FH | JOANNEUM
University of Applied Sciences

Best Practice Authentication and Authorization

in distributed Business Scenarios with OpenID Connect and OAuth 2.0

Master Thesis
submitted in conformity with the requirements for the degree of
Master of Science in Engineering (MSc)
Master's degree programme **IT & Mobile Security**

FH JOANNEUM (University of Applied Sciences), Kapfenberg

Supervisor: Elmar Krainz, FH JOANNEUM Kapfenberg

submitted by: Cornelia Rauch
personnel identifier: 1610419026

June 2016 <edit date!>

Assignment for the master thesis of
<your name>
Matr. no. 1400000000

Subject:
“<Title of Your Thesis>”

Abstract

Write your abstract here.

<place>, <date>

Academic adviser:
<firstname lastname>

<your name>

Formal declaration

I hereby declare that the present master's thesis was composed by myself and that the work contained herein is my own. I also confirm that I have only used the specified resources. All formulations and concepts taken verbatim or in substance from printed or unprinted material or from the Internet have been cited according to the rules of good scientific practice and indicated by footnotes or other exact references to the original source.

The present thesis has not been submitted to another university for the award of an academic degree in this form. This thesis has been submitted in printed and electronic form. I hereby confirm that the content of the digital version is the same as in the printed version.

I understand that the provision of incorrect information may have legal consequences.

Kapfenberg, 28.09.2018

Cornelia Rauch

Acknowledgement

Thanks to . . .

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the rise of social networks, online services experienced a revolution quite recently and thus have had a significant impact on the way private information gets propagated on the Internet. Information provided by online services is accessed by third-party applications with Application Programming Interfaces (API). Making third-party applications, which were not known upon service subscription able to access information. The trust is no longer just between two parties but also may include some third-parties which bring up concerns about the privacy of personal information, raising the need of secure authorization (Cirani et al., 2015). Sakimura et al. (2014), expresses that for the user to be able to use third party information, users are often required to create new accounts for each service they want to access. The complex task of managing all these tasks can lead to identity theft if not done right. In traditional architectures, a third party receives the user's credentials for the user to be able to access information from the third party as stated before. Pakhard (2016) however points out that sharing the credentials with a third party might lead to security problems. Third parties might be responsible for fatal security gaps, for example by storing passwords in plain-text. Pakhard (2016) further states that the compromise of one third-party will then lead to the compromise of the credentials of the end user. As a result, the safety of the secure resources cannot be granted anymore. Moreover, third-parties will receive the comparatively more significant amount of access to the user data than needed. In modern architectures, the process of authentication and authorization of the user is often implemented by a third party. An example of a such a modern Infrastructure is the Cloud. NIST Cloud Computing Standards Roadmap (2011) suggest for identity management, to rely on standards and specifications that are widespread and documented. Example for tools and standards include Internet protocols for accessing services like REST, SOAP, and XML and federate identity standards for service au-thentication such as

SAML, OAuth, and OpenID Connect. In this thesis, the focus is on modern distributed architectures and ways to provide users secure access to protected resources and confidential management of the user's credentials. The focus is on the protocols OpenID Connect an OAuth 2.0.

Scientific Question What are feasible ways to implement authentication and authorization for enterprises that depend on adaptability and focus on single page applications? Furthermore, what benefits of the token-based Authentication with OAuth2 and OpenID Connect are there versus traditional authentication and authorization methods? The questions should be elaborated while considering common security risks of modern Web Applications.

Terms and definitions. Technical terms … abbreviations are summarised at the end (in "Acronyms"), e.g. application binary interface (ABI) or man-in-the-middle (MITM). If ABI is referenced again, only the acronym is printed (as hyperlink though).

Code listings require the *listings* package which, in turn, requires some settings[1]; see command `\lstset{}` in preamble of this template. Additionally the package *courier* should be used because the defaults do not provide for proper syntax highlighting. Here is a reference to listing **??**.

[1]… because the defaults do not fit all purposes

Chapter 2

Authentication and Authorization

"Cloud-based services, the social Web, and rapidly expanding mobile platforms will depend on identity management to provide a seamless user experience." (Corre et al. (2017)).

Modern Devices are changing our everyday life. They change the way how we access information, interact with each other and share content. With this change of user behavior also the way we think of authentication and authorization methods has to adjust. Users find themselves struggling using multiple devices, accounts, and services. The user's burden of this site-by-site account management is putting security at risk. The goal of new authentication and authorization solutions is to help the user managing his accounts by providing single-sign-on, based on an exchange of identity-related assertion across security domains in a scalable way [cf. (Corre et al. (ibid.))].

2.1 Security Considerations

Before getting further into the topic of authentication and authorization, this section will shed light on some basic security Principles concerning authentication and authorization that help to understand the need for authentication and authorization mechanisms.

Some basic design principles formulated by Saltzer and Schroeder, 1975 in 1975 were paraphrased by Neumann, 2013 and are still relevant today. The principles give an underlying overview of what should be the focus when designing a secure system. The ten basic security principles are:

- The economy of mechanism: Keep the design as simple as possible.
- Fail-safe defaults: Access should not be explicitly denied instead it should be explicitly permitted. For example when using Access Control Lists (ACLs) all access should be explicitly denied by default.
- Complete mediation: Every access to every object has to be checked for authority without exceptions.
- Open design: The design of an application should not be secret, it can not be assumed that design secrecy will enhance security. This principle is applied in cryptography. The design of cryptographic algorithms is available for the public, just the keys remain secret.
- Separation of privileges: Two keys should be used to protect resources if feasible and privileges should be separated.
- Least privilege: Every application and user should be provided with lest privileges they need to complete their job. The existence of overly powerful mechanisms such as superuser is inherently dangerous.
- Least common mechanism: Minimize the amount of mechanism common to more than one user and depended on by all users.
- Psychological acceptability: Keep it simple. The design of the interface for the user should be easy to understand so that the user routinely and automatically applies the protection mechanism correctly.
- Work factor: Make cost-to-protect commensurate with threats and expected risks. It should not be possible to circumvent the mechanism with the resources of the attacker.
- Recording of compromises: Provide trails of evidence which are tamper-proof and difficult to bypass.

Besides formulating these very important principles which will be discussed in various forms, Saltzer and Schroeder, 1975 also discus the terms "privacy" and "security". Those therms get frequently used by authors writing about information storing systems, like in this paper. However the terms "privacy" and "security" are often used very differently. ibid. for example, define "privacy" as the ability of an individual to specify whether, when and to whom sensible information is released. And they describe security as a technique that can control who is able to modify resources on a computer. Another more recent description of the terms "security" and "privacy" comes form Brooks et al., 2017. ibid. state the importance of the distinction between privacy and security. This distinction is between privacy and security are essential because there are security issues unrelated to privacy, just as there are privacy issues that are unrelated to security. While security concerns arise from illegal system behavior, privacy concerns arise from byproducts of authorized personally identifiable information (PII) processing. Even byproducts that are considered to protect PII can raise security concerns, for example, it can be questioned to which degree a tool for persistent activity monitoring should reveal information about individuals that are related to security purposes. However, security and privacy have in common that they want to protect personal information and resources or PII.

These security issues and privacy issues, of course, raise particular concerns for users as well as for companies offering authentication services. When it comes to protecting personal resources, there are three primary concerns. According to Todorov, 2007 those three concerns are:

- Confidentiality: The term confidentiality means that personal information is protected from disclosure to unauthorized individuals and organizations.
- Integrity: The integrity of information is protecting information from accidental or intentional tampering. Modification of confidential data may affect the data validity.
- Availability: Availability is the need to be able to access information at the time a user requests it. The availability of the services that exposes information has to be given as well.

In an ideal world companies offering those services will do everything to use the best technologies regarding countermeasures to protect confidentiality, integrity, and availability. Establishing countermeasures, however, can be costly leading to a trade-off between costs and level of production of information. A typical approach to establishing information security management is to analyze risks first and then form counter measurements [cf. (ibid.)].

The result of all design principles, security and privacy considerations should be a system, network or component that is trustworthy. According to Neumann, 2013 trustworthiness is given if an entity satisfies its specified requirement, after a reliable assessment. The requirements that deserve especially consideration are, those that are critical to an enterprise, mission, system, network or other entity. One of the requirements to make a system trustworthy is a reliable authentication and authorization process. These processes are discussed in more detail in the next sections.

2.2 Authentication

"Digital identity is the unique representation of a subject engaged in an online transaction. The process used to verify a subject's association with their real-world identity is called identity proofing" (Grassi, Garcia, and L. (2017))

A digital identity as explained above is the result of what is called the authentication process. It is a way of identifying the user as whom he claims to be. For a very typical authentication process the user provides its username and password when the application demands it. If the user provides a correct username and password, an application assumes the user is indeed the owner of the account he wants to log on [cf. (Boyed (2012))].

The evidence provided by the user in the authentication process is called credentials. Most of the time as mentioned above credentials get provided in the form of username and password. Nevertheless, credentials also may take other forms like PIN's, key cards, eye scanners and so on [cf. (Todorov (2007))].

Credentials, which prove the identity of an entity and find use as authenticators in authentication systems, are called factors. Grassi, Garcia, and L. (2017) categorize following types of factors:

- Something the user knows - Cognitive information the user has to remember. Examples include passwords, PIN, answers to secret questions.
- What the user has - something the user owns. Examples include a security token, driving license, one-time password (OTP). What the user is - biometric information of the user. Examples include fingerprint, voice, and face.
- What the user is - biometric information of the user. Examples include fingerprint, voice, and face.

Other types of information which are not considered authentication factors but can be used to enrich the authentication process according to Dasgupta, Arunava, and Abhijit (2017) are:

- Where the user is - the location of the user can is used as a fourth factor of authentication. Examples include GPS, IP addresses.
- When the user logs on - Time can also be extracted as a separate factor. Verification of employee's identification in different office hours can prevent many kinds of grave data breaches. The time factor can easily prevent online banking fraud events to a great extent.

To secure a solution properly, it should at least use two factors of the three listed above. To make use of more than one factor of a pool of potential credentials to verify the identity of a user is referred to as Multi-factor Authentication (MFA). The goal of multi-factor authentication is it to provide a layered defense and make it harder for unauthorized individuals to gain access. If one of the factors breaks, the service can still rely on the non-compromised authentication factors [cf. (Dasgupta, Arunava, and Abhijit (2017))].

Using just one factor is called Single Factor Authentication(SFA). ibid. clearly describes the drawbacks SFA has compared to MFA, primarily the universal used password-based authentication. The user needs to remember different passwords for multiple accounts, therefore, the user often reuses one password also known as password fatigue.

In an Interview by Tomkins, 2009 with Jon Brody, he explains Password Fatigue like the following. An average user has 15 accounts; some people might even have up to 30 accounts - far too much to manage appropriately. Users then tend to adopt specific password patterns like using simple passwords for nontransactional sites and complex passwords for banking sites. Since many complex passwords are hard to remember users also often reuse passwords for different services at one point - this is called password fatigue.

Besides password fatigue Todorov, 2007 draws attention to one of the significant challenges of secure user authentication represented by default passwords. Vendors often ship their devices with pre-configured standard passwords. Although vendors recommend changing default passwords, system architects and engineers often fail to do so because they are more focused on the business logic than on security causing security issues. Systems with default passwords are more straightforward to attack, for example by knowing or guessing the device the attacker has an easy time authenticating with the system accordingly.

Due to the problem of password fatigue and default passwords, one factor like a password might easily get compromised. The user can then no longer use the service until the repair of the system which can lead to a delay of the user when trying to access necessary information. Also, there is a risk that the user does not notice the compromise of the single factor which can lead to devastating effects [cf. (Dasgupta, Arunava, and Abhijit, 2017)].

When designing an authentication process by using multiple factors the designers of the process should be acutely aware of the type of application and the information that has to be secured. For example, a solution for an international bank should have different standards than an app for a making a grocery list. On the one hand, challenging and complex authentication processes for trivial applications might scare away users. On the other hand, simple methods for applications protecting sensitive data might drive users away as well. Application designers should always try to find a middle way that suits both parties, the application owners and the users [cf. (Grassi, Garcia, and L., 2017)].

The factors are an important part of the authentication process which result should be an authenticated user. However, to become an authenticated user the user has to go through certain steps on the way. These steps the user has to go through are enabled be typical components unique to the authentication process. Todorov, 2007 identifies three typical components that are part of the authentication:

- The Supplicant is the party that provides the evidence to prove the identity of a user or client. The result of the authentication process should be the authenticated user or client.
- The Authenticator, also called server, is responsible for ascertaining the user identity. After prove of identity, the authenticator can authorize or audit the user access to resources.
- Security Authority Database, which is storage or a mechanism to check the user's credentials. The storage can be represented by as much as a flat file, a server on the network providing centralized user authentication or a distributed authentication server.

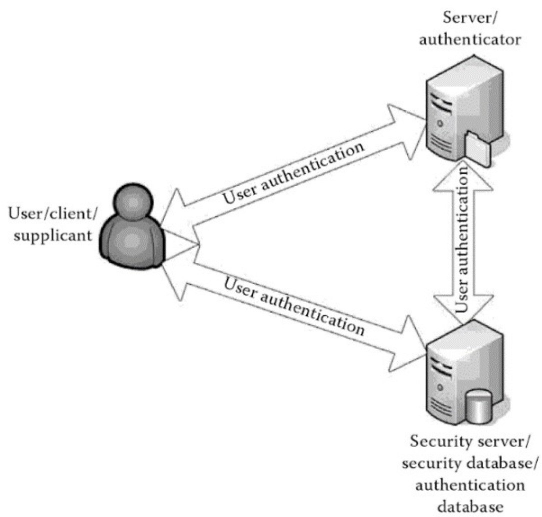

Figure 1.1 Components of a user authentication system.

Figure 2.1: Components User Authentication System

It is vital that all the components like shown in (figure 2.1), of a user authentication system can communicate independently of each other. Whether or not all communication channels are used depends on the authentication mechanism and the model of trust that it implements. For example, the Kerberos authentication protocol does not feature direct communication between the authenticator and security server [cf. (Todorov, 2007)].

Application security is a complex task and developing a customized siloed identity solution can be expensive. A Stand-alone identity store can besides being expensive also causes information assurance and administrative problems for organizations [cf.(JerichoSystems, 2018)].

A federate authentication or identity federation says Boyed, 2012 is a system that is maintaining its accounts, for example, username and password databases, with the help of third party service. Often big cooperate IT environments already use such solutions. Environment applications, for example, may trust an Active Directory server, an LDAP server or a SAML provider. Grassi, Garcia, and L., 2017 also claims that identity federation is preferred over some siloed identity solution that each serve a single agency or Relying Party (RPs). Furthermore ibid. lists certain benefits that come with using federated architectures, as can be examined before.

- Enhanced user experience. For example, an individual can be identity proofed once and reuse the issued credential at multiple RPs.
- Cost reduction to both the user (reduction in authenticators) and the agency (reduction in information technology infrastructure).
- Data minimization as agencies does not need to pay for collection, storage, disposal, and compliance activities related to storing personal information.
- Pseudonymous attribute assertions as agencies can request a minimized set of attributes, to include claims, to fulfill service delivery.
- Mission enablement as agencies can focus on the mission, rather than the business of identity management.

To reflect on authentication it can be said that authentication is a very important part of every applications. As users are getting more concerned on security the pressure on developers grows to provide a solution that secures sensible data while keeping up usability standards, which can often be a trade-off. Complex applications need complex security which can mean high costs for individually developed solutions; therefore application developers should also think about using federate authentication solutions.

2.3 Authorization

The authentication and authorization process are very closely related to each other and for users often hard to separate. After the authentication process of a user, the application has now proof that the user is who he claims to be, but not every user is the same. After the user authenticates, the user may want to access data or services. Based on the information provided by the authentication process the application has the possibility to allow or deny the user to access information or services. In other words we have to check if the user is authorized to access data of a service. Furthermore authorization, offers granular control to distinguish between read, write or execute access to individual resources, typically access control list (ACL) are used for each operation [cf.(Todorov, 2007, Boyed, 2012)].

Systems offer a user login process or sign-in process, in order to receive the information necessary to make authorization decisions. The login process initiates the authentication process between user and the system. As a result of this process, the user receives a system or application specific structure called an access token. The access token holds information about the user which will indicate what kind of resources the user can access. For every action the user then has to provide the access token and based on the information provided within the access token is then either granted or denied access [cf. (Todorov, 2007)].

Delegated authorization is granting access to another person or application to act on behalf of the user. Boyed, 2012

2.4 Token-base Authentication

The way web developers write back-end applications has changed significantly with the rising popularity of single page and mobile applications. Backend-developers no longer spend a lot of time building markup. Instead, they build APIs for front-end applications to consume. The split up of front-end and back-end allows the back-end to focus on business logic and data management while the front-end solely focus on the representation of the content. The number one way single page and mobile web applications are authenticating users according to Tkalec, 2015 is token-based authentication.

Sevilleja, 2015 shares the view that token-based authentication is the modern way to handle authentication. Token-based authentication should be considered because of various factors. It is optimal for mobile applications which work in a stateless way and need to adapt to the sudden change of demand in a scalable way. Furthermore, token-based authentication provides extra security and applications can pass on authenticated users to other applications. However before taking a closer look at token-based authentication, it should be taken into account how Sevilleja, 2015 and Tkalec, 2015 concluded that token-based authentication is the best alternative for modern web applications to authenticate their users. Therefore, Sevilleja, 2015 for example examines how authentication was done in the past. One approach to authenticate users used in th past that puts token-based authentication in perspective is server-based authentication.

Server-Based Authentication. A lot of modern-day API's built on the Representational State Transfer(REST) programming paradigm which basis is the HTTP protocol which is stateless as well. A protocol that is stateless does not recall the actions that were taken beforehand which means for example that if we authenticate the user, in the next request we have to authenticate the user again because the application will not know the user anymore [cf. (ibid.)].

The aim of server-based authentication is for the application to remember the user that logged on at the application. The application has to store the information on the server, which can be done in a few different ways on the session, usually in memory on the disk. The workflow of a server-based authentication starts with the server delivering the website and the user logging in with username and password. The server saves the information from the user login info in a session. After the session is established, the session is checked on the server for every request. If the session is valid, the server returns the requested data to the server. However, since modern single-application ant mobile applications are on the rise, this method to authenticate user shown some problems, especially when it comes to scalability [cf. (ibid.)].

The session handling with server-based authentication is especially hard on the server's bandwidth. Most of the time the session gets established in memory on the server when the user authenticates. This approach leads to an enormous overhead. The second problem with this approach is that the information of the user is held in memory on the server. Since more and more companies are moving servers to the cloud, this is not only a security issue but replicating servers to scale is limited. Also nowadays users want to access their data at any moment from every device. Providing the user with the possibility to access data across multiple mobile devices is vital, which means cross-origin resource sharing has to be enabled. With server-based authentication, it is possible to run into problems with the forbidden request when the user tries to access data from another domain [cf. (Sevilleja, 2015)].

Token-based authentication with JSON Web Token The most important thing about token-based authentication is that it is stateless, much like HTTP. The server does not have to hold the session of the user over an extended period on the server. Instead, the user can request resources by offering a token generated by an authentication server. The token send in the query string or Authorization header can then be validated at the resource server, and the secure resource will be returned to the user. The approach of using JSON Web Tokens gives one the ability to scale applications without considering on which domain the user logged on. Another advantage besides being scalable is that token-based authentication gives one the possibility to reuse the same token for authenticating the user. Therefore, it gets easier to build applications that share permission with other application because many separate servers, running on multiple platforms and domains can reuse the same token. The approach also gives performance advantages compared to server-side authentication because there is no need to find and deserialize the session on each request. However, since it is best practice to encrypt the token, the token still needs to be validated, and the content needs to be parsed. One way to implement token-based authentication is with the help of JSON Web Tokens. JSON Web Tokens are gaining popularity fast and are backed by huge companies like Google and Microsoft. Also the Internet Engineering Task Force defines a standard specification. OpenID and OAuth also use the JSON Web Token as a standard; therefore, the usage of the JSON Web Token will be explained in detail [cf. (Tkalec, 2015)].

JSON Web Token (JWT) is a compact structure that holds information about the authentication of a user or claims. The structure is indented for space-constrained environments such as HTTP Authorization headers. The payload of the JSON Web Token is of JSON, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and encrypted [cf. (Jones, Bradley, and Sakimura, 2015)].

The standard is used to transport data between interested parties. The transferred data can be for example the identity of a user or user's entitlements. Furthermore, with the possibility of digital signatures and encryption data can be transferred securely over an unsecured channel. The signatures also allow asserting the identity of a user if the recipient trusts JWT asserting party [cf. (Siriwardena, 2016)].

An example of a JWT Token is an id_token. Google provides for Developers an OAuth 2.0 Playground, where developers can choose a scope and try it out against the Google

API. To get an excellent example of a JWT, I selected the OAuth2 API v2 and authorized the API [cf. (Inc., 2018)].

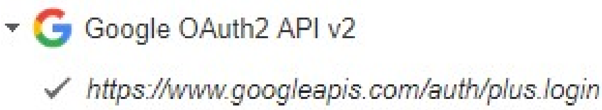

Figure 2.2: Google Developers Playground OAuth 2.0 API

After choosing and authorizing, the API Google returns an Authorization Code. This Authorization Code is specific for a certain Authentication Flow defined by OAuth and can then be exchanged for the tokens. As a result all information for the scopes that where selected beforehand is returned. Because OAuth 2 API v2 was selected, the returned value is an id_token. This id_token is a nice representation of a JWT and holds the standard information that are required according to the OAuth specification [cf. (ibid.)].


```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImRhZDQ0NzM5NTc2NDg1ZWMzMGQyMjg4NDJlNzNh
Y2UwYmMzNjdiYzQifQ
.
eyJhenAiOiI0MDc0MDg3MTgxOTIuYXBwcy5nb29nbGV1c2VyY29udGVudC5jb20iLCJh
dWQiOiI0MDc0MDg3MTgxOTIuYXBwcy5nb29nbGV1c2VyY29udGVudC5jb20iLCJzdWIi
OiIxMTIzMDE5MzgzMjI4MTAyMzk3MTIiLCJlbWFpbCI6ImNvcm5lbGlhcmF1Y2hAZ214
LmF0IiwiZW1haWxfdmVyaWZpZWQiOnRydWUsImF0X2hhc2giOiJSdjJRQjZoZzFqdDZR
aHRJWG5laWVnIiwiZXhwIjoxNTI5NzQ3MTk2LCJpc3MiOiJodHRwczovL2FjY291bnRz
Lmdvb2dsZS5jb20iLCJpYXQiOjE1Mjk3NDM1OTYsIm5hbWUiOiJDb25ueSBSYXVjaCIs
InBpY3R1cmUiOiJodHRwczovL2xoNC5nb29nbGV1c2VyY29udGVudC5jb20vLWJFbEt3
aDVaNUJnL0FBQUFBQUFBQUFJL0FBQUFBQUFBQk5rL0p5Qm1XTW9uYUlJL3M5Ni1jL3Bo
b3RvLmpwZyIsImdpdmVuX25hbWUiOiJDb25ueSIsImZhbWlseV9uYW1lIjoiUmF1Y2gi
LCJsb2NhbGUiOiJkZSJ9
.
yibOtrXy9_-cfOWytzwGuE4zqLv-MK_2-PYIKR_xecJt9ACnMnNMSmio6i8Vu7U061wF
OTb-qRennHbvy3lTRZLcTXttIrIUl-NdnZs2BrTSGWrw9aRzEjIHAXiY4fGRHj9VZXs_
_J3Nn0EoBmT7Cnua2hb4U_X3hAyGpEvlSGKc5HvbyzOAtNh081Cyj1TI-AidCPTuC5vh
68C55tLJ87PWNm8WU1rCPOPBdVTjhYlqJKCpgUJ39_p_MXL_uHBZXRrvbOyV_tZVlw47
rjd8GFnBq1QqsYAR-6wrFbNL1pY6tPyriqZnQdi5KqYWPUwGpXbFDUfhZAmWXT8-PTsc
gQETEp8o3RvRHtfSu8Gx4UOhukt9_VxVdHmpFw
```


Listing 2.1: JWT Token

The JWT Token in the figure above is presented as a sequence of URL-save base64url-encoded values. The different values are separated by a ('.') character. How many parts a JWT has is dependent on how the JWT is serialized. Either by using the JWS Compact Serialization or JWE Compact Serialization [cf. (Jones, Bradley, and Sakimura, 2015)].

To make sense of this JWT token in the figure above it is best to look at each of the three parts of the JWT separately. When decoding the first part of the JWT we receive a JSON object. Each part can be decoded individually but if a quick representation of a token is needed developers are best advice to use https://jwt.io/. The website not only decodes the information of the token it also verifies the token.The figure above shows a warning that pops up if the token is not flawless. The verification of the token and the signing is done with multiple libraries that inform about certain vulnerabilities of the JSON Web Token. The first decoded part of the JWT gives us the following JSON.

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

Listing 2.2: JOSE Header

This is the JOSE Header, representing the type of the token, the cryptographic operation applied and optionally additional properties of the JWT. Based on the information of the JOSE Header it can be said if the JWT is a JWS or a JWE. When speaking of JWT we speak of one of the implementations of JWT because in fact JWT does not exist itself. Concrete implementation of the JWT are JSON Web Signature (JWS) or JSON Web (Encryption). The figure below gives an optical representation of the structure [cf. (Siriwardena, 2016)].

2.5 Single Sign-On

The aim of Single Sign-On(SSO) is to design an authentication system that serves the interests of the user as well as the interests of the service provider. Whereas the user prefers a simple process, the service provider requires a complicated authentication procedure. Ironically trying to make the authentication procedure more save often leads to weakening the whole system due to the user always finding new ways to bypass it. An example mentioned before is password fatigue. Another challenge that SSO is trying to tackle is that standard web authentication solutions that require the user to login with a password, only authenticate the user and are not capable of providing access control or revealing additional information about the user. Most SSO solutions

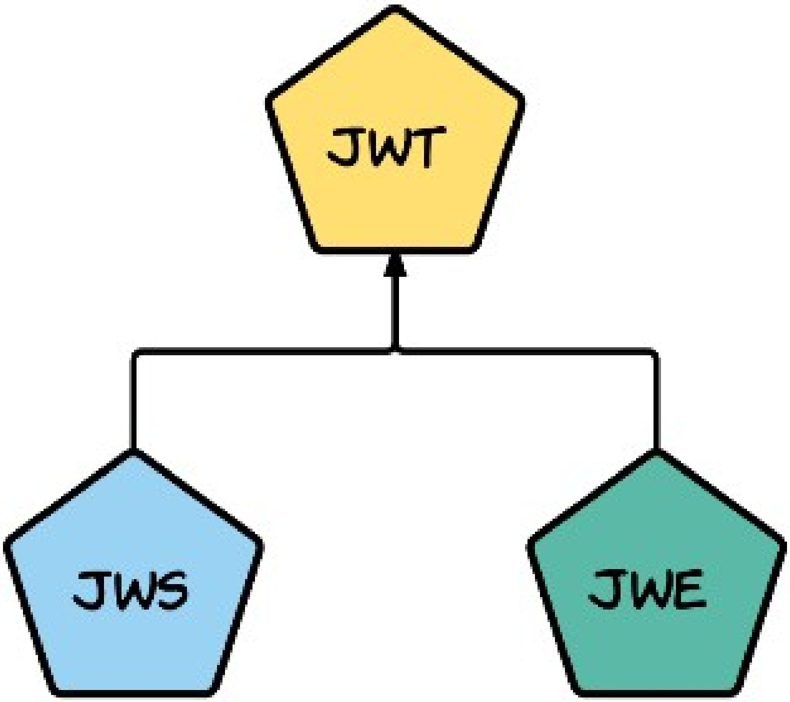

Figure 2.3: JWT Inheritance

try to combine the authentication process and authorization [cf. (Procházka, Kouřil, and Matyska, 2010)].

According to Lynch, 2011 two SSO solutions gained broad acceptance. On the one hand, SAML-based federations using SOAP, focusing on large enterprises also including governments and educational networks. On the other hand, the Web Authorization Protocol was introduced which is a combination of the Protocols OpenID and OAuth. SAML federations have been customized to address the security concerns of those institutions that typically have a large user base, significantly protected resources, complex authorization patterns and data and services spread across multiple domains. However, in a Web 2.0 world, the SAML solutions where seen as too rigid and too severe to maintain; a lightweight SSO was needed, therefore the Web Authorization Protocol was designed. The approach of this protocol is taking advantage of the lightweight RESTful APIs which are reusing the existing HTTP architecture features and the JavaScript Object Notation(JSON).

One of the well-known solutions based on Security Assertion Markup Language version 2 (SAML2) is Shibboleth. Shibboleth is one of the leading middlewares for building identity federations in a higher education sphere. To offer authentication, authorization and attribute assertion between entities. Procházka, Kouřil, and Matyska, 2010 identifies following entities defined by Shibboleth are:

- Identity Provider (IdP)
- Service Provider (SP)
- Discovery Service
- Metadata Operator
- Federation Operator

The discovery service is used to find the users organization IdP. The IdP defines an attribute release policy and releases different sets of the user's attributes to different SPs. Users are only able to agree or disagree with the whole set of attributes because the decision comes from the IdP. When the user tries to log on, to a service of a Service Provider, he gets redirected to a page where he can choose and IdP previously found by the Discovery Service. If an SP wants to provide its service to multiple federations it has to negotiate policy and technical detail with each federation operator. The federation operator manages the federation policies and introduces SPs and IdPs. Furthermore, the federations operator has to maintain and manage the information about all entities in a federation which is contained in the Metadata. A problem with this architecture is that the SP has to keep track of changes of the technical specification of various federation operators to maintain the configuration for each federation operator. A solution would be a significant federation registration, but this is indeed not possible because of technical and administrative severity and political will. This solution is also somewhat misleading for users since they have to maintain multiple credentials, select from multiple identifiers and so on [cf.Procházka, Kouřil, and Matyska, 2010].

According to ibid. Shibboleth is too restrictive, a solution with a centrally managed point of IdPs and SPs is preferred. Also, users should not have to deal with redundant accounts.

Chapter 3

Conclusion and Outlook

Your text here . . .

Acronyms

ABI application binary interface

ACL access control list

GUI graphical user interface

KISS keep it small and simple

MITM man-in-the-middle

OS operating system

UART universal asynchronous receiver/transmitter

UID unique identifier

Bibliography

Boyed, Ryan (2012). *Getting Started with OAuth 2.0 - Programming Clients for Secure Web API Authorization and Authentication*. O'Reilly Media.

Brooks, Sean, Michael Garcia, Naomi Lefkovitz, Suzanne Lightman, and Ellen Nadeau (Jan. 2017). *An Introduction to Privacy Engineering and Risk Management in Federal Systems*. NISTR 8062. NIST (National Institute of Standards and Technology).

Corre, Kevin, Oliver Barais, Gerson Sunyé, Frey Vincent, and Jean-Michel Crom (2017). *Why can't users choose their identity provider*. Available from: <https://petsymposium.org/2017/papers/issue3/paper18-2017-3-source.pdf> [May 2018].

Dasgupta, Dipankar, Roy Arunava, and Nag Abhijit (2017). *Advances in User Authentication*. Springer International Publishing AG. ISBN: 978-3-319-58808-7.

Grassi, Paul A., Michael E. Garcia, and Fenton James L. (July 2017). *Digital Identity Guidelines*. Special Publication 800-63-3. NIST (National Institute of Standards and Technology).

Inc., Google (July 2018). *Google Developers. OAuth 2.0 Playground*. Available from: <https://developers.google.com/oauthplayground/> [July 2018].

JerichoSystems (2018). *Identity Silo*. Available from: <https://www.jerichosystems.com/technology/glossaryterms/identity_silo.html> [June 2018].

Jones, M., J. Bradley, and N. Sakimura (2015). *JSON Web Token (JWT)*. Internet Engineering Task Force (IETF). Available from: <https://tools.ietf.org/html/rfc7519> [July 2018].

Lynch, Lucy (Sept. 2011). "Inside the Identity Game". In: *IEEE Internet Computing* 15.5, pp. 78–82.

Neumann, Peter G. (2013). *Principled Assuredly Trustworthy Composable Architectures*. Available from: <http://www.csl.sri.com/users/neumann/chats4.pdf> [May 2018].

Procházka, Michal, Daniel Kouřil, and Luděk Matyska (May 2010). "User centric authentication for web applications". In: *Collaborative Technologies and Systems (CTS)*. Chicago, IL, USA: IEEE. ISBN: 978-1-4244-6622-1.

Saltzer, Jerome H. and Michael D. Schroeder (1975). *The Prodection of Information in Computer Systems*. Available from: <http://www.cs.virginia.edu/~evans/cs551/saltzer/> [June 2018].

Sevilleja, Chris (Jan. 21, 2015). *The ins and outs of Token Based Authentication*. Scotch.io. Available from: <https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication> [July 2018].

Siriwardena, Prabath (Apr. 2016). *JWT, JWS and JWE for Not So Dummies! (Part 1)*. Available from: <https://medium.facilelogin.com/jwt-jws-and-jwe-for-not-so-dummies-b63310d201a3> [July 2018].

Tkalec, Tino (2015). *JSON Web Token Tutorial. An Example in Laravel and AngularJS*. Toptal. Available from: <https://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs> [July 2018].

Todorov, Dobromir (2007). *Mechanics of User Identification and Authentication - Fundamentals of Identity Managment*. Auerbach Publications. ISBN: 978-1-4200-5219-0.

Tomkins, Benjamin (2009). "Dealing With Password Fatigue". In: *Forbes*.