Security Investigation and Analysis of OpenID: Problems and Enhancements

Waleed A. Alrodhan and Alya I. Alqarni

College of Computer and Information Sciences
Al-Imam Muhammad ibn Saud University

Summary
OpenID is a widely used identity management system (IdMS) by which identity providers (IdPs) provide their users with 'open' identities that can be used to log in to particular relaying parties (RPs). OpenID implements a single sign-on (SSO) solution that reduces the number of authentication credentials that are required. An SSO permits users to authenticate themselves to many SPs by using one set of authentication credentials. OpenID is faster and easier than the traditional method, which requires the user to manage a large number of digital identities, since each SP only recognises the identity it has issued. This increases the security risk of identity theft and, at the same time, forms an obstacle with regard to user convenience. The aim of this paper is to analyse the security of OpenID by identifying its weaknesses and vulnerabilities using OWASP tools, and to enhance OpenID current protocols by proposing a novel high-level integration model of OpenID and Higgins (an Information Card based IdMS).

Key words:
OpenID, Higgins, Security, Identity, Privacy..

1. Introduction

This paper aims to analyse and enhance the security of OpenID, one of the most commonly used identity management systems on the web, by proposing enhancements to OpenID's by analysing and investigating its security framework. The analysis will be conducted using specific tools (OWASP ZAP1 and VCG2), and will cover most of the OpenID framework. We will identify the weaknesses in the OpenID's protocol and propose and evaluate ways to support the exposed weaknesses.

OpenID is an identity management system (or IDMS). An IdMS permits confidential sources to implement identity management tasks using an effective structure [1]. OpenID supports single sign-on (or SSO) authentication procedure. An SSO is an identity management protocol which permits users to use only one account which is secured and trusted for admission to different resources. A SSO permits networked services to realise authentication objectives using overwhelming just-in-time identity information from confident sources that exist in other systems or structural areas, at the moment when the user uses the method [2]. SSO aims at boosting-up the user-convenience and mitigating the risk of users forgetting their security credentials.

1.1 OpenID system

OpenID is one of the IDMS's that offer users with a universal identity, permitting them to sign in to several relying parties (or RPs). OpenID supports SSO which prevents the necessity to have individual signings and passwords for to each website which supports OpenID [3]. The method of confirming with OpenID on the SSO system has several security issues. One main threat inherited in the OpenID authentication procedure is phishing attacks [3]. Furthermore, weaknesses in the design of web SSO permit attackers to satirise users [2]. OpenID is in the possession of, and managed by, the OpenID Foundation3, a non-profit, global calibration organization of individuals and corporations dedicated to allowing, helping, and guarding OpenID knowledge. The foundation is held by numerous well-known administrations containing Google, IBM, Microsoft, Yahoo!, PayPal and VeriSign. Conferring with the OpenID website indicates that there are presently more than one billion OpenID-allowed identities and about nine million OpenID-allowed SPs on the Internet. On the other hand, almost nothing of these SPs provides admission to any data of any existent value, even though this may be modified in the future [1].

OpenID is regionalised. The regionalisation permits the user to select an OpenID provider appropriate to personal favourites and enables the user to develop an identity provider. In addition, the SSO advantage grants users' admission to various websites with the use of one login. Usability is an imperative feature of a user-collaborating protocol or system. In this deference, the usability of OpenID is imperative since the user cooperates directly with the protocol through the OpenID progression. There are two recognised attacks on the OpenID protocol: the

[1] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
[2] https://github.com/nccgroup/VCG/tree/master/VisualCodeGrepper
[3] https://openid.net/

Manuscript received October 5, 2017
Manuscript revised October 20, 2017

cross-site request forgery attack (CSRF) and the open forward attack. OpenID Connect is defenceless against CSRF attacks that are performed in several stages. Once attackers log on to OpenID, they be able to read secretive messages, post remarks, modify payment information, and so on. After that, they will be able to disconnect from the provider, and no one will know about whatever has occurred. The open forward attack is a method to forward the value related with a claims value to a user; for example, a URL without any authentication. This weakness is typically exploited in phishing attacks to entice users to access malicious sites without understanding what is happening [4].

OpenID has quickly emerged on the Internet. There were around one billion OpenIDs and 50 thousands enabled-SPs in 2010. The constraints of OpenID have been defined as worries from amount of security restrictions. Of specific concern is its dependence on the concept of universal indenters that increases important privacy concerns [1].

The reminder of the paper is organised as follows. Section 2 provides a background overview. In Section 3 we discuss our security analysis of OpenID Connect using two security analysis tools; namely, OWASP's Zed Attack Proxy (ZAP) and the Visual Code Gripper (VGC). In Section 4, we discuss two security enhancements of OpendID Connect; one has proposed by the OpenID developers and the other by the authors of this paper. An analysis of both enhancements will be given in the same section. Finally, Section 5 concludes the paper and discusses future work.

2. Background overview

This section provides an overview of the OpenID 2.0 and OAuth 2.0[1] protocols by defining their roles, endpoints, protocol flow. Also, it describes OpenID Connect (the latest OpenID version) and Higgins-Identity[2]. In addition, this section discusses the tools we used to analyse the OpenID connect protocol. Finally, an overview of previous related work will be provided.

2.1 OpenID 2.0

OpenID 2.0 is a regionalised web-based SSO protocol. Its main objective is to sign in an end user, epitomised by an identifier, at RPs by an identity provider (or IdP) named the OpenID provider (or OP). To possess an OpenID, an end user must create an index with an OP (e.g., Google or Yahoo).

OpenID 2.0 framework contains four main parties:

1. End User: A user applying a user agent (e.g. a web browser) to log in to an RP.

2. Relaying Party (RP): A service provider (e.g. web server).

3. OpenID Provider (OP): An identity provider.

4. Identifier Host: A host, assumed an OpenID Identifier, accountable for determining the identity of the issuing OP.

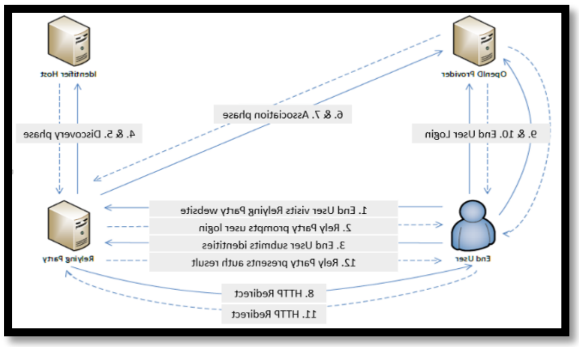

Identifier Host
OpenID Provider
6. & 7. Association phase
4. & 5. Discovery phase
9. & 10. End User Login
1. End User visits Relying Party website
2. Rely Party prompts user login
3. End User submits identities
12. Rely Party presents auth result
Relying Party
End User
8. HTTP Redirect
11. HTTP Redirect

Fig. 1 OpenID 2.0 Protocol Flow.

Figure 1 depicts the OpenID 2.0 protocol stream to validate the communication between the above-defined parts [5]:

1. The OpenID 2.0 confirmation method is start by the end user requesting a confirmation-required service of the RP.
2. The RP asks the end user to provide her/his OpenID Identifier within a signing page.
3. The end user provides her/his identifier.
4. The RP refers the expected identifier to the identifier host.
5. The identifier host determines the expected identifier and replies through the identity of its conforming OpenID provider composed through extra information.
6. (&7) In the *interior connotation stage*, the RP and the OP exchange a public secret for digitally logging in and confirming the to-be-exchanged token.
8. The RP forwards the end user to the OP.
9. (&10) The end user confirms her/his identity to the OP.
11. The OP issues a confirmation token, employed by the previously switched secret, forwards the end user back to the RP and attaches the token to the demand.
12. The RP confirms the authority of the expected token and presents the end user the effect of the confirmation method.

[1] https://oauth.net/

[2] http://www.eclipse.org/projects/archives.php

2.2 OAuth 2.0

Prior to discussing OpenID Connect it is important to understand the OAuth 2.0 protocol. OpenID Connect uses this protocol to provide identity services. OAuth 2.0 is an authorization protocol for gaining access tokens for web APIs and secure resources. OpenID Connect relies on the OAuth 2.0 semantics and streams to permit applications to admit users.

OAuth 2.0 structures, which are define by the Internet Engineering Task Force (IETF) in RFCs 6749 and 6750, became available in 2012. These structures were intended to support the improvement of authentication and authorization protocols. They provide an assortment of standardised information flows that depend on JSON and HTTP.

The OAuth 2.0 authorization structure allows a third-party (e.g. an RP) application to gain partial access to an HTTP service, either on behalf of a resource owner by coordinating an agreement communication between the resource owner and the HTTP service or by permitting the third-party application to gain access on its own behalf [6].

2.2.1 Roles

OAuth 2.0 framework involves four main parties:

- *Resource Server*: the server that holds the protected resource.
- *Resource Owner*: the individual who owns the protected resource.
- *Client*: the application that requests the access to the protected resource on behalf of the user; usually, it is a web browser.
- *Authorization Server*: a server that, on behalf of the owner, authorises the client to access the resource.

The communication between an authorization server and a resource server is beyond the scope of the OAuth 2.0 specifications. The authorization server could match the resource server or a diverse object. A single authorization server could issue access tokens as agreed by several resource servers [6].

2.2.2 Access Tokens

Access tokens are authorizations that are used to access resources. The access token is a string that acts as a representative of an authorization allotted to the client. This string is regularly impervious to the client. Tokens epitomise detailed scopes and times of access, approved by the resource owner, and that are required by the resource server and authorization server. The token could represent the identifier used to regain the authorization data or can self-hold the authorization data in a provable.

An access token provides a concept layer by changing several agreement values, such as user name and password, with one implicit token by the resource server. This concept allows issuing admission tokens more preventive than the authorization allowance that is used to gain them and also eliminates the resource server's desire to understand a varied choice of authentication approaches. The access tokens may have dissimilar arrangements, configurations, and procedures of operation, depending on resource server security [6].

2.2.3 Protocol Endpoints

There are three protocol endpoints in the OAuth 2.0 specifications. They can be defined as follows:

- *Authorization endpoint*. This endpoint is used through the client to gain authorization from the resource owner by the user agent (or UA) forwarding. The resource owner decisions are forwarded to the authorization endpoint of the authorization server so it can reply to the 'authorization request' of the client. This endpoint may have a redirection endpoint, which is found when each client has to inventory one or more forward URI(s), constructing its redirection endpoint with the authorization server that requires interconnection. The authorization grants are forwarded to the 'forwarding endpoint' of the client.

- *Token endpoint*. To regain a confirm access token, the client must refer to an HTTP POST request along with the received authorization grant as a claim value to the token endpoint of the authorization server. The user can identify herself/himself to the authorization server within this request.

- *Access token endpoint*. The authorization and token endpoints permit the client to require the scope of the admission request by using a 'scope request value'. The authorization server uses the 'scope response value' to cognize the client of the scope of the access token allotted.

2.2.4 Protocol Flow

The OAuth 2.0 protocol flow is showed in Figure 2; it defines the communication between the four parties and consists of the following steps:

A. The client sends an authorization request to the resource owner.
B. The client receives an authorization grant. This grant proofs the consent of the resource owner on the resource access.
C. The client forwards the received authorization grant to the authorization server.
D. The authorization server verifies the authorization grant; then, if the grant was successfully verified, it replies with an access token.
E. The client forwards the received access token to the resource server.

The resource server verifies the access token; then, if the token was successfully verified, it allows the access to the protected resource.

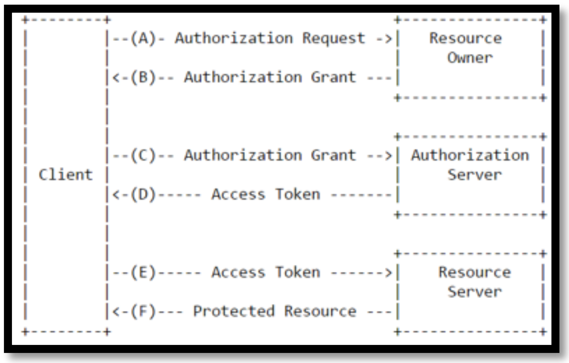

Client
--(A)- Authorization Request ->
<-(B)-- Authorization Grant ---
Resource Owner
--(C)-- Authorization Grant -->
<-(D)----- Access Token -------
Authorization Server
--(E)----- Access Token ------>
<-(F)--- Protected Resource ---
Resource Server

Fig. 2 OAuth 2.0 protocol flow.

2.3 OpenID Connect

OpenID is an open source identity management system in which IdPs provide the user with open identifiers that may be used to log in to several service providers (or SPs) [2]. An SP in OpenID terminology, is essentially an RP.

OpenID enable the user to use her/his account to log in to various SPs (e.g. websites) using a global identifier in order to eliminate the need to possess multiple identifiers along with the security credentials (e.g. usernames and passwords). Within the OpenID framework, the user controls which data associated with her/his OpenID account (e.g. name, email address, etc.) can be shared with the SPs. Through OpenID, the password is exclusive to the identity provider which is the party responsible of the user authentication and is trusted by all SPs in a given circle of trust (or CoT).

The initial version of OpenID was released in 2005, version 2.0 was released in December 2007, and the newest version of OpenID is OpenID Connect was released in March 2014. The first generation of OpenID had many implementation issues, but it was encouraged due to the new possibilities it promised. OpenID 2.0 presented good security and operated well when it was executed correctly. However, it was plagued by numerous policy limitations, primarily the fact that SPs might be web pages but not native applications.

OpenID Connect was built on top of OAuth 2.0, discussed in Section 2.1. The main limitation of the OAuth 2.0 protocol is its interoperability. Each RP wants to modify its operation for each maintained IdP that runs a unique API and endpoints for retrieving their private specific information. In addition, the level of identity guarantee is not carried in the protocol. OpenID Connect seeks to fix these limitations [2].

OpenID Connect resembles OpenID 2.0 in many architectural points. However, OpenID 2.0 uses XML and a traditional message signature scheme that, in repetition, was occasionally difficult for developers to attain. OpenID 2.0 executions would occasionally strangely cease interoperation. OAuth 2.0, the pillar of OpenID Connect, uses the Web's built-in Transport Layer Security (SSL/TLS) for encryption, which is widely adopted. Moreover, OpenID Connect uses JSON Web Token (JWT) data structures when marks are necessary. OpenID Connect is easier to deal for developers than its predecessor. OpenID Connect adheres to standard token type, standard cryptography, and validation process and combines authentication within short/long lived delegated access.

One of the biggest OpenID Connect adopters is Google. Google's OpenID Connect/OAuth 2.0 APIs are used for both authentication and authorization1. Google Sign-in and Google Client services are built on those APIs. User authentication is carried out by finding a specific ID token and verifying it. The ID token specifications are set by OpenID Connect as a part of the OpenID Connect's agreement with Google. There are two methods for user authentication, namely, the server flow and the implicit flow. The server flow method delegates the back-end server to authenticate the user's identity; whereas in the implicit flow method the user authentication is performed at client-side using a JavaScript execute it by the browser.

2.3.1 Roles

The OpenID Connect framework involves three parties. The relationship between these roles can be seen in Figure 3 [5]. The parties are:

- *The End User.* The end user is represented by her/his user agent (UA), and requests certain services from the client. Thus, they need to verify their identities to the

1 https://developers.google.com/identity/protocols/OpenIDConnect/

client. Moreover, the end user has the ability to authorize the client to admit a definite set of their resources, defined by the scope and claims value, in their names.

- *Client.* A client is an application that facilitates the authentication of an end user who requests access to services and the conforming OpenID provider (OP). Thus, the client asks the end user for her/his ID token for verification and the OP for the OAuth 2.0 access token to grant access to requested secure resources of the end user. The provisions on the application state that the client has the capability to authenticate himself or herself to the OP.
- *OpenID Provider (OP).* The OP provides an ID token by holding a detailed set of claims verifying the identity of the end user. It also generates an OAuth 2.0 access token to the client for the requested resources after it successfully authenticated and authorized the end user.

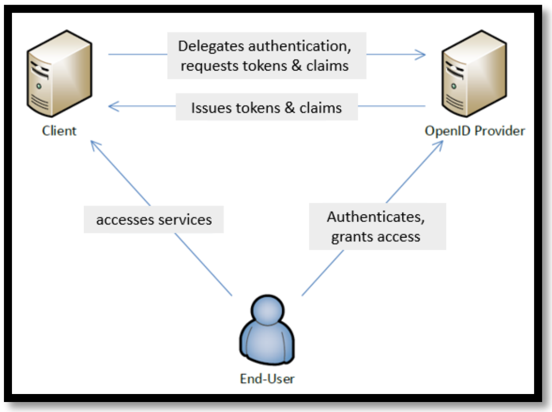

Delegates authentication, requests tokens & claims
Issues tokens & claims
Client
OpenID Provider
accesses services
Authenticates, grants access
End-User

Fig. 3 The relationship between roles on OpenID connect 1.0 Protocol.

2.3.2 The Endpoints

There are five types of endpoints within the OpenID Connect framework; namely:

- *Authorisation endpoint.* This is the OP server endpoint wherever the user is asked to authenticate and allow the client application admission to the user's identity (ID token) and, theoretically, other demanded details, such as email and name (named UserInfo claims). This is the endpoint that the user requests to intermingle with the OP, by the user agent, that role is classically via the web browser.

- *Forwarding endpoint.* Each client must register one or more Forward Uniform Resource Identifier URIs with the OP it needs to interconnect with. The client then augments a specific URI as a demand value to the initial authentication request. This URI must compete at a minimum of one of the client's registered ones. Therefore, after successfully relating with the OP, the end user can be forwarded to the forwarding endpoint of the client. The endpoint can accept an authorization code, ID token, access token, or grouping of the three as the invitation value.

-
- *Token endpoint.* This endpoint confirms the client application, then it provides it with the code resulted from the authorization endpoint for an ID token and access token. The token endpoint could, moreover, receive other OAuth 2.0 allowance types instead of issue tokens.

- *JavaScript Object Notation (JSON) web key set endpoint.* If asymmetric cryptography is used to digitally sign and encrypt the ID token, the client requests the signature verification and decryption keys from the OP. The JSON web key set endpoint of an OP holds a JSON web key set (JWKS) obtaining that for the need to the public keys.

- *User information endpoint.* To verify the end user claims, a client has the option to request certain end user information via the end user information endpoint of the conforming OP. This request must hold the access token issued by the OP after the end user was successfully authenticated. All the communications with user information endpoint must be SSL/TLS encrypted.

The following is an example of a user information endpoint:

```
{
"sub": "2482889761001",
"name": "Name Example",
"given_name": "Given Example",
"family_name": "Family Example",
"preferred_username": "g.family",
"email": "ginenFamily@example.com",
"picture": "http://example.com/ ginenFamily/me.jpg"
}
```

2.3.3 ID Token

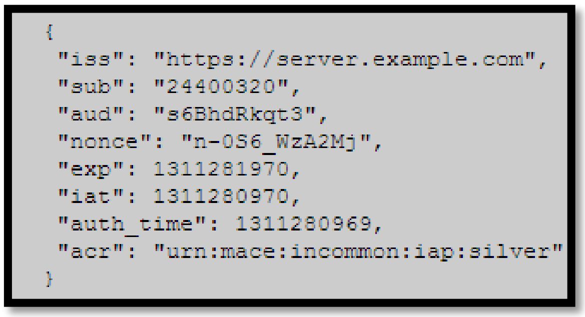
The ID token bears a resemblance to the idea of an identity card. To obtain one, the client asks to refer the end user to its OP with an authentication request.

The ID token is a security token inclosing claims about the end user. the token must be signed by the OP using the JSON web signature (JWS), and could also be encrypted using JSON web encryption (JWE). The token holds the following list of items:

- The user identifier, termed subject in OpenID (sub).
- The issuing OP (iss).
- The audience identifier (i.e., client; aud).

- A nonce (nonce).
- The authentication time (auth_time).
- The authentication method (acr).
- The issuance time (iat).
- The expiration time (exp).
- It can also contain some information about the end user, such as name and email address.
- If digitally signed, it would contain the signature value.

The following example is the JWT claims set in an ID token:



```
{
 "iss": "https://server.example.com",
 "sub": "24400320",
 "aud": "s6BhdRkqt3",
 "nonce": "n-0S6_WzA2Mj",
 "exp": 1311281970,
 "iat": 1311280970,
 "auth_time": 1311280969,
 "acr": "urn:mace:incommon:iap:silver"
}
```

2.4 Higgins

Higgins-identity is an information card-based identity management (ICIM) tool that provides a protected and simple method for users to manage and protect their personal information. Also. it allows RPs (e.g. websites) to verify end user claims for authentication and authorization purposes. Further, it can be identified as a personal data service (PDS) that allows users to control shared personal data.

There are different versions of Higgin. The first, Higgin 1.0, was released in 2008 and is based on a visual card-wallet metaphor. This card is named an information cards, or an i-card. In 2011, Higgins 2.0 was released; it is based on different code to implement a back-end services PDS. A PDS is a cloud-based service that works on behalf of the end user.

Higgins Personal Data Service (PDS) and Local Attribute Data Storage (ADS) are server modules with which the user communicates during the authentication phase, as shown in Figure 4.

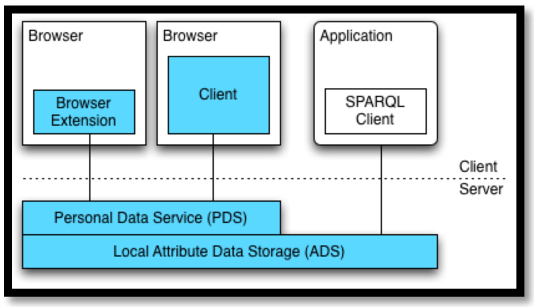


Fig. 4 Higgin 2.0 structure.

Higgins protocol workflow is as follows:

1. **UA → RP**: HTTP request: to GET (sign-in web page).
2. **RP → UA**: HTTP/S response. A sign-in page is reversion holding the Higgins-enabling labels in which the RP security rule is set in.
3. **User → UA**: The RP website page proposals the choice to usage Higgins. If it is first time to use it, the Higgins 'identity selector' will show the identity of the RP and provide the user the choice to either continue or terminate the protocol.
4. **Identity Selector → ADS**: The identity selector registered Higgins identities.
5. **User → ADS**: The user selects one of his Higgins identities.
6. **Selector→Higgins**. The identity selector generates a Security Token value as a reply to the Request Security Token (RST) to the Higgins.
7. **UA → RP**: forwards the received Security Token.
8. **RP → UA**: confirms the validity of the Security Token.

2.5 OWASP Tools

The Open Web Application Security Project (OWASP)1 was founded on the first of December 2001. It is a non-commercial organisation that sets operational and security standards for web applications. Also, it is an open society devoted to qualifying organisations to understand, develop, evaluate, turn on, and maintain trusted applications.

OWASP frequently publishes a list of the top ten application security risks along with recommended

[1] http://www.owasp.org

measures for each risk to be mitigated. The newest list was released in 2017 [7].

2.5.1 OWASP Zed Attack Proxy (ZAP)

The Zed Attack Proxy (ZAP) is a security analysis tool for discovering vulnerabilities in web applications. ZAP is one of the most prominent and active OWASP projects. It is aimed to be used through people with a wide assortment of security knowledge.

ZAP's main functions is to crawl websites actively and passively to scan web applications for vulnerabilities [8]. ZAP provides the following services:

1. *Interrupting Proxy*. By acting as an interrupting proxy ZAP helps monitoring the traffic flow request/response, interrupting it, and editing it on the pass.
2. *Automatic Scanner*. The automatic scanner recognises the security aperture that exists in the web application by putting on a real attack. Thus, it analyses the security position of an application with dynamism.
3. *Passive Scanning*. By analysing the responses from the server to identify security issues.
4. *Brute Force Scanner*. This scanner enforces access controls on files and directories.
5. *Spidering*. This aims at crawling websites to detect vulnerabilities.
6. *Dynamic SSL Licenses*. Such licenses are used to interrupt requests/responses to/from the server.

Figure 5 shows the basic ZAP screen.

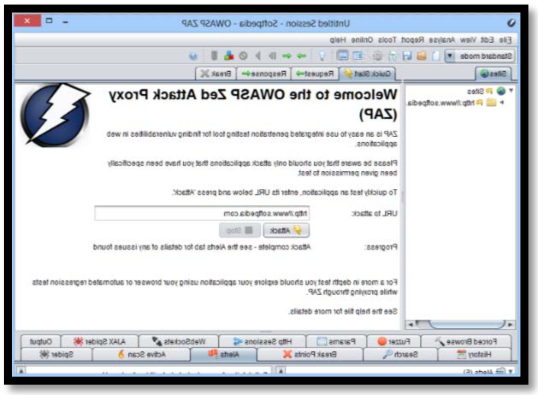
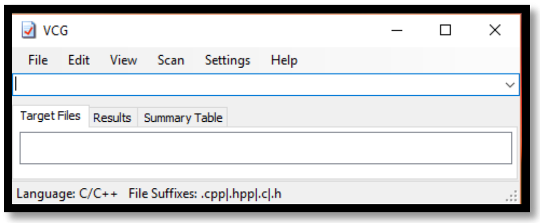


Fig. 5 A basic screenshot of ZAP.

2.6 The Visual Code Gripper

The Visual Code Gripper (VCG) is open-source static code analysis tool. It was developed by the NCC Group. The VCG has the ability to examine software without executing it.

VCG performs a 'white-box' analysis, and is capable of analysing both web and non-web software to discover security in issues in the inputs and outputs; these issues cannot be discovered by web scanning tools such as ZAP. Figure 6 shows the basic VCG screen.



Language: C/C++ File Suffixes: .cpp|.hpp|.c|.h

Fig. 6 A basic VCG screen shot.

2.7 Related Work

In this section we spot the light on previous related work. In [9] a comparison study was conducted between protected SSO protocols with regard to convenience and safety. The study focused on three SSO protocols, namely: LDAP, SAML, and OpenID. The security analysis is an investigation of publicly available data on the web. Another study [10] investigated tools and methods for growing the pliability and dependability of IdPs centred on OpenID. In addition, it has categorised the types attacks depending on the impact severity.

In [2] Sun proposes improvements to enhance of the security of web SSO system. He illustrated that though OpenID and OAuth have been approved via IdPs, including Google, Facebook, Yahoo, and Microsoft, as well as millions of RP websites, the normal user still poorly understands web SSO. Sun concluded that users need to advance their understanding and that enhancements to usability and security would assist them in doing so. Sun investigated users' opinions and concerns towards the usage of web SSO systems and implemented a systematic security analysis of OpenID 2.0. Moreover, Sun has studied the OAuth 2.0 executions of three major IdPs and 96 popular RP websites by examining browser-relayed requests through SSO. Finally, He has proposed a novel method by which websites could stop SQL injection attacks and a user-centric access control solution that would improve both Open ID and OAuth protocols. Similar study was also discussed in [11].

Bochum and Mainka advanced a security analysis tool that was based on OpenID SSO [12]. Since OpenID is commonly used by several SPs, including Own Cloud, Word Press, Open Street, Drupal, and Map, they documented that security should be examined in the standings of malicious users and network attackers. They presented an attack of OpenID that involved an open-source malicious OpenID IdP. Such malicious IdPs are able to create OpenID signing tokens for uninformed user identities that can hint to serious security defects. They developed a tool to address this issue. However, this tool can only analyse verified traffic. It is probable to operate any Open ID claims value of uninformed switched messages. The tool is flexible and permits an alteration in IdP performance in every single OpenID stage. Table 1 shows a comparison between the mentioned related studies.

Table 1. A comparison between the mentioned related studies

Author	Method	Result
Nick Heijmink – [9]	Compared and implemented security investigations on three SSO protocols: LDAP, SAML, and OpenID	They concluded that the LDAP protocol was designed for local networks, not web requests, and that SAML is out of date and is no longer use for web requests.
Kreutz, D., Feitosa, E., Cunha, H., Niedermayer, H., & Kinkelin, H – [10]	OpenID IdPs: Integrity, Availability, and Confidentiality for Information and Process.	They concluded that Ops could achieve good throughput, and are appropriate to support thousands of users.
Sun, S.-T – [2]	Systematic Analysis of the OpenID 2.0 and OAuth 2.0 Protocol.	They concluded that users lack security awareness, and that enhancements to usability and security would help.
Mainka, C – [12]	Development of a Security Analysis Tool for OpenID.	The tool is flexible and controls attack directions.
Sun, S.-T., Hawkey, K., & Beznosov, K – [11]	Systematic Analysis of the Protocol that uses both a Formal Model Checking an Empirical Evaluation.	They suggested an easy, scalable method for OpenID-enabled websites (RPs) to mitigate the risk of man-in-the-middle attacks even without SSL/TLS.

All of the discussed work omits the fact that OpenID is vulnerable to a series of CSRF attacks that are imperfect or have been poorly implement. Moreover, they all focus on vulnerabilities and weaknesses within the authentication phase. Finally, they have not proposed a solution for heterogeneous RPs and IdPs.

3. Security Analysis of the OpenID Protocol

This chapter is divided into four parts. The first two parts outline vulnerabilities and attacks on OpenID Connect along with the attacker's capabilities. The third part describes our security analysis using OWASP ZAP and discusses the results. Finally, the last part describes our security analysis using the VCG and discusses the results.

3.1 Vulnerabilities within OpenID Connect

OpenID and OpenID suffered from several security issues. Many of those issued have been addressed in OpenID Connect. OpenID Connect has resolved many problems by adopting an asymmetric cryptography framework. This framework provides essential security services such as confidentiality, integrity, authentication and non-repudiation. OpenID Connect has reached a better acceptance level, compared to its predecessors, due to its simplicity in both usage and integration. However, there are a number of security vulnerabilities in OpenID Connect that have not been addressed yet. This section discusses those vulnerabilities.

3.1.1 Vulnerabilities of the UA

In this section we briefly describe OpenID Connect vulnerabilities reside in the UA.

- *Covert Redirect Vulnerability.* This vulnerability allows attackers to disclose the end user protected data by redirecting the UA (e.g. a web browser) to malicious sites[1]. An attacker can generate OAuth 2.0 tokens to deceive users. Once the user has logged in to the malicious site, the attacker will be able to obtain access to the user's data stored at the OP sever. This vulnerability is called the 'covert redirec' since the attacker can defeat the pretentious protocols by a pop-up request over Facebook, for example, and impersonate genuine websites. Covert redirect vulnerability happens because there is not a reliable verification procedure of the forwarded URLs. In the case of Facebook, the data at risk are personal such as age, email address, work history, friends list, mailboxes, online state, etc.

- *Phishing vulnerability.* One of the main concerns associated to OpenID Connect is phishing. An attacker can deceive users' into log in to a malicious OP in order to obtain the users' security credentials.

[1] http://tetraph.com/covert_redirect/oauth2_openid_covert_redirect.html

3.1.2 Vulnerabilities of the RP

One of the most serious OpenID Connect vulnerabilities is the Second-order vulnerability that is associated with RP. This vulnerability is related to the exchanged messages amongst the framework parties (i.e. client and end user, client and OP, and OP and end user). A detailed description of this vulnerability can be found in [13].

3.2 Attacks on OpenID Connect

This section discusses a number of attacks on OpenID Connect.

3.2.1 Attacks on the UA

One of the most famous attacks that can target OpenID Connect is The 307 Redirect attack [10]. In the OAuth protocol, the UA is redirected from the IdP to the RP after the user has been authenticated by the IdP. This attack can be launched if the IdP uses an HTTP status 307 redirect in the POST request. This redirection method can be easily modified by an attacker controlling the UA to deceive the RP and access protected resources [14].

3.2.2 Attacks on the OP

In this section we list attacks on the OpenID Connect OP [13].

- *A Server Side Request Forgery (SSRF) attack.* An SSRF attack is realised by exploiting vulnerable web services in the Internet to bypass firewalls.
- *Denial-of-service (DoS) attacks.* There are many techniques by which an attacker can launch a DoS attack on the OP.
- *Code injection attacks.* The attacker can forge the ID an Access tokens via code injections in order to impersonate legit users and log in to the OP.
- *Breaking the end user authentication.* Using attacks like replay or wire-tapping.
- *Man-in-the-middle (MITM) attack.*

3.2.3 Attacks on the end user

By launching attack such as Masquerade attack, the attacker can impersonate the end user by falsely obtaining HTTP cookies [15].

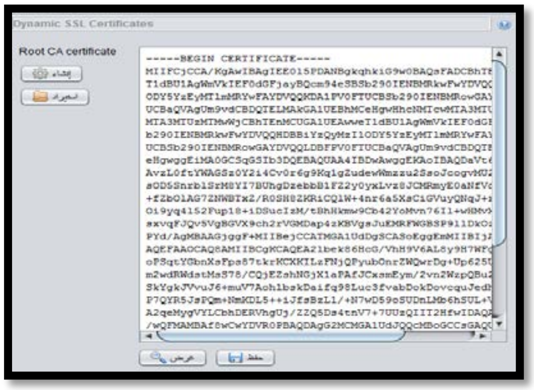
3.2.4 Attacks on the RP

A good example of such attacks would be the Identity provider mix-up attack by which the attacker deceives the RP by falsely using an access token that was issued for a legit user [14].

3.3 OWASP ZAP Analysis

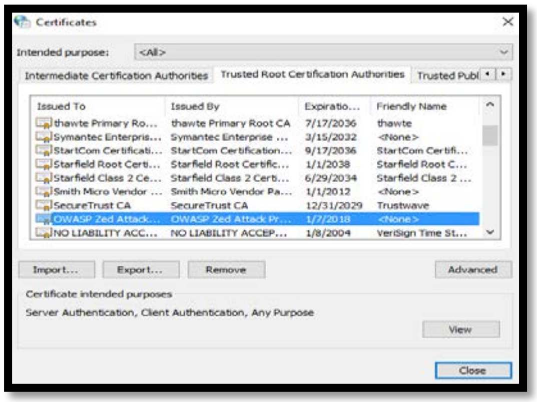
Our OWASP ZAP analysis consists of the following steps:

1. First we configure the web browser to use ZAP as a proxy (ZAP use: Address: localhost and Port: 8080)
2. Once ZAP is set as a proxy between our web browser and the web service, the connection will be interrupted by an SSL/TLS (HTTPS) certificate authentication request since ZAP encrypts and decrypts all inbound and outbound traffic using SSL/TLS. ZAP will automatically be issued with SSL/TLS certificates by ZAP's own Certificate Authority (CA). To make the web browser trust that CA, we must first import and trust the ZAP's Root CA certificate. On the bar menu, we select 'Options' then 'Dynamic SSL Certificates' and save it as shown in Figure 7. After that, we install the ZAP certificate as a 'Trusted Root Certificate', as shown in Figure 8.



```
-----BEGIN CERTIFICATE-----
MIIFCjCCA/KgAwIBAgIEE015PDANBgkqhkiG9w0BAQsFADCBhTE
T1dBU1AgWmVkIEF0dGFjayBQcm94eSBSb290IENBMRkwFwYDVQQ
ODY5YzEyMTlmMRYwFAYDVQQKDA1PV0FTUCBSb290IENBMRowGAY
UCBaQVAgUm9vdCBDQTELMAkGA1UEBhMCeHgwHhcNMTcwMIA3MIU
MTA3MTUzMTMwWjCBhTEnMCUGA1UEAwweT1dBU1AgWmVkIEF0dGE
b290IENBMRkwFwYDVQQHDBBiYzQyMzI1ODY5YzEyMTlmMRYwFAY
UCBSb290IENBMRowGAYDVQQLDBFPV0FTUCBaQVAgUm9vdCBDQTE
eHgwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaVt
AvzL0ftYWAGSz0Y2i4Cv0r6g9Kq1gZudewWmzzu2SsoJcogvMU
sOD5SnrblSrM8YI7BUhgDzebbB1FZ2y0yxLvz8JCMRmyE0aNfV
+fZbOlAG7ZNWBTx2/R0SH8ZKRiCQlW+4nr6a5XsCiGVuyQNqJ+
Oi9yq4152Fup18+iDSucIzM/tBhHkmw9Cb42YoMvn76I1+wHMv
sxvqFJQv5VgBGVX9ch2rVGMDap4zKBVgsJuEMRFWGBSP911DkO
PYd/AgMBAAGjggF+MIIBejCCATMGA1UdDgSCASoEggEmMIIBIjA
AQEFAAOCAQ8AMIIBCgKCAQEA21bek86HcG/VhH9V6AL8y9H7WF
oPSqtYGbnXsFps87tkrKCKKILzFNjQPyubOnrZWQwrDg+Up625
m2wdRWdstMs878/CQjEZshNGjX1aPAfJCxsmEym/2vn2WzpQBu
SkYgkJVvuJ6+muV7AohlbskDaifq98Luc3fvabDokDovcquJed
P7QYR5JsPQm+NmKDL5++iJfsBzL1/+N7wD59oSUDnLMb6hSUL+
A2qeMygVYLCbhDERVhgUj/ZZQ5Ds4tnV7+7UUzQIII2HfwIDAQ
/wQFMAMBAf8wCwYDVR0PBAQDAgG2MCMGA1UdJQQcMBoGCCsGAQ
```

Fig. 7 A ZAP root CA certificate.



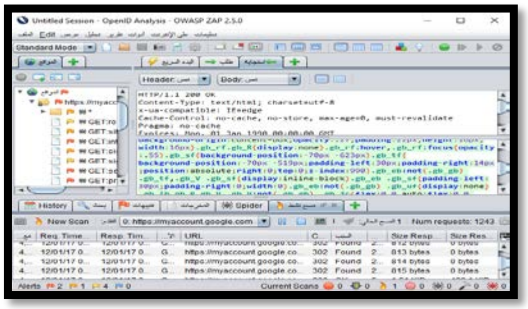
Issued To	Issued By	Expiratio...	Friendly Name
thawte Primary Ro...	thawte Primary Root CA	7/17/2036	thawte
Symantec Enterpris...	Symantec Enterprise ...	3/15/2032	<None>
StartCom Certificati...	StartCom Certification...	9/17/2036	StartCom Certifi...
Starfield Root Certi...	Starfield Root Certific...	1/1/2038	Starfield Root C...
Starfield Class 2 Ce...	Starfield Class 2 Certi...	6/29/2034	Starfield Class 2 ...
Smith Micro Vendor ...	Smith Micro Vendor Pa...	1/1/2012	<None>
SecureTrust CA	SecureTrust CA	12/31/2029	Trustwave
OWASP Zed Attack...	OWASP Zed Attack Pr...	1/7/2018	<None>
NO LIABILITY ACC...	NO LIABILITY ACCEP...	1/8/2004	VeriSign Time St...

Certificate intended purposes

Server Authentication, Client Authentication, Any Purpose

Fig. 8 Trusted root certification authorities.

3. We implement a 'Quick Start' test after typing the full URL of the web service that we want to analyse. After clicking the Attack button, ZAP will start crawling the web service by its spider, then passively scanning all retrieved pages. Figure 9 shows the run of an attack we launched on the Gmail log in subordinate to Google Corporation).

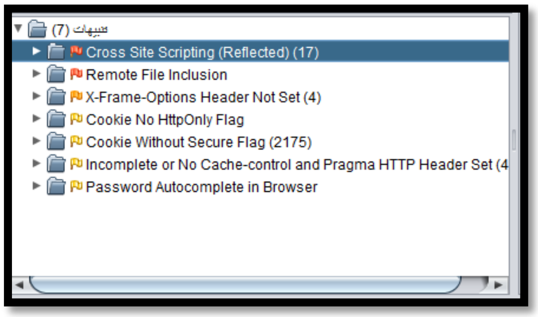


```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
x-ua-compatible: IE=edge
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: Mon, 01 Jan 1990 00:00:00 GMT
```

```
0: https://myaccount.google.com        Num requests: 1243
Req. Time    Resp. Tim...  URL                              C...  Size Resp.  Size Res...
12/01/17 0.. 12/01/17 0..  https://myaccount.google.co...   302   Found  2..  813 bytes  0 bytes
12/01/17 0.. 12/01/17 0..  https://myaccount.google.co...   302   Found  2..  814 bytes  0 bytes
12/01/17 0.. 12/01/17 0..  https://myaccount.google.co...   302   Found  2..  815 bytes  0 bytes
Alerts 2 1 4 0                          Current Scans 0 0 1 0 0 0 0
```

Fig. 9 Passive scan of a Gmail account.

After passively scanning the ZAP records and the requests and responses sent to all HTTP pages, ZAP shows alerts beside each suspicious record. Figure 10 shows ZAP alerts (red flags).



```
▼ تنبيهات (7)
  ▶ Cross Site Scripting (Reflected) (17)
  ▶ Remote File Inclusion
  ▶ X-Frame-Options Header Not Set (4)
  ▶ Cookie No HttpOnly Flag
  ▶ Cookie Without Secure Flag (2175)
  ▶ Incomplete or No Cache-control and Pragma HTTP Header Set (4
  ▶ Password Autocomplete in Browser
```

Fig. 10 Alerts after finishing a scan.

3.3.1 Results

After a planned ZAP analysis, we found the following vulnerabilities in OpenID Connect:

- *Cross-site scripting (XSS).* Cross-site scripting (XSS) defects occur when a web service receives an untrusted confirmation and refers it to a web browser devoid of appropriate authentication. XSS permits attackers to implement scripts in the victim's browser in order to hijack the user's sessions, damage websites, and/or forward the user to malicious websites. While the HTTPOnly flag is comprised in the HTTP reply header, the cookie cannot be retrieved over the client side script. XSS attacks aim to holdup session cookies. A server should set the HTTPOnly flag on the cookies it issues so that it cannot be accessible on the client side.

- *Remote File Include (RFI).* OpenID Connect is vulnerable to the RFI attack. Once web services get user responses (URL, claims value, etc.) and permit them in the instructions file, the web service can be deceived into with remote files through malicious code. Practically all web service structures support file implying. File implying is mostly used for packaging public code into dispersed files, which are indicated via main service elements. if a web service used a comprise file, the code in this file can be implemented indirectly or obviously by occupation definite actions. An attacker can use RFI to execute malicious code on both the server and/or client sides. To resolve this problem, we must apply a reliable input authentication policy.

3.4. Static code analysis using VCG

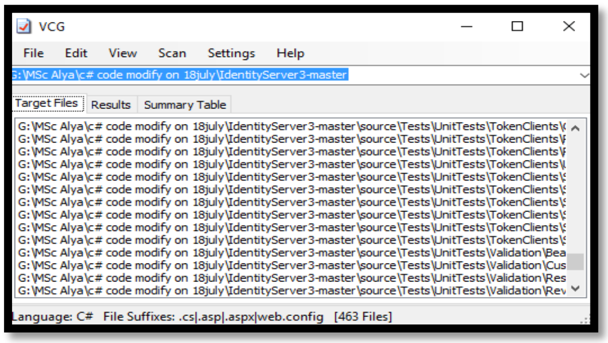
We have conducted a 'static code analysis' (or source code analysis) using VCG as a code evaluation tool. This analysis is performed at the execution level of the security development lifecycle (SDL). Static code analysis aims at discovering vulnerabilities within the static code (i.e. source code) via various methods (e.g. defect analysis, data stream analysis, and few more).

Visual code grepper (VCG) is an automatic code security evaluation tool that supports multiple languages such as C#, C++, PHP, VB, Java and PL/SQL.

Our VCG analysis consists of the following steps:

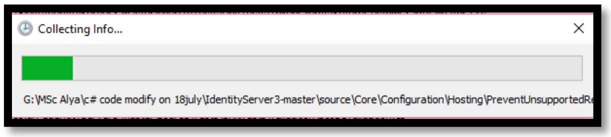
1. Download the source code of OpenID Connect written in C# language.

2. We configure the VCG tool to evaluate C# code. The, we open the OpenID Connect code, as shown in Figure 11. Finally, from the scan menu bar, we choose 'scan only code' as the 'scan type'. Figure 12 shows the scan progress.



```
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\(
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\F
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\F
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\I
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\S
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\S
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\S
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\S
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\S
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\TokenClients\S
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\Validation\Bea
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\Validation\Cus
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\Validation\Res
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Tests\UnitTests\Validation\Rev
Language: C#   File Suffixes: .cs|.asp|.aspx|web.config   [463 Files]
```

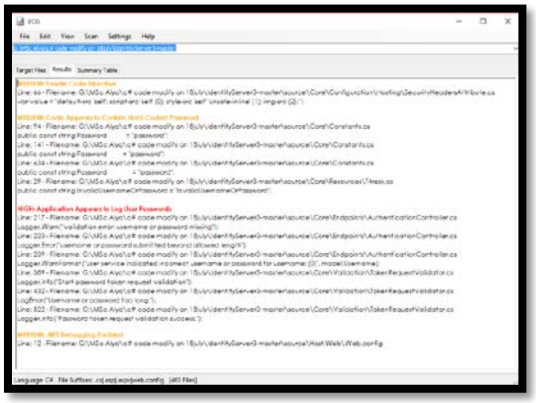
Fig. 11 Opening the OpenID connect code.



```
G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Configuration\Hosting\PreventUnsupportedR
```

Fig. 12 Starting the scan OpenID connect code.

Our static code analysis discovered a number of serious vulnerabilities, as shown in Figure 13. and sorts the weaknesses code that was discovered. The following image details these weaknesses:



```
MEDIUM: Unsafe Code Directive
Line: 66 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Configuration\Hosting\SecurityHeadersAttribute.cs
var value = "default-src 'self'; script-src 'self' {0}; style-src 'self' 'unsafe-inline' {1}; img-src {2};";

MEDIUM: Code Appears to Contain Hard-Coded Password
Line: 94 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Constants.cs
public const string Password          = "password";
Line: 141 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Constants.cs
public const string Password          = "password";
Line: 634 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Constants.cs
public const string Password          = "password";
Line: 29 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Resources\T4resx.cs
public const string InvalidUsernameOrPassword = "InvalidUsernameOrPassword";

HIGH: Application Appears to Log User Passwords
Line: 217 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Endpoints\AuthenticationController.cs
Logger.Warn("validation error: username or password missing");
Line: 223 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Endpoints\AuthenticationController.cs
Logger.Error("username or password submitted beyond allowed length");
Line: 239 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Endpoints\AuthenticationController.cs
Logger.WarnFormat("user service indicated incorrect username or password for username: {0}", model.Username);
Line: 369 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Validation\TokenRequestValidator.cs
Logger.Info("Start password token request validation");
Line: 432 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Validation\TokenRequestValidator.cs
LogError("Username or password too long");
Line: 522 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Core\Validation\TokenRequestValidator.cs
Logger.Info("Password token request validation success.");

MEDIUM: .NET Debugging Enabled
Line: 12 - Filename: G:\MSc Alya\c# code modify on 18july\IdentityServer3-master\source\Host.Web\Web.config

Language: C#   File Suffixes: .cs|.asp|.aspx|web.config   [483 Files]
```

Fig. 13. The results of the VCG source code scan. (red indicates a high threat and orange indicates a medium threat)

The discovered vulnerabilities are:

- *Invalidated/forwards redirects* **(**unsafe code directive**).**

```
var value = "default-src 'self'; script-src 'self' {0}; style-src 'self' 'unsafe-inline' {1}; img-src {2}; ";
```

This means that the web services could redirect or forward users to another website or service based on information that has not been properly validated. Attackers can exploit the vulnerability to forward victims to malicious web sites or use forwards to authenticate fake pages [16]. Also, attackers can use the 'unsafe' directive that permits the use of C-style pointers in the code, this will cause unexpected performance with memory leaks and crashes or buffer overflows.

- *Hard-coded password* **(**code appears to contain a hard-coded password**).**

```
public const string Password = "password";
public const string InvalidUsernameOrPassword =
"InvalidUsernameOrPassword";
```

The code has a hard-coded password that an attacker can obtain from the source code or by disassembling while the running. This vulnerability is a challenging one and often ignored; however, it is very sensitive.

- *User passwords and information to log files* **(**application appears to log user passwords**).**

```
Logger.Warn ("validation error: username or password missing");
```

The code logs user passwords in log files which results in generating a threat of qualification theft. This could be exploited by an attacker to log false access consents or inject malicious content on the logs. Coders usually use log files to save the history of actions. Depending on the environment of the code, revising log files can be done either manually or automatically using a tool that automatically rejects suspicious logs or imposed records. Discovering such logs would not be straightforward if the attacker has figured out a way to write them using a legit service. For example, an attacker can write false records in the log file by launching a code injection attack on a legit service and force it to write false records in the log file. Moreover, the attacker could try to modify the structure of the log file or inject invisible characters to it in order to make it difficult for automatic log files managing tools to discover false records. Finally, thae attacker can indirectly modify log files' indicators. This can be realised be injecting code and/or commands into the log files [16].

- *CSRF* **(**.NET debugging enabled**).**

The code execution is designed so that it returns .NET debugging information to help developers finding code bugs. However, this could leak private data and/or useful information about the code that can be used to attack the system. A cross-site request forgery (CSRF) is an attack by which the attacker can forge HTTP requests to obtain such information. An attacker can build malicious web pages that produce fake HTTP requests and deceive victims by appearance labels, XSS, or many other methods [16].

4. Proposed Security Enhancements

In this section we discuss a security solution for OpenID Connect that was proposed lately by OpenID developers. This solution mandates the use of cryptographic hash and token binding. Then, we propose a high-level and generic integration model of OpenID Connect with Higgins, this will result in a number of practical and security advantages.

4.1 Enhanced Authentication Profile (EAP)

OpenID has proposed a modification in the OpenID Connect specifications that permit users to authenticate OpenID Providers using strong cryptographic authentication mechanisms. This is achieved by mandating the usage of cryptographic hash and token binding in order to resolve RFI attacks.

The cryptographic hash function provides an instrument to examine the integrity of messages. A hash function returns an arbitrarily message as an input then creates, a generally much smaller, fixed length hash, called hash value or some time called digest. A cryptographic hash is used to generate a Message Authentication Code (MAC), which is a symmetric cryptography mechanism for message authentication. By using MACs, the integrity of all transferred messages will be preserved. Examples of hash functions include: MD5 (produces a 128-bit hash value), SHA-1 (produces a 160-bit hash value) and many more [17].

Token binding protects the authentication stream from XSS and CSRF attacks and invalidated/forwards redirects attacks. It will protect the ID Token over the SSL/TLS session during the authentication phase. This solution mandates the usage of a token binding ID within the ID token instead of the RSA digital signature. This technique is more efficient and practical and it uses SHA-256 cryptographic hash1. Also, this solution would decrease the size of the ID token.

This is an example of an ID token before using the token binding ID:

```
  {
   "iss": "https://ServerExample.DomainExample.com",
   "sub": "NzrgLsXh8uDCcd-
6MfNwXF4W_7noWX5FZAfHkxZsRGC9Xs",
   "aud": "0d8f5947e-bcj45-46]b2-957cf-043c88aa5ecc ",
   "nonce": "n-0R6_WzA2Mj",
   "exp": 131124381970,
   "iat": 13124280970,
   "sub_jwk": {
     "kty":"RSA",
     "n":
"0vxfdf7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFb
WhM78LhWx

4cbbfAAtVT86zwu1RK7aPFFxuhDR1L6tSoc_BJECPebW
KRXjBZCiFV4n3oknjhMs
     tn64tZ_2W-
     SD08qNLyrdkt-
bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-
G_xBniIqb
     w0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
     "e":"AQAB"
    }
  }
```

This is an example of an ID token that uses token binding ID (tbh field holds the token binding hash and cnf field holds the confirmation claim):

```
 {
   "iss": "https://ServerExample.DomainExample.com",
   "sub": "0fgh6LkoE3Ks23PyxQ",
   "aud": "0d8f5947e-bcj45-46]b2-957cf-043c88aa5ecc",
   "iat": 146715103451,
   "exp": 146217151651,
   "nonce": "1KjVrfsFnQRd4V2XC6",
   "cnf":{
     "tbh":
"l1X0aVl3repikNqDhaH92VwGgrFdAY0tSackYis1r_-fPo"
    }}
```

4.2 A Proposed Integration Model

As we discussed in Section 3, our security analysis of OpenID Connect using OWASP ZAP and VCG tools showed that OpenID suffers from an number of security issues such as XSS (which is ranked amongst the OWASP's top ten most influential Internet attacks in 20172), the use of hard-coded passwords, and implementing Invalidated/forwards redirects. Our proposed integration model aims to address these issues.

Integrating OpenID Connect with Higgins (discussed in Section 2.3) could help boosting up the security, scalability and practicality of OpenID Connect. Higgins is an Information Card based Identity Management (ICIM) system [1]; and its security tokens are issues by a Security Token Service (STS). The use of an STS is mandatory for IdPs and optional for RPs; However, in our proposed model we assume that all RPs are equipped with an STS. Also, we assume that the end user's browser understands Higgins browser extensions (HBX). In addition, we assume that RP is Higgins-enabled, whilst the IdP is both OpenID and Higgin-enabled. Finally, we assume that the user is an OpenID Connect and Higgins user.

The proposed model's protocol-run is as follows.

1. **UA → RP**: it is a HTTP request: to GET (sign-in web page).
2. **RP → UA**: HTTP/S response. A sign-in page is reversion holding the Higgins-enabling labels.
3. **HGX ↔ UA**: The Higgins extension service retrieves the RP security policy to determine the registered claim set and which ones to embed in the information card.
4. **User → UA**: The user confirms using Higgins to log in. If it is first time, the identity selector on the user machine will show the identity of the RP and provide the user the choice to either continue or terminate the protocol.

[1] http://openid.net/wg/eap/

[2] https://www.owasp.org/index.php/Top_10_2017-A3-Cross-Site_Scripting_(XSS)

5. **Identity Selector → ADS**: The identity selector highlights the ADS policy and the remnant information.
6. **User → ADS**: The user provides her/his consent on the selected claims and ADS policy.
7. **Identity Selector**: The identity selector generates a Request Security Token (RST) message.
8. **Identity Selector ↔ HGX**: The RST will not be forwarded to the RP; instead, the HGX intercepts it and converts it to an OpenID authentication request.
9. **HGX → OP**: The HGX forwards the OpenID authentication request to the OpenID Provider (OP).
10. **OP**: The OP authenticates the user.
11. **OP → HGX**: If the authentication was successful, the OP will reply with an authentication response to the RP.
12. **HGX → RP**: The HGX verifies the received OpenID authentication response, and if successfully verified, it creates a Higgins-like security token, and forwards it to the RP.
13. **RP → user**: The RP verifies the received security token, and if successfully verified, it logs the user in.

All messages must be SSL/TLS protected. Figure 14 sketches the protocol flow.

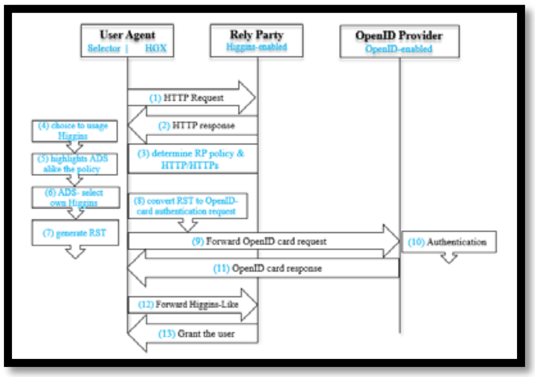

User Agent
Selector | HGX
Rely Party
Higgins-enabled
OpenID Provider
OpenID-enabled
(1) HTTP Request
(2) HTTP response
(3) determine RP policy & HTTP/HTTPs
(4) choice to usage Higgins
(5) highlights ADS alike the policy
(6) ADS- select own Higgins
(7) generate RST
(8) convert RST to OpenID-card authentication request
(9) Forward OpenID card request
(10) Authentication
(11) OpenID card response
(12) Forward Higgins-Like
(13) Grant the user

Fig. 14 Proposed model's protocol-run.

4.3 Analysis of the Enhancement Method

In the previous section, we have analysed OpenID Connect and discovered a number of sensitive vulnerability, such as XSS, RFI, Invalidated forward redirects, hard-coded passwords, log file exposure, and CSRF. Also, we have described a newly adopted solution by OpenID that aims to enhance OpenID Connect's security and efficiency by introducing a cryptographic hash function and a Token Binding value to the transferred tokens. In addition, we have described our novel high-level integration model the facilitates a systematic integration of OpenID Connect and Higgins Identity. In this section we provide a brief analysis of these two enhancements.

4.3.1 Analysis of the Enhanced Authentication Profile

The enhancement, which mandates the presence of a cryptographic hash function and a Token Binding value within tokens, can efficiently help mitigating the risk of MITM, replay and phishing attacks by mandating a good cryptographic protection of the fresh nonces within the tokens. Hence, an attacker will not be able to reply a token or intervene and create fake nonces without the cryptographic keys. The use of cryptographic hash function minimises the size of the token; this boosts up efficiency and practicality in both processing and transforming.

4.3.2 Security Analysis of Our Proposed Integration Model

The Integration model does not require any changes in the IdP's configuration and processes. It mandates the use of SSL/TLS, and this can be easily achieved, given the wide adoption of this protocol on the web nowadays. By implementing this model, users of OpenID Connect can log in to Higgins-enable RPs; this increases the scalability of the SSO's CoT and provide more security and user-convenience.

In order to mitigate the risk of a phishing attack, the model states that the token forward task is performed by a verified browser extension on the user machine instead of the RP. The log file vulnerability is still an issue since the model does not involve an enhanced logging system. However, The IETF has specified the syslog protocol in RFC 5424. This standard specifies a reliable logging system that produces, records, filters, and investigates log messages. We believe that by adhering to this standard, most of the OpenID Connect log file issues can be resolved.

An attack cannot produce a Higgins-like security token since she/he does not have an access to neither the ADS nor the private key. Similar to the original Higgins process, no private information of the end user will be leaked to the RP. However, in the integration model, and although that the RP is Higgins-enable, it does not even have to know what user attributes are registered with the Higgins IdP. This is a good privacy enhancement. The HGX requests and retrieves the security tokens by itself without any intervention of the RP; this would dramatically mitigate the risk of XSS, RFI, Invalidated redirects, and CSRF.

A limitation of the proposed model is that it does not protect against dishonest IdPs. However, an IdP is a trusted third party by definition.

5. Concluding Remarks and Future Work

In this paper we have provide an overview of OpendID and investigated its security. Also, we have conducted a security analysis of it. The analysis consisted of two parts; first, an analysis of the static of source code (written in C# language) using the VCG tool. This discovered four weaknesses in the code: unsafe code directive, code that appears to contain a hard-coded password, an application that appears to log a user password, and .NET debugging-enabled code. Secondly, an analysis of the HTTP messages' using OWASP ZAP tools. This discovered two weaknesses: cross-site scripting (XSS) attacks and remote file include (RFI).

There are several security issues within OpenID. Some of these issues have been addressed in OpenID latest version: OpenID Connect. Although some of these issues remained in OpenID Connect, like for example the XSS, CSRF and Invalidated/forward redirect vulnerabilities, an effective solution has been proposed by OpenID afterwards. This solution involves the use of a cryptographic hash function and a Token Binding value within tokens. However, other security issues (e.g. log files exposure) are still unresolved. In addition, we have proposed a novel high-level integration model of OpenID Connect and Higgins. This integration should result in a number of advantages with regard to security, privacy, practicality and scalability. A brief analysis of OpenID proposed enhancements and our proposed integration model has been given.

In future, we will conduct similar security analysis to one described in this paper to other IDMSs like for example, Liberty Alliance by Kantara Initiative[1] and/or Shibboleth by Shibboleth Consortium[2]. Also, we will investige possible integrations between OpendID Connect and other IDMSs.

References

[1] Waleed A. Alrodhan. Privacy and Practicality of Identity Management Systems: Academic Overview. VDM Verlag Dr. Müller GmbH, Germany. ISBN 978-3639380255. 2011.

[2] San-Tsai Sun. Towards improving the usability and security of Web single sign-on systems. Ph.D thesis, University of British Columbia. November 2013.

[3] Hyun-Kyung Oh and Seung-Hun Jin. The Security Limitations of SSO in OpenID. Proceedings of the 10th International Conference on Advanced Communication Technology. 2008.

[4] Abu Shohel Ahmed. A User Friendly and Secure OpenID Solution for Smart Phone Platforms. M.Sc. thesis, Faculty of Information and Natural Sciences, School of Science and Technology, Aalto University. June 2010.

[5] Julian Krautwald. Security Analysis of the OpenID Connect Standard and its Real-life Implementations. M.Sc. thesis, RUHR University Bochum. 2014.

[6] Internet Engineering Task Force (IETF). RFC 6749: The OAuth 2.0 Authorization Framework. 2012. https://datatracker.ietf.org/doc/rfc6749/ [last accessed: November 2017].

[7] The Open Web Application Security Project (OWASP). Top 10 Application Security Risks. 2017. https://www.owasp.org/index.php/Top_10_2017-Top_10 [last accessed: November 2017].

[8] Lars Kristensen and Stefan Østergaard Pedersen. Multi-step Scanning in ZAP-handling Sequences in OWASP ZAP. M.Sc. thesis, Applied Mathematics and Computer Science, Technical University of Denmark. 2014.

[9] Nick Heijmink. Secure Single Sign-on Comparison of Protocols. M.Sc. thesis, CCV & Radboud University Nijmegen. 2015.

[10] Diego Kreutz and Eduardo Feitosa and Hugo Cunha and Heiko Niedermayer and Holger Kinkelin. Increasing the Resilience and Trustworthiness of OpenID Identity Providers for Future Networks and Services. Proceedings of Ninth International Conference on Availability, Reliability and Security (ARES), pages 317–324. Switzerland. 2014.

[11] San-Tsai Sun and Kirstie Hawkey and Konstantin Beznosov. Systematically breaking and fixing OpenID security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures. The Computers & Security journal, Elsevier. Volume 31, Issue 4, Pages 465–483. June 2012.

[12] Christian Mainka. Developing a Security Analysis Tool for OpenID-based Single Sign-On Systems. Bachelor thesis, Ruhr-Universität Bochum. November 2013.

[13] Vladislav Mladenov and Christian Mainka and Julian Krautwald and Florian Feldmann and Jörg Schwenk. On the Security of Modern Single Sign-On Protocols: Second-Order Vulnerabilities in OpenID Connect. The Computing Research Repository journal, volume abs/1508.04324, eprint 1508.04324. 2015.

[14] Daniel Fett and Ralf Küsters and Guido Schmitz. A Comprehensive Formal Security Analysis of OAuth 2.0. The Computing Research Repository journal, volume abs/1601.01229, eprint 1601.01229. June 2016.

[15] Marino Miculan and Caterina Urban. Formal analysis of Facebook Connect Single Sign-On Authentication Protocol. In SofSem, Proceedings of Student Research Forum, pages 99–116 2011.

[16] The Open Web Application Security Project (OWASP). The Ten Most Critical Web Application Security Risks. 2013. https://www.owasp.org/index.php/Top_10_2013-Top_10 [last accessed: November 2017].

[17] Dieter Gollmann. Computer Security. John Wiley & Sons; 3rd ed. edition. ISBN 0470741155. February 2011.

[1] https://en.wikipedia.org/wiki/Kantara_Initiative

[2] https://www.shibboleth.net/