



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Muster des Deckblatts für Abschlussarbeiten

Masterarbeit

Entwurf vom
12. September
2023

Intrusion detection for OAuth

vorgelegt von

Florian Nehmer

Matrikelnummer 6417446

Studiengang Informatik

MIN-Fakultät

Fachbereich Informatik

eingereicht am 06.01.2023

Betreuer: Pascal Wichmann, M. Sc. Informatik

Erstgutachter: Prof. Dr.-Ing. Hannes Federrath

Zweitgutachter: Pascal Wichmann, M. Sc. Informatik.

Aufgabenstellung

OAuth [RFC6749] is a widely used authentication protocol, which is typically used between multiple actors, such as different organizations. As authentication is at the core of application security, it is specifically essential to prevent attacks on the authentication.

The tasks of this thesis are as follows: Firstly, a systematic literature study should be performed on existing properties and attacks on the OAuth protocol or its implementations. Secondly, the thesis should design protection strategies for the threats that are not sufficiently solved in existing solutions. Two options for this step are (i) the utilization of anomaly-based intrusion detection for OAuth and (ii) specification-based intrusion detection for OAuth. Thirdly, the thesis should evaluate the security of the designed architecture and compare it to other solutions.

Zusammenfassung

Für die eilige Leserin bzw. den eiligen Leser sollen auf etwa einer halben, maximal einer Seite die wichtigsten Inhalte, Erkenntnisse, Neuerungen bzw. Ergebnisse der Arbeit beschrieben werden.

Durch eine solche Zusammenfassung (im Engl. auch Abstract genannt) am Anfang der Arbeit wird die Arbeit deutlich aufgewertet. Hier sollte vermittelt werden, warum man die Arbeit lesen sollte.

Inhaltsverzeichnis

1	Introduction	6
1.1	Motivation	6
1.2	Research Question	6
1.3	Outline	6
2	Fundamental Knowledge	7
2.1	OAuth 2.0 protocol	7
2.1.1	Involved actors	7
2.1.2	Front-channel and Back-channel messages	7
2.1.3	Grant Types	8
2.1.4	OpenID	9
2.1.5	The Future: OAuth 2.1	9
2.2	Intrusion Detection	9
2.2.1	zeek IDS	9
2.3	Algorithms	9
3	Literature Study: Taxonomy of OAuth Vulnerabilities	10
3.1	OAuth Vulnerabilities	10
3.1.1	Insufficient Redirect URI Validation [Lod+20] [Wan+19]	10
3.1.2	Credential Leakage via Referer Headers	10
3.1.3	Credential Leakage via Browser History	11
3.1.4	Mix-Up Attacks	11
3.1.5	Authorization Code Injection [PPJ22]	11
3.1.6	Countermeasures	11
3.1.7	Access Token Injection	11
3.1.8	Cross Site Request Forgery	11
3.1.9	PKCE Downgrade Attacks	12
3.1.10	Access Token Leakage at the Ressource Server	12
3.1.11	Misuse of Stolen Access Tokens	12
3.1.12	Open Redirection	12
3.1.13	307 Redirect	12
3.1.14	TLS Terminating Reverse Proxies	12
3.1.15	Refresh Token Protection	12
3.1.16	Client Impersonating Resource Owner	12
3.1.17	Clickjacking	12
3.1.18	Authorization Server Redirecting to Phishing Site	12
3.1.19	Attacks on In-Browser Communication Flows	12
3.2	Countermeasures	12
3.2.1	PKCE system	12
4	Intrusion Detection: State of the Art	13

5	Algorithmic Approach	14
6	Experimental Analysis	15
7	Conclusion	16
	Literatur	17

1 | Introduction

1.1 Motivation

1.2 Research Question

1.3 Outline

2 | Fundamental Knowledge

2.1 OAuth 2.0 protocol

The Open Authorization 2.0 protocol nowadays often referred to as the *OAuth* protocol, is an authorization framework, that allows third-party applications to gain limited access to resources in a different location on behalf of the party, that owns these resources. For many users of the Internet, it is in practice the protocol behind the “*Sign in with ...*” button. The current standard, first defined in 2012 in RFC6749, is already the successor of the OAuth 1.0 standard, which was officially published in 2010 by the IETF in RFC5849 [Ham10]. In the meantime, several extensions for the protocol were published as standards and technical reports. These extensions include new functionalities for the protocol e.g. the ability to use the protocol with devices like smart TVs and printers [Den+19] or documents, which describe several security considerations when implementing the protocol in practice [Lod+20].

The current version of the OAuth protocol grew over time and nowadays has many involved actors and offers several different modes of operation.

2.1.1 Involved actors

In general, there are four involved actors in the OAuth protocol:

- **Resource owner:** The resource owner is the entity that owns or is allowed to manage protected resources. The resource owner might grant access to these resources.
- **Resource server:** The resource server is the server, where the protected resources are stored. It can accept or decline authorization tokens, which it receives from the client.
- **Client:** The client is an entity, which makes requests to get access to the protected resources, on behalf of the resource owner.
- **Authorization server:** The server, that manages access to the protected data. It issues access tokens to the client after successful authentication of the resource owner.

Depending on the protocol mode, these four actors exchange different messages in variable ways.

2.1.2 Front-channel and Back-channel messages

Regarding security considerations messages are categorized into two main categories, *front-channel* and *back-channel*. As the protocol is mostly used in the application layer using HTTP and TLS, *front-channel* means that the message is transported via the *Request-URI* [Fie99, Sec. 5.1.2] e.g. by using query parameters. An attacker, which is able to eavesdrop on network traffic would be able to read such messages. *Back-channel* means on the other hand that the message is

transported via the HTTP message body. In the case of HTTPS, this means the attacker, who eavesdrops on the network traffic is not able to read the message anymore, as it is encrypted.

2.1.3 Grant Types

The protocol flow is dependent on the protocol mode. In the case of OAuth the different protocol modes are called *Grant types*, as the modes differ on how the authorization is granted to the resource owner via the client.

write about general
OAuth flow with
graphic, see rfc

Authorization Code Grant

According to a recent study, the authorization code grant mode of OAuth is the most used protocol mode for OAuth on the internet [PPJ22, Table 1]. It is offered by more than 90% of all identity providers.

In this mode illustrated in Figure 2.1, a resource owner aiming to access protected data via a REST API utilizes a user agent, in this case a web browser, to interact with a client application that manages API requests through the user agent. The process begins as the client redirects the user agent to the authorization server through the front channel. The message contains a client ID, a redirect URL and a state. The client ID is a unique identifier of the protected resource. The redirect URI is the location the user agent is redirected to after authentication. The state can hold any string value and is most commonly used for CSRF tokens. The authorization server now asks the resource owner for authentication. If the resource owner is authenticated and is allowed to access the desired protected resources, the user agent gets redirected back using the redirect URI. This means that the message is again sent via the front channel and contains now an authorization code and the state. Using the valid authorization code and the state, the client proceeds to request an access token via the back channel at the authorization server. With the acquired access token the client follows with the request of the desired protected resource at the resource server.

- Implicit Grant
- Resource Owner Password Credentials Grant
- Client Credentials Grant
- Refresh Token Grant
- JWT Bearer Grant
- Device Code Grant
- UMA Grant
- SAML 2.0 Bearer Grant
- Token Exchange Grant

Decide on how to
handle niche grant
types

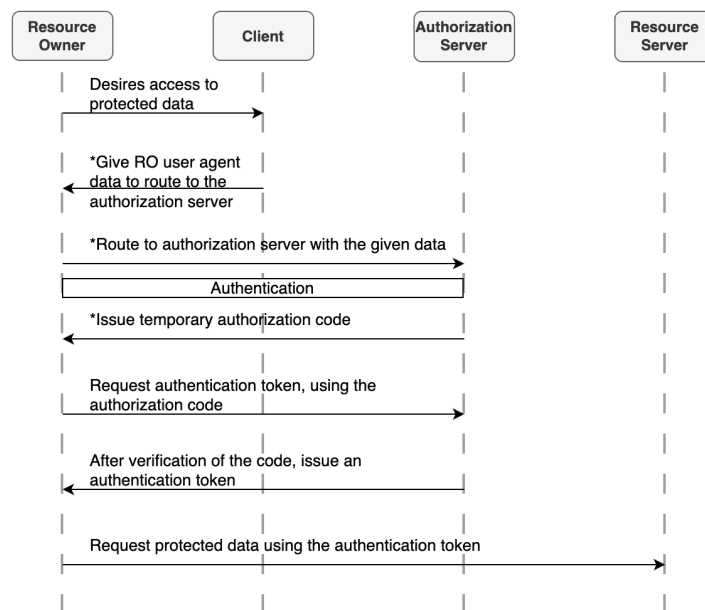


Abbildung 2.1: Authorization Code Grant flow without any extensions

2.1.4 OpenID

2.1.5 The Future: OAuth 2.1

2.2 Intrusion Detection

2.2.1 zeek IDS

2.3 Algorithms

3 | Literature Study: Taxonomy of OAuth Vulnerabilities

3.1 OAuth Vulnerabilities

3.1.1 Insufficient Redirect URI Validation [Lod+20] [Wan+19]

Authorization Servers need to whitelist redirection URLs in order to make sure, that an attacker cannot craft a hyperlink, which leads to the victim initiating an OAuth flow and sending the authorization code or token to an attacker-controlled domain. Some authorization servers may allow the usage of patterns in order to allow several domains at once. As well as the absence of any sort of whitelist mechanism even a pattern-matching functionality could lead to security problems. Among the possibility that a user is entering patterns that are too broad and allow the usage of unintended redirect URLs, the attack surface includes issues with the URL parsing implemented by the authorization server as shown by Wang et al. [Wan+19]. They presented several techniques to trick the parser into accepting unintended domain names, like using squared brackets for IPv6 parsing or the *Evil Slash Trick*, where the parser does not treat a forward slash as a path separator, while modern browsers do. Depending on the OAuth grant type in use this vulnerability leads to different possibilities to exploit it.

Authorization Code Flow

- The attacker uses techniques like phishing to make its victim open an attacker-controlled webpage, which initiates an OAuth flow with the vulnerable authorization server.
- The request is crafted with a valid client ID (which is public information), “code” as response type and a malicious redirect URI, which leads to an attacker-controlled server again.
- If the user logs in at the authorization server, the authorization code now gets transmitted to the attacker’s webpage, via the redirect URI.
- The attacker page can now use the received authorization code, to retrieve a token

Implicit Flow

3.1.2 Credential Leakage via Referer Headers

The Referer HTTP header is a potential attack surface. It can be utilized by a malicious actor to capture query parameters, which are sent via the front channel, like the state and the authorization code. The authorization code may be used to redeem an access token before the victim retrieves it and the state parameter oftentimes includes a CSRF token, which could potentially open up vulnerabilities in other parts of the application as explained by Fett et al [FKS16].

Maybe write down open redirection with the implicit flow here regarding redirect_uri check circumvention.

OAuth Client

If a client renders third-party content, like advertisements in iframes or images, before redeeming the authorization code for an access token an attacker, who places these advertisements or images can capture the code via the referer header and redeem it for an access token.

Authorization Server

At the authorization server, the state parameter could be leaked via the Referer header, when 3rd party images or advertisements are being rendered on the page. This may be an issue when the state contains a CSRF token as explained by Fett et al [FKS16].

3.1.3 Credential Leakage via Browser History

Find studies about risks of confidential data in browser history

3.1.4 Mix-Up Attacks

3.1.5 Authorization Code Injection [PPJ22]

The precondition for an authorization code injection is that an attacker has successfully stolen an authorization code. This can be accomplished in various ways for example by tricking the user into installing a malicious browser extension, using other vulnerabilities in a web app like open redirections, or abusing proxy auto-configuration files [PPJ22].

In the case, that the client is using the authorization code flow the attacker can use the stolen authorization code to fetch an access token before the client does.

3.1.6 Countermeasures

There are several ways to mitigate the risk of authorization code injection. For example, if an authorization code got used twice in the case it got stolen and the attacker, as well as the client, tried to redeem a token, every token that was redeemed with this authorization code should get invalidated. Another method is the usage of PKCE to make sure the OAuth flow is only used, between the two legitimate parties.

3.1.7 Access Token Injection

3.1.8 Cross Site Request Forgery

This type of attack, often referred to by its abbreviation “CSRF, is about the attacker executing a request in the name of the user, by tricking the user into executing requests for the attacker including all required authentication, or authorization information. The default OAuth protocol does not include protection mechanisms against this type of attack.

3.1.9 PKCE Downgrade Attacks

3.1.10 Access Token Leakage at the Ressource Server

3.1.11 Misuse of Stolen Access Tokens

3.1.12 Open Redirection

3.1.13 307 Redirect

3.1.14 TLS Terminating Reverse Proxies

3.1.15 Refresh Token Protection

3.1.16 Client Impersonating Resource Owner

3.1.17 Clickjacking

3.1.18 Authorization Server Redirecting to Phishing Site

3.1.19 Attacks on In-Browser Communication Flows

3.2 Countermeasures

3.2.1 PKCE system

As an extension to OAuth defined by RFC 7636, “Proof Key for Code Exchange” is a technique to mitigate Authorization Code Injection misuse or CSRF attacks [BA15]. Specifically, it extends the authorization code flow

4 | Intrusion Detection: State of the Art

5 | Algorithmic Approach

6 | Experimental Analysis

7 | Conclusion

Literatur

- [BA15] J Bradley und N Agarwal. *RFC 7636: Proof Key for Code Exchange by OAuth Public Clients*. 2015.
- [Den+19] William Denniss u. a. *OAuth 2.0 device authorization grant*. Techn. Ber. 2019.
- [Fie99] Roy Fielding. *Hypertext transfer protocol*. In: *RFC 2616* (1999).
- [FKS16] Daniel Fett, Ralf Küsters und Guido Schmitz. *A comprehensive formal security analysis of OAuth 2.0*. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, S. 1204–1215.
- [Ham10] Eran Hammer-Lahav. *Rfc 5849: The oauth 1.0 protocol*. 2010.
- [Lod+20] Torsten Lodderstedt u. a. *OAuth 2.0 security best current practice*. In: *IETF Web Authorization Protocol, Tech. Rep. draft-ietf-oauth-security-topics-16* (2020).
- [PPJ22] Pieter Philippaerts, Davy Preuveneers und Wouter Joosen. *OAuch: Exploring Security Compliance in the OAuth 2.0 Ecosystem*. In: *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*. 2022, S. 460–481.
- [Wan+19] Xianbo Wang u. a. *Make redirection evil again: Url parser issues in oauth*. In: *BlackHat Asia 2019* (2019).

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ggf. streichen: Ich bin damit einverstanden, dass meine Abschlussarbeit in den Bestand der Fachbereichsbibliothek eingestellt wird.

Hamburg, den 06.01.2023

Florian Nehmer

Bitte verwenden Sie hier in jedem Fall die offizielle von der Prüfungsbehörde vorgegebene Formulierung der Selbständigkeitserklärung.

Thema: Privacy Enhancing Technologies zum Schutz von Kommunikationsbeziehungen

Bearbeiter: Eva Musterfrau, Heinz Mustermann

Datum: 12. September 2023

[Muster der Literaturliste](#)

Literaturliste

Todo list

■ write about general OAuth flow with graphic, see rfc	8
■ Decide on how to handle niche grant types	8
■ Maybe write down open redirection with the implicit flow here regarding redirect_uri check circumvention	10
■ Find studies about risks of confidential data in browser history	11
■ Bitte verwenden Sie hier in jedem Fall die offizielle von der Prüfungsbehörde vorgege- bene Formulierung der Selbständigkeitserklärung.	18