can lead to access of any associated resources (without demonstrating possession of a cryptographic key). To prevent misuse, bearer tokens need to be

protected from disclosure in storage and in transport. Client implementations must ensure that bearer tokens are not leaked to unintended parties, as they will be able to use them to gain access to protected resources (Miculan, 2011). This is the primary security consideration when using bearer tokens and underlies all the more specific recommendations that follow.

The client must validate the TLS certificate chain when making requests to protected resources. Failing to do so may enable DNS hijacking attacks to steal the token and gain unintended access.

Clients must always use the TLS protocol (https) or equivalent transport security when making requests with bearer tokens. Failing to do so exposes the token to numerous attacks that could give attackers unintended access.

Implementations must not store bearer tokens within cookies that can be sent in the clear (which is the default transmission mode for cookies). Implementations that do store bearer tokens in cookies must take precautions against cross-site request forgery.

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 (Hardt, 2012). Using short-lived bearer tokens can reduce the impact of them being leaked.

Token servers should issue bearer tokens that contain an audience restriction, scoping their use to the intended relying party or set of relying parties.

Bearer tokens should not be passed in page URLs (for example, as query string parameters). Instead, bearer tokens should be passed in HTTP message headers or message bodies for which confidentiality measures are taken. Browsers, web servers, and other software may not adequately

secure URLs in the browser history, web server logs, and other data structures. If bearer tokens are passed in page URLs, attackers might be able to steal them from the history data, logs, or other unsecured locations (Lodderstedt, 2013).

5. Privacy Concerns

In addition to the above security considerations, there are privacy consideration to be made when using OpenID Connect. While related to security, privacy goes beyond simply the unauthorized access and could potentially lead to longer lasting damages. The OIDC standard addresses security first but does not ignore privacy and the consequences of its breach. Consequently, the standard clearly specifies the major points where security is a concern and provides practical and concrete recommendations to ensure privacy is not compromised. In this section, the user information at risk is identified and proper remedies to ensure its privacy are recommended.

5.1 User Identifiable Information

User identifiable information is any information about an individual that can be used directly, or in connection with other data, to identify, contact or locate that person. Such information can include medical, educational, financial, legal and employment records (Corella, 2013).

The *UserInfo* Response typically contains Personally Identifiable Information (PII). As such, end-user consent for the release of the information for the specified purpose should be obtained at or prior to the authorization time in accordance with relevant regulations. The purpose of use is typically registered in association with the *redirect_uris*.

Only necessary *UserInfo* data should be stored at the Client and the Client should associate the received data with the purpose of use statement.

5.2 Data Access Monitoring

The Resource Server should make end-users' *UserInfo* access logs available to them so that they can monitor who accessed their data. This is necessary to investigate any unauthorized access and to prevent similar breaches in the future.

5.3 Correlation

To protect the end-user from a possible correlation among Clients, the use of a Pairwise Pseudonymous Identifier as the subject. This requires issuing a unique subject identifier value for each Client this avoiding the ability to correlate a user among all Clients (Corella, 2013).

5.4 Offline Access

Offline access enables access to Claims when the user is not present, posing greater privacy risk than the Claims transfer when the user is present. Therefore, it is prudent to obtain explicit consent for offline access to resources (Schwartz, 2014). This specification mandates the use of the prompt parameter to obtain consent unless it is already known that the request complies with the conditions for processing the request in each jurisdiction.

When an Access Token is returned via the User Agent using the Implicit Flow or Hybrid Flow, there is a greater risk of it being exposed to an attacker, who could later use it to access the UserInfo endpoint. If the Access Token does not enable offline access and the server can differentiate whether the Client request has been made offline or online, the risk will be substantially reduced. Therefore, this specification mandates ignoring the offline access request when the Access Token is transmitted through the User Agent.

Note that differentiating between online and offline access from the server can be difficult especially for native clients. The server may well have to rely on heuristics. Also, the risk of exposure for the Access Token delivered through the User Agent for the Response Types of code token and token is the same (Kiani, 2011). Thus, the implementations should be prepared to detect whether the Access Token was issued through the User Agent or directly from the Token Endpoint and deny offline access if the token was issued through the User Agent.

Note that although these provisions require an explicit consent dialogue through the prompt parameter, the mere fact that the user pressed an "accept" button etc., might not constitute a valid consent. Developers should be aware that for the act of consent to be valid, typically, the impact of the terms have to be understood by the end-user, the consent must be freely given and not forced (i.e., other options have to be available), and the terms must fair and equitable. In general, it is advisable for the service to follow the required privacy principles in each jurisdiction and rely on other conditions for processing the request than simply explicit consent, as online self-service "explicit consent" often does not form a valid consent in some jurisdictions.

6. Successful OIDC Implementation

Every standard or protocol is only as successful as the implementations that adopt it. In that regard alone, OIDC is one of the most successful protocols to have been created. Even so, success does not come easily or swiftly. The OIDC standard, in addition to its predecessor protocols, has gone through several iterations to reach its current state. In this section, the details of what makes a successful OIDC implementation are explored and emphasized. Conversely, the aspects of unsuccessful, or failed, implementations are highlighted in contrast.

As a four-year old standard, OpenID Connect is considered in its infancy. However, due to its wide adoption by industry giants and smaller organizations alike, OpenID Connect has thrived in such a short period (Mladenov, 2017). OpenID Connect is now the leading standard for single sign-on and identity management on the Internet. OIDC is successful due to the simple JSON-based identity tokens (JWT), delivered through the OAuth 2.0 protocol to serve web, browser-based and native or mobile applications.

What makes OIDC successful is both its simplicity and improved security over earlier OpenID versions and other standards. However, security of any standard, including OIDC, is only as good as the implementation of that standard. With that in mind, a close scrutiny of OIDC implementations has shown that while the standard offers theoretical security, a poorly implemented OIDC standard only invites attackers.

Many of the issues uncovered in large implementations of OIDC reside with the Relying Parties (RPs). The RP (also known as Client) often customize the OIDC standard to maximize efficiency and user experience at the cost of security. While it may seem expedient to take a freely available standard such as OIDC and retrofit it to an organization's needs, the fundamental purpose of the standard is to ensure security. User experience and efficiency will often stand in contrast to the security recommendation made by OIDC.

Relying Parties may also deviate from the standard in different ways during implementation. For example, the *state* parameter is often ignored as a value to be used and validated. The OIDC standard insists that Cross-Site Request Forgery (CSRF) can successfully be mitigated if the *state* is used and validated as part of the user authentication request. This simple, yet important parameter may be unutilized due to lack of understanding of the standard or neglect. The consequences, however, may be dire to the entire system (Li, 2013).

Similarly, OpenID Providers (OP) are not immune to poor implementations and my also deviate from the standard in their implementation. For example, the use of the *access token*, instead of *id token*, as proof of identity contravenes the OIDC standard. In fact, stolen access tokens have been used by malicious attackers to impersonate a user. OpenID Providers must not issue or validate an *access token* as evidence of identity.

Even when both the RP and OP properly implement their relevant areas of the standard, potential issues may arise from misuse of other areas of the standard. For example, OIDC insists that any communication between the end-user, the RP and the OP must use an encrypted channel and that tokens issued may not be sent in cleartext over the wire. If any part of the communication between the three parties occurs over an unencrypted network, both the *access token* and the *id token* may be intercepted. This is known as to token leakage. And since both tokens encode user attributes, this can easily lead to loss of privacy or worse.

Despite the meteoric rise in popularity of OIDC and its wide adoption, there exists many implementations, some from large organizations, with critical security flaws due to deviations from the standard. It is, therefore, not unreasonable to assume that strict adherence to the standard may not be feasible (Sun, 2012). Many developers tasked with implementing the standard as RP or OP may not have the deep security expertise required to fully understand and stay true to the letter and spirit of the OIDC standard.

7. Improving OpenID Connect

While there is no perfect standard in existence today, the OIDC protocol solves many issues discovered over the years in IAM. As the requirements and usages of OIDC evolve, this section explores improvements to the standard and practical applications that further enhance its security.

7.1 OpenID Certification

In addition to maintaining and improving the OpenID standard, the OpenID Foundation enables implementations of OpenID Connect to be certified to specific conformance profiles to promote interoperability among implementations. Any online deployment of a product or service that implements a conformance profile of the OpenID Connect protocol is eligible for certification. The foundation's certification process utilizes self-certification and a conformance test suite developed by the foundation. Entities looking to use or rely on a deployment of a product or service that implements a specific conformance profile of the OpenID Connect protocol often need some assurance that the deployment actually conforms to the profile (Foundation, 2016). A certification can help provide that assurance.

Since the OpenID certification process is conducted through self-certification, organizations submitting a certification of conformance are legally bound. This legal binding is due to the organization declaring both to the OpenID Foundation and to the general public the accuracy of the matters it declared in the Certification. Inherently, the trustworthiness of a self-certification is partially a function of the trustworthiness of the entity that is certifying itself, discounted, perhaps, by the self-interest involved. When an entity makes a self-certification, it puts its reputation on the line. In addition, it undertakes potential liability for damages suffered by those who rely on its self-certification in the event that the self-certification is not accurate. And it also exposes itself to

potential liability under government regulatory statutes and regulations, such as laws that prohibit unfair and deceptive business practices.

To encourage participation, the process of certification is both relatively cheap and self-driven by design. Unlike third-party certification, those implementing a deployment of a product or service conduct reviews to determine whether the deployment complies with specific conformance profile, and upon successful completion of such review, issue declaration of compliance. Self-certification is easier, quicker, and significantly cheaper than third-party certification. In third-party certifications, someone other than the entity deploying the product or service (usually a specially accredited and trustworthy auditor or assessor authorized to conduct such a review) reviews, tests, assesses, and verifies that the entity's deployment of the product or service conforms to a specific conformance profile, and then issues a statement to the effect that it has conducted the specified assessment, and certifies that the entity's deployment of the product or service conforms to the specified conformance profile.

In the case of self-certification, the trustworthiness of the certification is a function of the trustworthiness of the entity that is assessing itself. In the case of third-party certification, the trustworthiness of the certification is a function of the trustworthiness of the assessing entities/certifying entity as well as the trustworthiness of the entity requesting the assessment.

The OpenID Foundation allows certified implementations to use the "OpenID Certified" certification mark and publishes the date of certification on its website.

Figure 8- OpenID Certification Logo

7.2 OpenID Connect and Two-Factor Authentication (2FA)

The Universal Second Factor (U2F) protocol from the FIDO Alliance is a successful new authentication protocol that, when used with the emerging OpenID Connect standard, can strengthen both protocols. With the pair of protocols, even more authentication challenges can be solved that either can solve alone.

U2F uses a hardware cryptographic device to allow users to authenticate to sites. This is accomplished by using public key cryptography, but without the deficiencies inherent in legacy PKI systems. For every service or site to which the user connects, a new key pair is generated offering authentication that is secure and privacy-preserving.

In practice, the U2F protocol doesn't actually identify any particular user on its own. It merely proves someone has the device with control over a registered key (Richer, 2015). The user's identity is intentionally missing from the U2F process, and it must always be bound to a user account for it to represent a person. This is where OpenID Connect as an identity federation protocol is enhanced by U2F. Since OIDC doesn't actually authenticate the user but rather propagates the user authentication across the network to an IdP where the actual authentication happens through a certificate, a username and password pair, or a hardware token. U2F is ideally suited to serve as that authentication through a secure physical mechanism.

Both OpenID Connect and U2F address authentication in different ways and both leave some areas intentionally unaddressed to be filled by other technologies and components. Using both protocols, a wider array of challenges can be addressed. For example, an OpenID Connect IdP could use a U2F device as part of its primary authentication mechanism allowing a serial use of the two protocols and allows the user to strongly protect their primary online identity.

Similarly, the OIDC and U2F protocols can be used in parallel. In this setup, OIDC acts as the primary login of a user to a resource provider, while the U2F device is registered on top of the federated login for enhanced validation resource provider can itself perform.

The simplest way of combining the two protocols is to make U2F a part of the login process at the IdP. Since the OIDC protocol deliberately delegates the primary authentication mechanism to the IdP and leaves it unspecified in the protocol, it can leverage a large number of different technologies. A user's account at an IdP can be backed by several mechanisms including a U2F device (Richer, 2015). An IdP can even dynamically choose among several mechanisms and to insist on the presence of a U2F device for authentication if needed, perhaps at the request of the user by configuration or at the RP's request.

This method is largely transparent to the RP, and its presence can easily be ignored by the RP, if necessary. In addition, OpenID Connect, through the use of the "authentication context reference" and "authentication method reference" mechanisms, already provides RPs a way to request multifactor authentication. This is already baked into the protocol and allows it to be told about the primary authentication was used in the ID Token response. Furthermore, since OpenID Connect (and its underlying OAuth2 protocol) recommends applications use the system browser, instead of embedded web views, for example, users can take advantage of existing sessions and preferences, and, more importantly, take advantage of extended functionality such as a U2F agents. This makes the user's interaction with the IdP familiar and trusted, even in the context of a U2F session.

A more integrated method of combining OIDC and U2F is to use them in parallel at a given RP. In this method, when a user creates an account at an RP, the RP requests both a federated identity and a U2F device registration. Both credentials can be presented in parallel

within the same session, strongly binding the two of them to the same user at the RP. This allows the RP the flexibility to choose whether either or both mechanisms are needed to perform a successful login. While a federated login is sufficient to authenticate the user for the most actions available on at the RP, when an elevated action is requested by the user, the presence of a trusted (and registered) U2F device is confirmed through the use of the U2F protocol. This would thwart most attackers who must gain access to both the remote IdP account as well as the U2F device in order to impersonate the user at that RP

There is a cost to the RPs in terms higher complexity to implement both the OIDC and U2F protocols, in addition to potentially poor user experience. However, this method is might be particularly suited to mobile applications. The use of a mobile-friendly identity federation protocol such as OpenID Connect can both identify the user to a device and bind their attributes to it (Richer, 2015). The OpenID Connect tokens can be shared across applications on the device to uniquely identify the user while the U2F protocol can be used locally on the mobile device to provide a stronger security for certain elevated actions. This, again, prevents attackers from comprising the user's authentication since they must now possess the U2F device in addition to the mobile device.

8. Conclusion

Identity and Access Management (IAM) is a critical part of every organization in existence today. Successful and robust implementations of IAM are made even more important by the complexity of the distributed nature of the modern enterprise. The balance between security and the convenience provided by a streamlined authentication mechanism has never been more important. This delicate balance recognizes the advantages to users provided by single sign-on experience with the ability to prevent unscrupulous actors from inflicting damage to the enterprise. To that end, enterprises face the daunting task of making the right strategic decisions about which IAM solutions to adopt based on proven security and which to discard due to inherent flaws or known vulnerabilities.

The improvements made in IAM technologies are barely keeping pace with the vast transformation in business landscape. The current rapid migration from local and desktop-based computing to a global and networked workforce requires a shift in the traditional IAM paradigms. On-demand access to critical information resources to authorized users across vast networks is a challenge with no single solution. OpenID Connect, among other technologies, is an important piece of this solution (Fremantle, 2014). By encouraging the centralization of identity residence to Identity Providers, OpenID Connect enhances the security of IAM and provides a robust single sign-on experience to end-users.

OpenID Connect as a standard provides comprehensive and concrete recommendations to several fundamental enterprise IAM challenges in modern business computing. Among the difficulties facing end-users in an increasingly connected world is the increasing pressure to remember an ever-increasing number of credentials for disparate applications that use different authentications

mechanisms. With the explosive growth of applications deployed to the cloud, not only does this diminish the usability of such applications but increases the likelihood that credentials are compromised through Phishing or other social engineering exploits. OpenID Connect provides a streamlined and secure mechanism for single sign-on that allows enterprises to delegate IAM to a trusted and centralized IdP with sufficient checks on the integrity of the IdP and the security tokens issued to maintain the SSO experience.

Another obstacle to streamlined IAM is the proliferation of heterogenous devices available on enterprise networks. It is no longer sufficient to assume that all users, employees and third-party partners would use the same devices to access resources. Furthermore, it is now becoming the norm that many users bring their own devices (BYOD) to access critical enterprise resources within and outside of corporate networks. This presents a challenge on many levels but particularly in the IAM realm, it poses the risk of excluding users with certain devices from access to resources necessary to their productivity. OpenID Connect goes a long way in reducing the impact of BYOD on IAM since it supports applications running on desktops, mobile devices, and the cloud. This technology and device agnostic feature of OpenID Connect has made it the widely adopted standard it is today (Schwartz, 2014).

Since OpenID Connect is an extension and enhancement of OAuth2, it was designed to tackle and standardize another challenge to IAM, namely distributed access control. OpenID Connect, through the use of secure tokens, provides both an authentication and authorization mechanism. This dual feature allows enterprises to combine identity verification with a simple and robust mechanism for resource access control. The claims available in an OpenID Connect secure token allow a finely granular mechanism to grant and revoke access to resources to individual users without affecting access to other users in the enterprise.

In this thesis, the OIDC standard was explored from both a technical and practical point of view. The aim of this exploration was to show why OIDC is considered the future of federated Identity and Access Management. Considering the discussed strength of the protocol in terms of security and privacy, and its wide adoption by many, if not most, large technology firms, it is not unreasonable to conclude that OIDC is now the protocol of choice for IAM. Furthermore, OIDC has built-in flexibility, as shown in the many areas where implementation details are left unspecified, to allow for future improvements. The extensibility features of OIDC enhance its appeal when the cost of implementation is considered over a long period of use. This simultaneous ability to address the IAM needs of organizations today and ensure

References

Barnes, R., Mozilla, (2014). *Use Cases and Requirements for JSON Object Signing and Encryption (JOSE).* Retrieved from https://tools.ietf.org/html/rfc7165

Corella, F., Lewison, K., (2013). *Privacy Postures of Authentication Technologies*. Retrieved from https://pomcor.com/techreports/PrivacyPostures.pdf

Craig, L., (2016). *Cloud Federation Management and Beyond: Requirements, Relevant Standards, and Gaps*. Retrieved from https://www.computer.org/csdl/mags/cd/2016/01/mcd2016010042-abs.html

Dhamija, R., Dusseault, L. (2008). *The seven flaws of identity management: usability and security challenges*. IEEE Secur. Priv. 6(2), 24–29

Fett, D., Kusters, R., Schwenk, J., Schmitz, G., (2015). *The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines*. Retrieved from https://sec.uni-stuttgart.de/_media/publications/FettKuestersSchmitz-CSF-2017.pdf

Fett, D., Kusters, R., Schwenk, J., Schmitz, G., (2016). *A Comprehensive Formal Security Analysis of OAuth 2.0*. Retrieved from https://arxiv.org/abs/1601.01229

Fremantle, P., Aziz, B., Kopecky, J., (2014). *Federated Identity and Access Management for The Internet of Things.* Retrieved from https://ieeexplore.ieee.org/abstract/document/7058903

Google Identity Platform. (2018). *OpenID Connect Reference Implementation*. Retrieved from https://developers.google.com/identity/protocols/OpenIDConnect

Halpin, H., (2016). *The next step in the evolving identity eco-system?* Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.448.8891&rep=rep1&type=pdf

Hardt, D., (2012). *The OAuth 2.0 authorization framework*. Retrieved from https://tools.ietf.org/html/rfc6749

Internet Society. (2016). Public Policy Briefing. *Identity on the Internet*. Retrieved from https://www.internetsociety.org/wp-content/uploads/2017/09/ISOC-PolicyBrief-Identity-20160603-en-nb.pdf

Kiani, K. (2011). *Four Attacks on OAuth – How to Secure Your OAuth Implementation.* Retrieved from sans.org: https://software-security.sans.org/blog/2011/03/07/oauth-authorization-attacks-secure-implementation

Li, Wanpeng, Mitchell, C., (2013). *Analysing the Security of Google's Implementation of OpenID Connect*. Retrieved from https://link.springer.com/chapter/10.1007/978-3-319-40667-1_18.

Li, Wanpeng, Mitchell, C., (2013). *Security issues in OAuth 2.0 SSO implementations*. Retrieved from http://www.chrismitchell.net/Papers/siio2s.pdf.

Lodderstedt, T., McGloin, M., Hunt, P. (2013). *OAuth 2.0 Threat Model and Security Considerations.* Retrieved from https://tools.ietf.org/html/rfc6819

Miculan, M., Urban, C., (2011). *Formal Analysis of Facebook Connect Single Sign-On Authentication Protocol.* Retrieved from https://people.inf.ethz.ch/caurban/Publications_files/fbconnect.pdf

Mladenov, V., Mainka, C., Schwenk, J., (2015). *On the security of modern Single Sign-On Protocols: Second-Order Vulnerabilities in OpenID Connect*. Retrieved from https://arxiv.org/abs/1508.04324

Mladenov, V., Mainka, C., Schwenk, J., Wich, T. (2016). *Single Sign-On Security – An Evaluation of OpenID Connect*. Retrieved from https://www.nds.rub.de/media/ei/veroeffentlichungen/2017/01/30/oidc-security.pdf

Mladenov, V., (2017). *On the Security of Single Sign-On.* Retrieved from https://d-nb.info/1142001725/34

Naik, N, Jenkins, P., (2016). *A Secure Mobile Cloud Identity: Criteria for Effective Identity and Access Management Standards*. Retrieved from https://ieeexplore.ieee.org/abstract/document/7474415/

OpenID Foundation. (2014). Final Specification. *OpenID Connect Core*. Retrieved from https://openid.net/specs/openid-connect-core-1_0.html.

OpenID Foundation. (2014). Final Specification. *OpenID Connect Discovery*. Retrieved from https://openid.net/specs/openid-connect-discovery-1_0.html.

OpenID Foundation. (2014). Final Specification. *OpenID Connect Federation*. Retrieved from https://openid.net/specs/openid-connect-federation-1_0.html.

Organization for Economic Co-Operation and Development. (2011). *Enabling Innovation and Trust in the Internet Economy.* Retrieved from https://www.oecd.org/sti/ieconomy/49338380.pdf

Politze, M., Decker, B., (2017). *Extending the OAuth2 Workflow to Audit Data Usage for Users and Service Providers in a Cooperative Scenario.* Retrieved from https://www.researchgate.net/publication/317578977

Richer, J., (2015). *Universal Second Factor and OpenID Connect*. Retrieved from https://www.yubico.com/wp-content/uploads/2015/08/Yubico-U2F-and-OIDC-Final.pdf

Siriwardena, P., (2014). *Advanced API Security: Securing APIs with OAuth2.0, OpenID Connect, JWS, and JW.* Retrieved from https://www.kobo.com/us/en/ebook/advanced-api-security

Sun, S.T., Beznosov, K. (2012) *The devil is in the (implementation) details: An empirical analysis of OAuth SSO systems*. In Proc. CCS '12, *ACM*

Wang, R., Chen, S., Wang, X., (2012). *Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed single-sign-on web services*. In: Proc. IEEE Symp. on Security and Privacy, *IEEE*

Wilton, R., (2014). *Have You Chosen an Identity Provider Lately?* Retrieved from https://www.internetsociety.org/resources/doc/2014/have-you-chosen-an-identity-provider-lately/

Yang, R., Lau, W., Liu, T. (2016). *Signing into One Billion Mobile Accounts Effortlessly with OAuth2.0*. Retrieved from https://www.blackhat.com/docs/eu-16/materials/eu-16-Yang-Signing-Into-Billion-Mobile-Apps-Effortlessly-With-OAuth20-wp.pdf

Appendix A – Glossary

2FA:

Two-factor authentication.

Attack:

An intentional act by which an entity attempts to evade security services and violate the security policy of a system. That is, an actual assault on system security that derives from an intelligent threat.

Authentication:

The process of verifying a claim that a system entity or system resource has a certain attribute value.

Authorization:

An approval that is granted to a system entity to access a system resource.

Authorization Server (AS):

The authorization server issues access tokens to the client after successfully authenticating a user and gives authorization. Authentication Request: Request towards an identity provider asking to verify the identity of the user. Authorization Code Flow: OAuth2 or OpenID Connect flow in which an authorization code is returned from the authorization server, where all tokens are returned from the Token Endpoint. Authorization Request: Request towards an Authorization server asking for the rights of the authenticated user.

Bearer token:

Tokens without a secret or other verification mechanism.

Claim:

Piece of information asserted about an Entity.

Client:

A device owned by a user, such as a desktop PC, a tablet or smartphone.

Credential:

Data presented as evidence of being a certain identity. A pair of a unique identifier and a matching secret

ID Token:

JSON Web Token (JWT) that contains claims about the authentication event.

Identifier:

Value that uniquely characterizes an entity in a specific context.

Federation:

An association containing a number of service providers and identity providers.

Hybrid Flow:

OpenID Connect flow in which an authorization token is returned from the authorization server and the ID token from the Token Endpoint.

Identity:

Set of attributes related to an Entity.

IdP:

An Identity Provider (IdP) is responsible for the authentication process and handles the storing of user information as attributes in an ID token. When a user authenticates, the IdP creates an object that contains the information of the user, which can be used when a service provider request user attributes.

Implicit Flow:

OAuth2 or OpenID Connect flow in which all tokens are returned from the Authorization Endpoint and neither the Token Endpoint nor an authorization code are used.

Issuer:

Entity that issues a set of Claims.

JWT:

JSON web token.

OIDC:

Abbreviation of OpenID Connect

Personally Identifiable Information (PII):

Information that can be used to identify the natural person to whom such information relates or is or might be directly or indirectly linked to a natural person to whom such information relates.

Protocol:

A complete solution, for example "the OAuth2 protocol".

Relying Party (RP):

Client application requiring user authentication and Claims from an SSO provider.

Request Object:

JWT that contains a set of request parameters as its Claims.

Request URI:

URL that references a resource containing a Request Object.

Service:

An application or a resource to which a user wants to gain access.

SSO:

A solution that allows users to log in on a single login page. Afterwards, access is granted to multiple services. SSO requires an IdP and in some cases also handles authorization using an AS.

Token:

A unique identifier issued by the server and used by the client to associate authenticated requests with the resource owner whose authorization is requested or has been obtained by the client. Tokens have a matching shared-secret that is used by the client to establish its ownership of the token, and its authority to represent the resource owner.

Token Info Endpoint:

Protected Resource that, when presented with a by-reference token returns a by-value token and when presented with a by value token returns a by reference token.

User Info Endpoint:

Protected Resource that, when presented with an access token by the Client, returns authorized information about the user. Communication towards the User-Info Endpoint must be protected using https.

Validation:

Process intended to establish the soundness or correctness of a construct.