Master Arbeit

Sicherheitsanalyse von OAuth 2.0 mittels Web Angriffen auf bestehende Implementierungen

Christoph Nickel

Datum: 01.12.2013

Gutachter: Prof. Dr. Jörg Schwenk
Dipl.-Ing. Vladislav Mladenov

Betreuer: M. Sc. Christian Mainka
M. Eng. Tobias Wich
Dipl.-Ing. Vladislav Mladenov

Ruhr-Universität Bochum

RUB

Lehrstuhl für Netz- und Datensicherheit
Prof. Dr. Jörg Schwenk
Homepage: www.nds.rub.de

Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für ein anderes Examen eingereichten Arbeit. Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

Ort, Datum

Unterschrift

Abstract

OAuth 2.0, ein offenes und standardisiertes Protokoll, wird für die sichere Autorisierung von Dritt-Anwendungen für die eingeschränkte Nutzung von HTTP Diensten verwendet. Es wird dabei von einer Vielzahl von Dienstanbietern, wie z.B. Google, Facebook oder Salesforce, implementiert. Da OAuth 2.0 jedoch nur TLS als einziges Sicherheitskonzept einsetzt, gab es in letzter Zeit mehrere Angriffe auf die einzelnen Implementationen bei den Dienstanbietern, welche das OAuth 2.0 Protokoll implementiert haben. Dementsprechend wird in dieser Arbeit eine Sicherheitsanalyse durchgeführt, welche allgemeine Protokolleigenschaften, das Verhalten des Protokolls auf bestimmte Ereignisse sowie die Auswirkungen von Änderungen an bestimmten Protokollparametern bei einzelnen Dienstanbietern überprüft. Darauf aufbauend wird ein Konzept sowie eine prototypische Implementierung für das automatisierte Testen des OAuth 2.0 Protokolls vorgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis v
Listingverzeichnis vi
Tabellenverzeichnis vii
Abkürzungsverzeichnis viii

1. Einleitung **1**
1.1. Gegenstandsbereich und Problemstellung 1
1.2. Zielsetzung 2

2. Grundlagen **3**
2.1. Geschichtlicher Hintergrund 3
2.1.1. Vor OAuth 3
2.1.2. OAuth 1.0 4
2.1.3. OAuth 2.0 4
2.2. Terminologie 5
2.2.1. Authentifizierung und Autorisierung 5
2.2.2. Rollen 6
2.2.3. Token 8

3. OAuth 2.0 **10**
3.1. Überblick OAuth 2.0 Framework 10
3.1.1. Zugehörige RFCs 10
3.1.2. Zugehörige aktive Internet-Drafts 11
3.2. Abstrakter Protokollfluss 13
3.2.1. Client Authentifizierung 14
3.2.2. Protokoll Endpunkte 14
3.3. Authorization Grant Types 15
3.3.1. Authorization Code Grant Type 15
3.3.2. Implicit Grant Type 19
3.3.3. Resource Owner Password Credentials Grant Type 20
3.3.4. Client Credentials Grant Type 21
3.4. Extension Grant Types 22
3.4.1. JWT Bearer Token Grant Type 22
3.4.2. SAML 2 Bearer Assertion Grant Type 25
3.5. Sicherheitsempfehlungen und -vorkehrungen 27

4. Durchführung der manuellen Sicherheitsanalyse **31**
4.1. Angriffsmodell 31
4.1.1. Ziele des Angreifers 31
4.1.2. Annahmen 32
4.1.3. Fähigkeiten des Angreifers 33

4.1.4. Verhalten des Benutzers . . . 33
4.2. Entwicklung einer Testumgebung . . . 33
4.3. Untersuchungsaspekte . . . 37
4.4. Identifizierung und Auswahl von Anbietern . . . 46
4.5. Ergebnisse der Sicherheitsanalyse . . . 47
4.6. Angriffsszenarien . . . 54
4.6.1. Angriffsszenario 1: Erhalt des Authorization Code durch Redirection URI Manipulation . . . 54
4.6.2. Angriffsszenario 2: Erhalt des Access Token durch Redirection URI Manipulation 56

5. Konzept zur Durchführung von automatisierten Penetrationstests 58
5.1. Schwierigkeiten in Bezug auf das automatisierte Testen . . . 58
5.2. Konzept . . . 61
5.3. Prototypische Implementierung . . . 62

6. Reflexion der Ergebnisse der Arbeit 66
6.1. Verwandte Arbeiten . . . 66
6.2. Fazit . . . 67
6.3. Ausblick . . . 67

Literaturverzeichnis 69

A. Ergebnisse der manuellen Sicherheitsanalyse 74

Abbildungsverzeichnis

2.1. Zusammenhänge zwischen den Rollen im OAuth Protokoll 6

3.1. Abstrakter Protokollfluss 13
3.2. Authorization Code Grant 16
3.3. OAuth Consent 17
3.4. Implicit Grant Type 20
3.5. Resource Owner Password Credentials Grant Type 21
3.6. Client Credentials Grant Type 22
3.7. JWT Bearer Token Grant Type 23
3.8. SAML 2 Bearer Assertion Grant Type 25

4.1. Aktivitätsdiagramm der Durchführung der manuellen Sicherheitsanalyse 31
4.2. Google OAuth 2.0 Playground 34
4.3. Test Client Anwendung - Authorization Request 35
4.4. Test Client Anwendung - Authorization Response und Access Token Request 36
4.5. Test Client Anwendung - Access Token Response und Resource Request 37
4.6. Erhalt des Authorization Code durch Redirection URI Manipulation beim Authorization Code Grant Type 55
4.7. Redirection URI Manipulation beim Implicit Grant 57

5.1. Verhalten des Anbieters Cheddar bei Änderung des Redirection URI Parameters 59
5.2. Verhalten des Anbieters Box bei Änderung des Redirection URI Parameters 59
5.3. Verhalten des Anbieters Fourquare bei Änderung des Redirection URI Parameters 60
5.4. Verhalten des Anbieters Dailymotion bei Änderung des Redirection URI Parameters . . 60
5.5. OAuth Consent 62
5.6. Selenium Architektur 63
5.7. Architektur der prototyischen Implementierung 64

Listingverzeichnis

3.1. Authorization Request beim Authorization Code Grant 15
3.2. Authorization Response beim Authorization Code Grant 17
3.3. Access Token Request beim Authorization Code Grant Type 18
3.4. Access Token Response beim Authorization Code Grant Type 18
3.5. Access Token Response beim Implicit Grant Type 20
3.6. Access Token Request beim Client Credentials Grant Type 22
3.7. Access Token Request beim JWT Bearer Token Grant Type 23
3.8. JWT Header 24
3.9. JWT Claim Set 24
3.10. Access Token Request beim SAML 2 Bearer Assertion Grant Type 26
3.11. SAML 2 Assertion 27

5.1. Beispielhafter Testcase 65

Tabellenverzeichnis

4.1. Untersuchungsaspekt 1 - Grant Types 38
4.2. Untersuchungsaspekt 2 - Auslegung des Standards durch den Dienstanbieter 39
4.3. Untersuchungsaspekt 3 - Absicherung des Kommunikationskanals durch den Einsatz von TLS 40
4.4. Untersuchungsaspekt 4 - Änderung der Redirection URI 41
4.5. Untersuchungsaspekt 5 - Änderung des Response Types 42
4.6. Untersuchungsaspekt 6 - Komplexität der Client ID 42
4.7. Untersuchungsaspekt 7 - Nutzung und Änderung des Scopes 43
4.8. Untersuchungsaspekt 8 - Einsatz des State Parameters 43
4.9. Untersuchungsaspekt 9 - Fehlermeldungen 44
4.10. Untersuchungsaspekt 10 - Erneute Autorisierungsanfrage 44
4.11. Untersuchungsaspekt 11 - Bindung des Authorization Codes 45
4.12. Untersuchungsaspekt 12 - Wiederholtes Übersenden des Authorization Codes 45
4.13. Untersuchungsaspekt 13 - Bindung des Access Token an den Client 46
4.14. Untersuchungsaspekt 14 - Widerrufung von Tokens 46
4.15. Ausgewählte Dienstanbieter 47
4.16. Ergebnissübersicht der manuellen Analyse 48

Abkürzungsverzeichnis

API	Application Programming Interface
CSRF	Cross Site Request Forgery
CSS	Cascading Style Sheets
HTTP	Hypertext Transfer Protocol
HTTS	Hypertext Transfer Protocol Secure
ID	Identifikator
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
JWT	JSON Web Token
MAC	Message Authentication Code
MIME	Multipurpose Internet Mail Extensions
PIN	Personal identification number
RC	Remote Control
RFC	Request for Comments
SAML	Security Assertion Markup Language
SDK	Software Development Kit
SR	Sicherheitsempfehlung und -vorkehrung
SSL	Secure Sockets Layer
TLD	Top-Level Domain
TLS	Transport Layer Security
UA	Untersuchungsaspekt
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
XML	Extensible Markup Language

1. Einleitung

1.1. Gegenstandsbereich und Problemstellung

Das OAuth 1.0 Protokoll wurde 2007 durch eine IETF Community entwickelt, um eine Methode für den delegierten Zugriff auf geschützte Ressourcen zu bieten. Es löste mehrere ähnliche proprietäre Protokolle großer Internetfirmen, wie ClientLogin und AuthSub von Google oder Yahoo!'s BBAuth ab, da diese das Problem aufwiesen, dass Entwickler ihre Anwendung an die einzelnen APIs anpassen mussten [1].

Das OAuth Protokoll als ein offener Standard bietet einem Client die Möglichkeit im Namen des Besitzers, dem sogenannten Resource Owner, ohne Kenntnis seines Passwortes auf eine geschützte Ressource zu zugreifen [2]. Dieses kann z.B. eine Anwendung zur Verwaltung von Geburtstagen sein, welche auf die eingetragenen Geburtstage im Google Kalender zugreifen möchte. In dem Fall, dass der Client noch keine Berechtigungen besitzt, um auf eine geschützte Ressource zu zugreifen, wird der User Agent an einen Authentifikation Server seiner Wahl weitergeleitet [3]. Dort authentifiziert sich der Nutzer und bestätigt den Zugriff. Als Antwort wird ein sogenannter „Access Token" für den Client erzeugt. Mittels diesem speziellen Token erhält nun der Client limitierten Zugriff (bezogen auf Zeit und Umfang, dem sogenannten „scope") auf die Ressourcen des Nutzers bei einem anderen Anbieter [2].

Das OAuth 1.0 Protokoll setzt voraus, dass Zugriffsanfragen signiert werden, damit die Identität und Authentizität des Client nachvollzogen werden kann. Dieses Vorgehen stellt jedoch viele Entwickler vor großen Herausforderungen, da Kenntnisse über Kryptographie vorausgesetzt werden. Durch die starke Verbreitung von SSL/TLS in den Jahren nach der Entwicklung von OAuth 1.0, sahen viele Entwickler die Verwendung der Signatur zur Zugriffsabsicherung als obsolet an. Dieses führte unter anderem zu einer Neukonzipierung des OAuth Protokolls (OAuth 2.0), welches auf TLS zur Absicherung setzt und nicht kompatibel zur ersten OAuth Version ist [1].

Jedoch stimmten nicht alle Autoren des neuen Standards diesem Vorgehen eindeutig zu [4], so dass sich im Jahr 2012 Eran Hammer - Autor der OAuth 1.0 Version - entschied die OAuth Working Group zu verlassen. In seinem Blog stellt er hervor, dass aus seiner Sicht die zweite OAuth Version gegenüber der ersten komplexer, weniger interoperabel, weniger nützlich, unvollständiger und vor allem weniger sicher ist [5]. Darüber hinaus weist er darauf hin, dass zur sicheren Umsetzung des OAuth 2.0 Protokolls ein tiefes Sicherheitsverständnis bei den Entwicklern vorhanden sein muss, wie es bei vielen Implementierungen nicht der Fall gewesen war: „To be clear, OAuth 2.0 at the hand of a developer with deep understanding of web security will likely result is a secure implementation. However, at the hands of most developers – as has been the experience from the past two years – 2.0 is likely to produce insecure implementations" [5].

Letztendlich gehen mehrere in diesem Jahre publizierte Angriffe auf das OAuth 2.0 Protokoll [6] [7] [8] darauf ein, dass die Implementierung des Protokolls bei den Anbietern Schwachstellen aufweisen.

1.2. Zielsetzung

Ein Ziel der Masterarbeit ist es mittels einer Literaturrecherche einen Überblick über das Themengebiet „OAuth-basierte Autorisierung und Authentifizierung“ zu geben. Daher wird zu Beginn der Arbeit in einem kurzen Abschnitt auf den geschichtlichen Hintergrund von OAuth sowie auf die unterschiedlichen Versionen und deren Veränderung eingegangen. Anschließend werden für die Arbeit wichtige Begriffe und Konzepte erläutert. Im folgenden Kapitel wird detailliert auf das OAuth 2.0 Protokoll [9] eingegangen. Dazu wird zunächst ein Überblick über die zum OAuth Framwork zugehörigen RFCs und aktiven Internet Drafts gegeben. Mittels eines abstrakten Protokollflusses wird danach das zugrunde liegende Konzept erläutert. Darauf aufbauend werden die unterschiedlichen Protokollflüsse, welche vielfältige Anwendungsszenarien darstellen, vorgestellt. Diese sind notwendig, da verschiedene Clients (z. B. Webanwendung oder native Anwendung) andere Anforderungen und Methoden besitzen [10]. Abgeschlossen wird dieses Kapitel mit der Darlegung von Sicherheitsempfehlungen und -vorkehrungen, welche innerhalb des OAuth 2.0 Standards [9] und dem „OAuth Threat Model“ [11] gegeben werden bzw. aufgeführt sind.

Ein weiteres Ziel der Masterarbeit ist es, die Ergebnisse einer Sicherheitsanalyse des OAuth 2.0 Protokolls, welche anhand von Tests auf führenden Systemen (z. B. Salesforce, Facebook, Google u.a.) durchgeführt wurde, zu präsentieren. Im Interesse der Sicherheitsanalyse steht dabei, wie das OAuth Protokoll von den einzelnen Dienstanbietern implementiert wurde und welche Schwachstellen sich dabei ergeben. Diese können einerseits Dienstanbieter spezifisch sein oder allgemein für das OAuth Protokoll gelten. Innerhalb des vierten Kapitels wird zunächst ein Angriffsmodell spezifiziert. Es wird dabei angenommen, dass die Implementierungen des Protokolls nicht vollständig Standard-konform durchgeführt sowie die Sicherheitsvorgaben des „OAuth Threat Model“ [11] nicht umgesetzt wurden. Als Angreifer wird ein Webangreifer definiert, der lediglich die Fähigkeit besitzt bösartige Webseiten aufzusetzen sowie manipulierte Links zu verbreiten, jedoch keinen Netzwerkverkehr belauschen oder manipulieren kann. Zur Durchführung der Sicherheitsanalyse wurde eine Testumgebung entwickelt, sowie ein Katalog an Untersuchungsaspekten erarbeitet. Weiterhin wurde auf der Grundlage der Ergebnisse der manuellen Sicherheitsanalyse ein Konzept sowie eine prototypische Implementierung entworfen, welches das automatisierte Penetration-Testen ermöglicht.

2. Grundlagen

Im folgenden Kapitel wird zunächst wird auf den geschichtlichen Hintergrund von OAuth 2.0 eingegangen. Dabei wird zuerst die Situation vor der Entwicklung von OAuth dargestellt, um den Zweck und Nutzen von OAuth aufzuzeigen. Anschließend wird dargelegt, welche Problematik sich durch die erste OAuth Version ergeben hat und wie es zur Neukonzipierung des OAuth Protokolls gekommen ist. Im nachfolgenden Teil des Kapitels werden die für das Verständnis des Themas und der Arbeit relevanten Termini und Zusammenhänge erläutert.

2.1. Geschichtlicher Hintergrund

2.1.1. Vor OAuth

Im Zuge der Entwicklung des Internets zum Web 2.0 hat sich der Schwerpunkt von der reinen Informationsvermittlung auf den Austausch von Informationen und deren Vernetzung verlagert [12]. Das Web wird nun als eine Plattform genutzt. Dieses hat zur Folge, dass die Grenze zwischen Desktop, Web und anderen Geräten zunehmend verschwimmen. Weiterhin besteht die Möglichkeit verschiedene Inhalte mit einander zu verknüpfen, um so neue Anwendungen zu genieren. Mit Hilfe von Schnittstellen (den APIs) können auf die bestehenden Datenbestände von anderen Anwendungen zugriffen werden [13].

Möchte nun ein Nutzer eine Anwendung nutzen, um z.B. seine Daten, welche bei einem anderen Anbieter gespeichert sind, zu verarbeiten, so musste er der Anwendung seinen Benutzernamen und das dazu gehörige Passwort übermitteln. Die Anwendung war anschließend in der Lage die vom Nutzer gewünschten Daten abzurufen. Diese Vorgehensweise kann jedoch negative Nebeneffekte für den Nutzer besitzen [1] [14]:

- *Speicherung der Passwörter:* Häufig wurden die Passwörter durch den Anbieter gespeichert, um eine erneute Nutzung ohne vorherige Autorisierung zu ermöglichen. Dieses Vorgehen ist jedoch nicht von jedem Anwender gewünscht. Weiterhin kann eine Kompromittierung der Anwendung im schlimmsten Fall zu einer Offenlegung des Passwortes führen.

- *Zugriffsrechte:* Wenn der Benutzer seinen Benutzernamen und Passwort übermittelt, kann die Anwendungen nicht nur auf die vom Benutzer gewünschten Daten zu greifen. Sondern die Anwendung besitzt die vollen Zugriffsrechte und kann so auf die gesamten Daten des Anwenders zu greifen.

- *Rechteentzug:* Für den Benutzer gibt es keine einfache Lösung der Anwendung die Erlaubnis für den Zugriff zu widerrufen. Der Anwender kann lediglich sein Passwort ändern, um der Anwendung ihre Rechte zu entziehen.

Die genannte Problematik der Autorisierung von Drittanwendungen wurde durch einige der großen Anbieter des Internets erkannt, so dass sie eigene proprietäre Protokolle entwarfen. Diese waren unter anderem Google's AuthSub [15] und Yahoo's BBAuth [16]. Jedoch wiesen diese Protokolle das Problem auf, dass sie unter einander nicht kompatibel waren. Dementsprechend mussten Webseitenentwickler mehrere webbasierte Autorisierungslösungen erlernen und implementieren [1]. Auf Grund der erhöhten Kosten für die Entwickler baute sich eine Hemmschwelle bezüglich der Umsetzung der Protokolle auf:

> *We want something like Flickr Auth / Google AuthSub / Yahoo! BBAuth, but published as an open standard, with common server and client libraries, etc. " (Blaine Cook, 05.04.2007* [17]*)*

2.1.2. OAuth 1.0

Das OAuth 1.0 Protokoll wurde 2007 durch eine Community von Entwicklern entworfen, um eine Methode für den delegierten Zugriff auf geschützte Ressourcen zu bieten und andere proprietäre Protokolle abzulösen [17]. OAuth sieht sich selbst als ein offener Standard, welcher auf Best Practices Erfahrungen, die durch kleine und große Anbieter getragen wird, beruht [18].

Im April 2009 kam es zu einer Veröffentlichung eines Session Fixation Angriffes für die Core 1.0 Spezifikation [19]. Dabei war es einem Angreifer mittels Social Engineering möglich eine zuvor von ihm begonnene Autorisierungsanfrage durch ein Opfer fortführen zulassen. Da im Vorfeld ein Request Token festgelegt wurde, konnte der Angreifer, nachdem das Opfer Zugriff auf seine Daten gewährt hatte, diesen gespeicherten Request Token nutzen, um auf die Daten des Opfers zu zugreifen. Aufgrund dieses Angriffes wurde die OAuth Core 1.0 Revision A Spezifikation publiziert [20].

Ein Jahr später veröffentlichte die IETF das informationelle Dokument „RFC 5849 The OAuth 1.0 Protocol" als Ablösung für die OAuth 1.0 Revision A Spezifikation. Es enthält weitere kleinere Korrekturen und Verbesserungen, welche durch die OAuth Working Group heraus gearbeitet wurden [2].

2.1.3. OAuth 2.0

Das OAuth 1.0 Protokoll setzt voraus, dass Zugriffsanfragen signiert werden, damit die Identität und Authentizität des Clients nachvollzogen werden kann [2, sec. 3.4]. Dieses Vorgehen stellte jedoch viele Entwickler vor große Herausforderungen, da Kenntnisse über Kryptographie vorausgesetzt wurden. Durch die starke Verbreitung von SSL/TLS in den Jahren nach der Entwicklung von OAuth 1.0, sahen viele Entwickler die Verwendung der Signatur zur Zugriffsabsicherung als obsolet an [1].

Weiterhin sollte OAuth verschiedene Gerätetypen durch unterschiedliche Protokollabläufe unterstützen. Jedoch wurden während der Entwicklung der Spezifikation die Protokollabläufe zu einem zusammen gefasst. Dieses grenzte nach der Meinung von *Hammer* die Möglichkeiten des Protokolls stark ein,

so dass eine positive Benutzererfahrung nur für Web Anwendungen aber nicht für Desktop oder Mobile Anwendungen gegeben war [21].

Diese beiden Probleme sowie die unzureichende Skalierbarkeit, welche auf Grund von übermäßiger Speicherung von teilweise nicht benötigten Informationen entsteht [21], führten unter anderem zu einer Neukonzipierung des OAuth Protokolls (OAuth 2.0. Das OAuth 2.0 Protokoll setzt dabei auf TLS zur Absicherung des Kommunikationskanals und ist nicht kompatibel zur ersten OAuth Version [1]. Nach 31 Entwürfen kam es im Oktober 2012 zur Publikation des „RFC 6749 The OAuth 2.0 Authorization Framework“ durch die IETF [9].

2.2. Terminologie

2.2.1. Authentifizierung und Autorisierung

Authentifizierung

Authentifizierung bezeichnet den Prozess der Verifizierung einer vorgegebenen Identität durch eine andere Entität (in diesem Fall ein Authentifizierungssystem) [1]. Der Authentifizierungsprozess gliedert sich in zwei Schritte [22] [23] [24]. Zunächst wird der Benutzer identifiziert. Dazu muss dem System ein Identifikator vorgelegt werden, mit dem dieser den Benutzer eindeutig identifizieren kann. Dabei ist es möglich, dass dieser Identifikator einem Namen entspricht oder abstrakt gewählt ist. Anschließend wird die Übereinstimmung zwischen dem Identifikator und der gespeicherten Entität verifiziert. Hierfür muss der Benutzer dem System eines (oder mehrere) der folgenden Authentifikationsinformationen übergeben:

- *Etwas, das der Benutzer weiß*: z.B. ein Passwort, eine PIN oder beliebige Zeichenkette.
- *Etwas, das der Benutzer besitzt*: z.B. Übersendung eines Tokens oder Vorlage einer Smartcard, wobei der Benutzer im Besitz dieses physischen Gegenstandes sein muss.
- *Etwas, das der Benutzer ist*: Bezieht sich meist auf biometrische Merkmale eines Menschen. Diese können z.B. der Fingerabdruck, die Iris oder die Stimme sein.

Stimmen die übergebenen und im System gespeicherten Authentifikationsinformationen[1] überein, so ist der Benutzer authentifiziert.

Autorisierung

Autorisierung bezeichnet den Prozess der Zuweisung von Rechten an einen Nutzer eine bestimmte Aktion auszuführen [22] sowie der Verifikation der Rechte bei Ausführung der Aktion [1]. Dementsprechend ist das Ziel der Autorisierung den Zugriff eines Nutzer nur auf die Ressourcen und Dienste, für welche er berechtigt ist, zu beschränken [23]. Dieses setzt voraus, dass sich der Nutzer gegenüber dem System authentifiziert hat. In Bezug auf Web Anwendungen wird beim Einloggen die Identität des Nutzers bestimmt und durch den Einsatz von z.B. Zugriffskontrolllisten gewährleistet, dass der Nutzer nur auf autorisierte Ressourcen zugreifen kann [1].

[1] als Klartext, als Hashwert und in einem anderen sicheren Format

Delegierte Autorisierung

Delegierte Autorisierung beschreibt einen Vorgang bei dem Zugriffsrechte auf eine andere Person oder Anwendung übertragen werden, um im Auftrag des Besitzers auf dessen Ressourcen zu zugreifen [1]. Bei den Zugriffsrechten für die andere Entität handelt es sich meist um eine Teilmenge der Zugriffsrechte des Besitzers. Dieses bedeutet, dass die andere Entität nur über die gleichen oder weniger Zugriffsrechte als der Besitzer verfügt.

Credentials

Im Allgemeinen sind Crendentials Repräsentationen der Beziehung zwischen einem Identifikator und einem im Zusammenhang stehenden Nachweis. In Bezug auf die Authentifizierung handelt es um die Beziehung zwischen einem Identifikator und einer Authentifikationsinformation und kann genutzt werden, um eine vorgegebene Identität für den Zugriff auf ein System zu verifizieren. Hinsichtlich einer Autorisierung handelt es sich bei Credentials um die Beziehung zwischen einem Identifikator und einer Zugriffsautorisation und kann zur Verifikation eines Zugriffsversuches verwendet werden [22].

2.2.2. Rollen

Innerhalb des OAuth Protokolls gibt es vier verschiedene Hauptakteure und Funktionen, welche in Rollen zusammengefasst werden [9][10][1]. Die Zusammenhänge zwischen den einzelnen Rollen werden in Abbildung 2.1 verdeutlicht. Es ist jedoch nicht festgelegt, dass eine Entität nur eine Rolle einnehmen kann. Dieses begründet sich dadurch, dass der Authorization Server und der Resource Server der gleiche Server sein können [9, sec. 1.1].

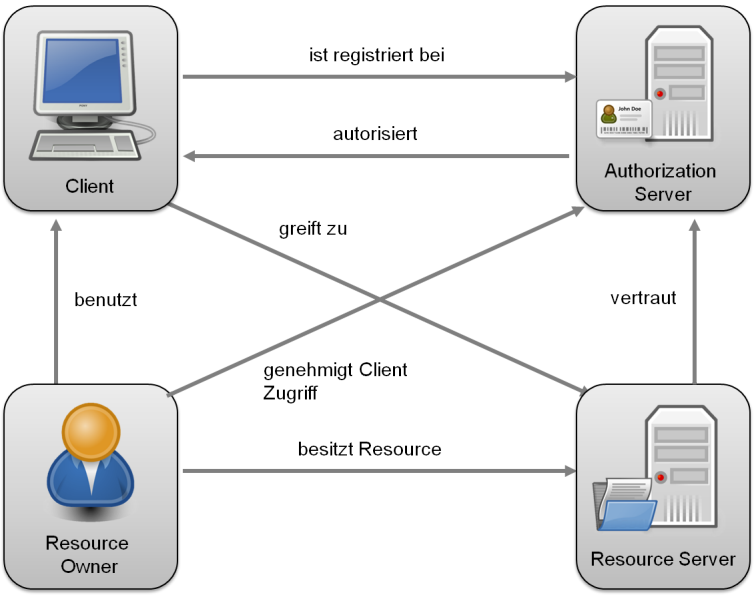

Abbildung 2.1.: Zusammenhänge zwischen den Rollen im OAuth Protokoll [25]

- *Resource Owner*: Der Resource Owner ist hauptsächlich ein Endanwender, welcher mittels eines Clients auf Ressourcen auf dem Resource Server zugreifen möchte. In diesem Zusammenhang kann er Rechte für den Zugriff des Clients auf die zu schützenden Ressourcen vergeben. Bei der zu schützenden Ressource kann es sich um Daten (Bilddateien, Dokumente oder andere Dateien), Dienste (einen Eintrag posten, ein Datei hochladen) oder andere Ressourcen, welche Zugriffsbeschränkungen unterliegen, handeln [26].

- *Resource Server*: Auf diesem Server sind die zu schützenden Ressourcen des Resource Owners hinterlegt. Der Zugriff darauf wird durch den Resource Server nur freigeben, wenn er einen gültigen Access Token erhält (siehe Abschnitt 2.2.3). Dieser Token repräsentiert die delegierte Autorisierung des Resource Owners.

- *Authorization Server*: Ein Server, welcher nach Autorisierung durch den Resource Owner einen Access Token für den Client ausstellt. Zuvor wird der Resource Owner durch den Authorization Server authentifiziert.

- *Client*: Eine Anwendung, welche durch den Resource Owner autorisiert auf dessen Ressourcen zugreifen kann. Der Begriff Client ist unabhängig von seiner Implementierung zu verstehen. Ein Client kann eine Anwendung sein, welche auf einem beliebigen System ausgeführt wird. Innerhalb der OAuth Spezifikation werden zwei Client Typen (vertraulich und öffentlich) aufgrund ihrer Authentifizierungsfähigkeit unterschieden. Im Gegensatz zum öffentlichen Client ist der vertrauliche Client fähig sich sicher gegenüber dem Authentication Server zu authentifizieren. Dieses beinhaltet auch die Fähigkeit Authentifizierungsinformationen vertraulich zu behalten. Da Clients aus verschiedenen Komponenten bestehen und diese unterschiedliche Sicherheitsanforderungen haben könne, werden folgende Client Profile definiert:
  - *Serverseitige Webanwendung*: Dieser vertrauliche Client läuft auf einem Webserver und benutzt serverseitige Programmiersprachen, um auf die API des Resource Servers zu zugreifen. Die Anwendung kann durch den Resource Owner mittels eines beliebigen User-Agent erreicht werden. Authentifizierungsinformationen des Clients (zur Authentifizierung gegenüber dem Authorization Server) und der Access Token[2] (für den Zugriff auf die Ressourcen beim Resource Server) werden auf dem Webserver gespeichert. Hierauf besitzt der Resource Owner keinen Zugriff.
  - *Clientseitige Anwendung, welche innerhalb des Webbrowser läuft*: Dieser öffentliche Client läuft innerhalb des Webbrowsers, d.h. eines User Agents, des Resource Owners. Der Anwendungscode kann dabei innerhalb einer Webseite mittels JavaScript, als Browser Erweiterung oder als Plug-in wie z.B. Flash ausgeliefert werden. Protokolldaten sowie Credentials sind daher für einen Benutzer der Anwendung leicht zugänglich.
  - *Native Anwendungen (Desktop bzw. Mobile)*: Dieser öffentliche Client ist auf dem System des Resource Owners installiert. Da der Client als native Anwendung ausgeführt wird, kann es

[2] Begriff wird im nächsten Abschnitt erläutert

sein, dass ihr nicht die gesamten Funktionalitäten eines Webbrowsers zur Verfügung stehen. Auch bei diesem Client Typ sind die Protokolldaten sowie Credentials durch den Resource Owner zugänglich.

2.2.3. Token

Access Token

Um auf geschützte Daten und Dienste auf dem Ressource Server zu zugreifen, muss ein Access Token vom Client als Repräsentation der Autorisierung übermittelt werden. Es ist zum einen möglich den Access Token bei einer Anfrage innerhalb des HTTP Authorization Header, als URL Parameter, sowie als Form-Encoded Body Parameter zu übersenden. Der präferierte Weg sollte dabei die Übermittlung innerhalb des HTTP Headers sein, da der Header in den wenigstens Fällen durch Proxys geloggt, niemals gecached und durch den Browser nicht in der History gespeichert wird [1]. In alle Fällen sollte der Kommunikationsweg mittels HTTPS verschlüsselt sein, weil der OAuth 2.0 Standard TLS als einziges Sicherheitskonzept für die Tokenübermittlung vorsieht [10].

Da der Token selber als ein Geheimnis (siehe Abschnitt 2.2.1) verwendet wird, wird in diesem Zusammenhang der Access Token auch als Bearer Token bezeichnet [21]. Das bedeutet, dass jeder der im Besitz des Tokens ist, diesen ohne weiteren Besitz und/oder Wissen von kryptographische Mitteln verwenden kann, um auf die entsprechenden Ressourcen zu zugreifen [27]. Insgesamt wird daher die Sicherheit des Tokens und somit auch des OAuth 2.0 Protokolls durch die Sicherheit von TLS bestimmt.

Innerhalb der RFC Spezifikation werden das Format, Struktur sowie weitere Eigenschaften für die Verwendung (z.B. kryptografische Eigenschaften) von Access Tokens nicht festgelegt, sondern darauf verwiesen, dass diese durch die Sicherheitsanforderungen des Resource Servers bestimmt werden müssen [9, sec. 1.4]. Jedoch lässt sich mittels des „scope“ Parameters die mit dem Access Token verbundenen Berechtigungen festlegen. Dementsprechend kann der Client den Parameter nutzen, um die seinerseits erwünschten Berechtigungen anzufragen. Andersherum kann der „scope“ Parameter durch den Authorization Server genutzt werden, um den Client zu informieren, welche Berechtigungen ihm gewährt wurden. Falls dem Client aufgrund von Policy Richtlinien bzw. durch Anweisungen des Resource Owners nicht alle erwünschten Berechtigungen gewährt werden, muss dem Client dieses mit dem Parameter mitgeteilt werden. Dabei werden die durch den Authorization Server festgelegten Berechtigungselemete innerhalb des „scope“ Parameters als eine Liste von Strings, welche durch Leerzeichen getrennt werden, angegeben [9, sec. 1.4].

Die Zugriffsdauer des Clients auf die Ressourcen wird durch die Gültigkeitsdauer des Access Tokens bestimmt. Diese wird häufig durch den Authorization Server bei der Übermittlung des Access Tokens mit gesendet. Dadurch wird dem Client im Vorfeld ermöglicht zu überprüfen, ob der Access Token noch gültig ist. Die Gültigkeitsdauer kann sehr unterschiedlich sein: sie kann von wenigen Minuten, über die Gültigkeit der Benutzersitzung bis zur Widerrufung der Berechtigungen gehen. Unter bestimmten Bedingungen[3] kann ein Access Token mittels eines Refresh Token erneuert werden [1].

[3] Wenn der Authorization Server die Refresh Token Funktion implementiert hat und der Client einen Refresh Token besitzt

Refresh Token

Ein Refresh Token kann dazu verwendet werden beim Authorization Server einen neuen Access Token anzufragen, falls der alte Access Token beispielsweise abgelaufen oder ungültig geworden ist. Der Refresh Token wird wie der Access Token nach der Autorisierung des Resource Owners vom Authorization Server an den Client gesendet. Dadurch korrespondieren Access Token und Refresh Token im Aufbau und in der Verwendung. Der Refresh Token repräsentiert ebenfalls die Autorisierung des Resource Owners. Dementsprechend muss für die Neuanfrage eines Access Tokens keine erneute Autorisierung des Resource Owners eingeholt werden [9, sec. 1.5].

Der Einsatz von Access Token und Refresh Token besitzt den Vorteil, dass die Lebensdauer des Access Tokens gering (wenige Minuten) gehalten werden kann und somit die Sicherheit des Protokolls erhöht wird. Dieses lässt sich durch folgendes Szenario begründen: Unter der Bedingung, dass der Resource Server nur bei der ersten Anfrage die Autorisierung überprüft, würde ein Rechteentzug keine Folgen haben. Ein Zugriff auf Daten und Dienste beim Ressource Server wäre für den Client weiterhin möglich. Da jedoch die Lebenszeit des Access Token wenige Minuten beträgt, würde ein Erlangen des Access Token durch einen Angreifer keine weitreichenden Folgen haben [1].

3. OAuth 2.0

Innerhalb dieses Kapitels wird die Funktionsweise des OAuth 2.0 Frameworks nach dem RFC 6749 [9] detailliert beschrieben. Zunächst wird ein abstrakter Protokollfluss dargestellt, um anschließend die unterschiedlichen Protokollflüsse, welche an die einzelnen Client Typen sowie unterschiedlichen Rahmenbedingungen angepasst sind, zu erläutern. Dabei ist anzumerken, dass innerhalb dieser Arbeit die Bezeichnung „OAuth (2.0) Spezifikation“ als Synonym für den „RFC 6749 The OAuth 2.0 Authorization Framework“ [9] verwendet wird.

3.1. Überblick OAuth 2.0 Framework

Neben der Kernspezifikation, dem RFC 6749 - The OAuth 2.0 Authorization Framework [9], werden weitere RFCs und aktive Internet-Drafts zur Workgroup „Web Authorization Protocol (oauth) “der IETF [28] zugeordnet. Diese erweitern teilweise den Umfang der OAuth Spezifikation und werden daher nachfolgend kurz beschrieben.

3.1.1. Zugehörige RFCs

RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage

Innerhalb des RFC 6750 [27] wird auf die Besonderheiten im Zusammenhang mit der Nutzung von Bearer Token bei HTTP Anfragen auf OAuth 2.0 geschützte Ressourcen eingegangen. Es wird noch einmal verdeutlicht, dass der alleinige Besitz eines Bearer Token ausreicht um auf die geschützten Ressourcen zu zugreifen. Daher ist die Nutzung von TLS zur Absicherung des Kommunikationskanals unerlässlich. Weiterhin werden in dem RFC verschiedene Methoden zur Übersendung des Bearer Tokens innerhalb des Resource Requests aufgezeigt, sowie Sicherheitsbedenken in Bezug auf die Nutzung von Bearer Token erläutert.

RFC 6755 - An IETF URN Sub-Namespace for OAuth

URIs werden innerhalb des OAuth 2.0 Frameworks häufig dafür eingesetzt Erweiterungen und andere relevante Kontexte zu identifizieren. Daher hat der RFC 6755 [29] einen IETF URN Subnamespace [30] registriert. Dieser Subnamespace lautet *urn:ietf:params:oauth* und ermöglicht OAuth relevante Parameter an einer gemeinsamen Stelle zu sammeln.

RFC 6819 - OAuth 2.0 Threat Model and Security Considerations

Der RFC 5819 [11] erweitert die innerhalb der OAuth 2.0 Spezifikation dargelegten Sicherheitsbedenken

durch ein umfassendes Bedrohungsmodell. Dazu dokumentiert es welche Sicherheitsfeatures die OAuth 2.0 Spezifikation bietet und wie diese Angriffe abwehren bzw. verhindern sollen. Anschließend werden Bedrohungen und Risiken inklusive möglicher Angriffe auf das OAuth Protokoll und die damit im Zusammenhang stehenden Entitäten (z.B. Authorization Server) aufgezeigt. Zu jeder Bedrohung werden entsprechende Gegenmaßnahmen genannt.

RFC 7009 - OAuth 2.0 Token Revocation

Der RFC 7009 [31] erweitert die OAuth 2.0 Spezifikation dahingehend, dass ein Mechanismus vorgestellt wird, mit der es möglich ist, Access Token bzw. Refresh Token zu widerrufen. Dazu erhält der Authorization Server einen zusätzlichen Aufhebungsendpunkt, dem sogenannten Revocation Endpoint. Die Notwendigkeit eines solchen Mechanismus begründet sich dadurch, dass die Gültigkeitsdauer des Zugriffes innerhalb der OAuth 2.0 Spezifikation lediglich an die Lebensdauer des Access Tokens gebunden war. Diese kann jedoch beliebig durch den Authorization Server festgelegt werden. Loggt sich ein Endbenutzer in der Zwischenzeit aus, ändert seine Identität oder Deinstalliert die entsprechende Anwendung, so bleibt der Access Token weiterhin gültig. Mit Hilfe eines Widerrufungsmechanismus ist es in diesen Fällen nun gegeben, dass der Access Token sowie die damit verbundene Rechte für ungültig erklärt werden können.

3.1.2. Zugehörige aktive Internet-Drafts

Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants (Draft 12)

Innerhalb der OAuth 2.0 Spezifikation wird darauf verwiesen, dass auch andere Methoden der Client Authentifizierung bzw. andere Grant Types, als die beschriebenen, verwendet werden können. Basierend auf dieser Gegebenheit stellt die Spezifikation „Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants“ [32] ein abstraktes Framework zur Nutzung von Assertions bereit. Unter einer Assertion wird in diesem Zusammenhang eine Sammlung von Informationen zum sicheren Austausch von Identitäts- und Sicherheitsinformationen verstanden. Zu einer Assertion gehören auch Informationen über den Aussteller, das Ausstellungsdatum, unter welchen Bedingungen die Assertion gültig ist und im welchen Kontext sie genutzt werden kann. Da es sich bei der Spezifikation lediglich um ein Framework handelt, werden in diesem Entwurf nur abstrakte Nachrichtenflüsse vorgestellt. Für konkrete Implementationen muss die Spezifikation durch weitere Spezifikationen ergänzt werden.

OAuth 2.0 Dynamic Client Registration Protocol (Draft 14)

Der Client muss vor der Ausführung des OAuth 2.0 Protokolls beim Authorization Server zur Herstellung einer Vertrauensbeziehung und zur Festlegung von Client Eigenschaften (z.B. Redirection URI oder Client Typ) registriert sein. Jedoch legt die OAuth 2.0 Spezifikation keine Verfahren fest, wie dieses durchgeführt werden soll. Sie geht lediglich davon aus, dass eine Kommunikation im Vorfeld stattgefunden hat. Allerdings kann es in besonderen Fällen nötig sein, dass ein Client eine Autorisierung vom

Authorization Server erhält, ohne dass im Vorfeld eine Beziehung zwischen den beiden initiiert wurde. An dieser Stelle setzt die Spezifikation des „OAuth 2.0 Dynamic Client Registration Protocols“ [33] an und definiert einen Endpunkt sowie ein Protokoll zur dynamischen Registrierung eines Clients bei einem Authorization Server. Weiterhin werden Methoden festgelegt damit ein Client seine Registrierung mittels einer OAuth 2.0 geschützten Web API dynamisch verwalten kann.

JSON Web Token (JWT) (Draft 12)

Die Spezifikation zu JSON Web Token (JWT) [34] verfolgt den Ansatz Informationen (Namen / Werte Paare), welche in einem JSON Text Object zusammen gefasst werden, gesichert zwischen zwei Parteien als Base 64 codierter String auszutauschen. Gesichert heißt in diesem Zusammenhang, dass die Informationen, sogenannte Claims, signiert und/oder verschlüsselt werden.

JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants (Draft 06)

In dieser Spezifikation [35] wird das „Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants“ dahingehend erweitert, dass ein Extention Grant Typ definiert wird, welcher JSON Web Tokens (JWT) nutzt um Access Tokens anzufragen. Weiterhin können JWT Bearer Token auch für die Client Authentifizierung eingesetzt werden. Beide Einsatzmöglichkeiten können dabei kombiniert oder jeweils einzeln verwendet werden.

SAML 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants (Draft 17)

Diese Spezifikation [36] ist eine weitere Erweiterung des „Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants“. Ähnlich dem JSON Web Token (JWT) Profile wird in dieser Spezifikation eine SAML Assertion als Authorization Grants oder für die Client Authentifizierung genutzt. Zur Übertragung der XML Daten wird die SAML Assertion base64url codiert, wobei auf die Auffüllung mit Gleichheitszeichen, dem sogenannten Padding, verzichtet werden sollte.

OAuth 2.0 Message Authentication Code (MAC) Tokens (Draft 4)

Anstatt einen Bearer Token für den Zugriff auf eine geschützte Ressource innerhalb des HTTP Request zu nutzen, definiert diese Spezifikation [37] einen Message Authentication Code (MAC) Token. Dieses Verfahren setzt voraus, dass ein gemeinsamer Schlüssel zwischen dem Client und dem Resource Server festgelegt wurde. Mittels diesem geheimen Schlüssel und weiterer Parameter als Input, welcher den Access Token mit einschließt, kann der Client eine Prüfsumme generieren, welche bei der Resource Anfrage mit dem Access Token mit geschickt wird. Durch dieses Verfahren kann der Resource Server validieren, ob der Access Token von dem Resource Owner autorisierten Client stammt. Damit der Token nicht trotzdem von einem Angreifer abgefangen und für den Zugriff auf die geschützten Ressourcen verwendet wird, muss die Kommunikationsverbindung (End-zu-End) weiterhin mittels HTTPS gesichert sein.

3.2. Abstrakter Protokollfluss

Bevor im nächsten Abschnitt auf die einzelnen Protokollflüsse eingegangen wird, wird zunächst ein abstrakter Protokollfluss beschrieben [9, sec. 1.2] [1]. Dabei zeigt der in Abbildung 3.1 abstrakt dargestellte Protokollfluss die OAuth spezifischen Interaktion zwischen den einzelnen Rollen (siehe dazu Kapitel 2.2.2).

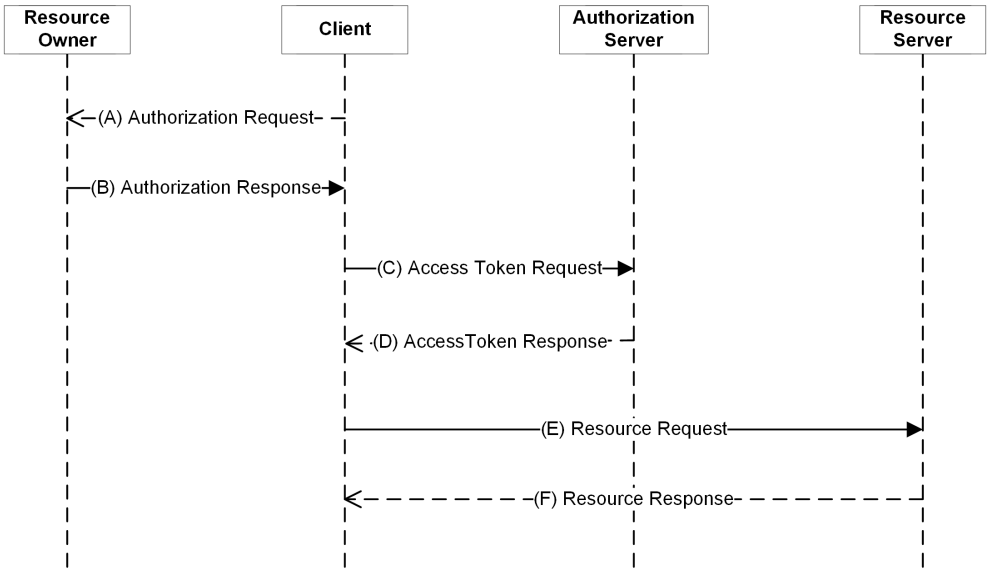

Abbildung 3.1.: Abstrakter Protokollfluss [9]

(A) *Authorization Request*: Der Client benötigt vom Resource Owner eine Autorisierung. Dazu kann die Autorisierungsanfrage direkt an den Resource Owner (wie in Abbildung 3.1 gezeigt wurde) oder über einen Authorization Server als Vermittler gestellt werden.

(B) *Authorization Response*: Falls der Resource Owner der Autorisierungsanfrage zugestimmt hat, wird ein sogenannter Authorization Grant zurück an den Client gesendet.

(C) *Access Token Request*: Nach Erhalt des Authorization Grant kann der Client nun einen Access Token beim Authorization Server anfragen. Dazu übermittelt er ihm den Authorization Grant.

(D) *Access Token Response*: Der Authorization Server authentifiziert bzw. identifiziert den Client und validiert den Authorization Grant. Falls dieser gültig ist, sendet er dem Client einen Access Token zurück.

(E) *Resource Request*: Durch Übersendung des Access Token an den Resource Server, kann der Client geschützte Ressourcen anfragen.

(F) *Resource Response*: Falls der Resource Server einen gültigen Access Token erhält, sendet er die angefragten Ressourcen zurück.

3.2.1. Client Authentifizierung

Nimmt ein vertraulicher Client am Protokollablauf teil, muss dieser sich gegenüber dem Authorization Server authentifizieren. Dabei wird im Vorfeld bei der Registrierung des Client beim Authorization Server ein Authentifizierungsverfahren sowie Credentials bestehend aus einer ID, welche öffentlich zugänglich ist und einem Secret festgelegt. Zur Übertragung der Credentials kann beispielsweise das HTTP Basic Authentication Schema, wobei als Benutzername die ID und als Passwort das Secret festgelegt sind oder einzelne Form Parameter innerhalb eines POST Requests gewählt werden. Jedoch legt sich die OAuth 2.0 Spezifikation nicht näher fest, so dass auch andere HTTP Authentifikationsmethoden[1] verwendet werden können. Ein öffentlicher Client identifiziert sich gegenüber dem Authorization Server lediglich mit seiner ID [9, sec. 2.3].

3.2.2. Protokoll Endpunkte

Innerhalb der OAuth 2.0 Spezifikation werden mehrere Protokollendpunkte definiert [9]:

Authorization Endpoint

Der Client leitet den User Agent des Resource Owners an den Authorization Endpoint weiter, damit dieser mit dem Authorization Server interagieren und der Autorisierungsanfrage zustimmen kann. Vorher muss der Resource Owner durch den Authorization Server authentifiziert werden. Die OAuth Spezifikation gibt hier keine eindeutige Methode zur Authentifizierung vor. Da jedoch Credentials übertragen werden, ist der Einsatz von TLS verpflichtend.

Redirection Endpoint

Nachdem die Interaktion mit dem Resource Owner abgeschlossen wurde, wird der User Agent des Resource Owners zurück an den Client geleitet. Dazu wird während der Registrierung des Clients beim Authorization Server oder beim Authorization Requests mittels eines Parameters der Redirection Endpoint spezifiziert. Der Einsatz von TLS zur Absicherung des Kommunikationskanals wird nur empfohlen. Es wird jedoch ausdrücklich auf die Folgen einer unzureichenden Absicherung, welche die Offenlegung von sensiblen Daten zur Folge haben könnte, hingewiesen. Weiterhin wird empfohlen, die Redirection URI des Clients möglichst genau (bis auf die Query Parameter) zu registrieren. Sollten mehrere Redirection URIs registriert sein, muss die genaue (d.h. einer der registrierten) als Parameter des Authorization Requests übermittelt werden.

Token Endpoint

Um einen Access Token zu erhalten, sendet der authentifizierte Client den Authorization Grant oder den Refresh Token an den Token Endpoint. Da Credentials sowohl an, als auch vom Token Endpoint übertragen werden, wird der Einsatz von TLS vorausgesetzt. Weiterhin legt die OAuth Spezifikation die HTTP POST Methode für Requests an den Token Endpoint fest.

[1] werden durch Zusatz Spezifikationen festgelegt

3.3. Authorization Grant Types

Um diverse Anforderungen und Zwecke durch das Protokoll abdecken zu können, wurden verschiedene Protokollabläufe durch unterschiedliche Definitionen des Authorization Grants erzielt. In der Spezifikation werden dazu vier Grant Types (authorization code, implicit, resource owner password credentials, client credentials) definiert sowie ein Mechanismus vorgestellt, um weitere Grant Types zu spezifizieren. Im anschließenden Abschnitt wird zunächst ausführlich auf den Authorization Code Grant inklusive Darstellung der Requests und Responses eingegangen. Darauf aufbauend werden bei den anderen Authorization Grant Types nur noch die Besonderheiten bzw. die Unterschiede zum Authorization Code Grant dargestellt [9].

Es ist anzumerken, dass innerhalb dieses Abschnittes nicht bei allen Authorization Grant Types auf die Darstellung von Resource Request und Resource Response eingegangen wird. Dieses begründet sich dadurch, dass die Methoden zum Zugriff auf geschützte Ressourcen innerhalb der OAuth Spezifikation nicht spezifiziert sind („*[...] the methods used to access protected resources are beyond the scope of this specification and are defined by companion specifications [...]*“ [9]), diese jedoch teilweise für weitere Analysen benötigt werden.

3.3.1. Authorization Code Grant Type

Der Authorization Code Grant Type wird für serverseitige Anwendungen verwendet, wobei der Client am Ende einen Access Token sowie optional einen Refresh Token erhält. Insgesamt gliedert sich der Authorization Code Flow, wie im vorherigen Kapitel beschrieben, in vier Schritte. Die dazugehörigen Nachrichten werden in Abbildung 3.2 mit den wichtigsten Parametern (zweite Zeile einer jeweiligen Nachricht) dargestellt. Optionale Parameter sind in eckigen Klammern eingefasst.

Authorization Request (Schritte 1-8)

Der Authorization Code Flow wird dadurch begonnen, dass der Client den User Agent des Resource Owners an den Authorization Endpoint weiterleitet. Dazu sendet der Client beispielhaft den im Listing 3.1 dargestellten Request an den Authorization Endpoint:

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz&
   redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

Listing 3.1: Authorization Request beim Authorization Code Grant [9]

Die in diesem Request verwendeten bzw. möglichen Parameter haben folgende Bedeutungen:

- *response_type*: Dieser Parameter definiert den Grant Type und muss in diesem Fall den Wert „code“ besitzen.
- *client_id*: Der verpflichtende client_id Parameter dient dem Authorization Server als Identifizierungsmerkmal für den Client.

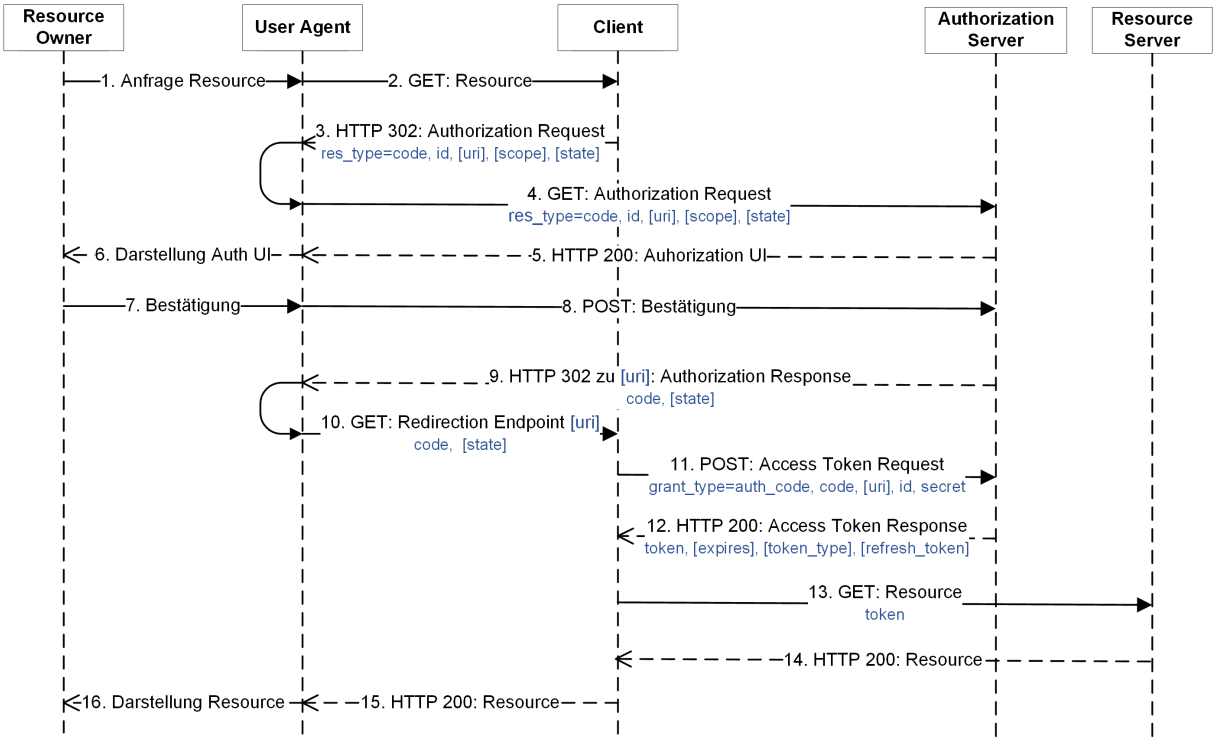

Abbildung 3.2.: Authorization Code Grant [9]

- *redirect_uri*: Nach der Autorisierung wird der User Agent des Resource Owners wieder zurück zu dem Client geleitet. Dazu wird die optional angegebene URL verwendet. Ist keine URL angegeben, wird die bei der Registrierung des Clients beim Authorization Server spezifizierte URL verwendet.
- *scope*: Dieser optionale Parameter definiert die Berechtigungsfelder (siehe Abschnitt 2.2.3).
- *state*: Dieser Parameter kann durch den Client zur Nachverfolgung der Response sowie zum Schutz vor Cross-Site Request Forgery Angriffen genutzt werden.

Falls sich der Resource Owner im Vorfeld nicht gegenüber dem Authorization Server authentifiziert hat, wird der Nutzer mittels einer Anmeldemaske aufgefordert sich zu authentifizieren. Dieses wurde in Abbildung 3.2 nicht dargestellt, da davon ausgegangen wird, dass eine gültige Session besteht. Anschließend erhält der Nutzer einen Bestätigungsdialog (siehe Abbildung 3.3) indem aufgeführt wird, welche Anwendung auf welche Ressourcen zugreifen möchte. In den meisten Fällen kann der Nutzer dieser Anfrage nur zustimmen bzw. ablehnen, jedoch nicht die angefragten Rechte einschränken oder ändern.

Authorization Response (Schritte 9-10)

Hat der Nutzer der Autorisierungsanfrage zugestimmt, leitet der Authorization Server den User Agent mittels der im Authorization Request angegebenen Redirection URL zum Client zurück und übermittelt dabei den Authorization Grant in Form eines Authorization Codes (siehe Listing 3.2).

An den Response fügt der Authorization Server die Parameter:

Abbildung 3.3.: OAuth Consent

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=
    xyz
```

Listing 3.2: Authorization Response beim Authorization Code Grant [9]

- *code*: Der Parameter code enthält den Authorization Code, welcher durch den Authorization Server generiert wurde. Die Verwendung des Codes durch den Client sollte nur einmalig möglich sein. Weiterhin empfiehlt die OAuth Spezifikation die Lebenszeit auf maximal 10 Minuten zu begrenzen.
- *state*: Wurde innerhalb des Authorization Requests der state Parameter verwendet, muss der identische Wert wieder zurück an den Client gesendet werden. Dieser sollte im Anschluss beide Werte miteinander vergleichen um Manipulationen durch einen Cross Site Request Forgery Angriff [38] auszuschließen.

Access Token Request (Schritt 11)

Damit der Client Anfragen an den Resource Server stellen kann, muss er den erhaltenen Authorization Code gegen einen Access Token eintauschen. Dazu sendet der Client den nachstehenden Request (siehe Listing 3.3) an den Token Endpoint (in diesem Fall wird zur Authentifizierung des Clients die HTTP Basic Authentification Methode genutzt)

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA&redirect_uri=
   https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

Listing 3.3: Access Token Request beim Authorization Code Grant Type [9]

Folgende Parameter werden dabei übermittelt:

- *grant_type*: Der Wert muss auf „authorization_code“ gesetzt sein und zeigt an, dass ein Authorization Code gegen ein Access Token getauscht werden soll.
- *code*: Der vom Authorization Server erhaltene Authorization Code.
- *redirect_uri*: Wenn der „redirect_uri“ Parameter während des Authorization Requests gesetzt war, muss hier derselbe Wert angegeben werden.
- *client_id*: Dieser Parameter ist zur Identifikation notwendig, wenn der Client nicht gegenüber dem Authorization Server authentifiziert ist.
- *client_secret*: Dieser Parameter wird zur Übertragung des Client Secrets genutzt, falls die Authentifizierungsinformationen als Query Parameter übertragen werden sollen.

Access Token Response (Schritt 12)

Der Authorization Server validiert die Anfrage nach einem Access Token, in dem er die Authentifizierungsinformationen des Clients, die Gültigkeit des Authorization Codes und die optional angegebene Redirection URL überprüft. Sind diese gültig, sendet er einen Access Token sowie einen optionalen Refresh Token an den Client (in diesem Fall im JSON Format), welches im Listing 3.4 dargestellt ist.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

Listing 3.4: Access Token Response beim Authorization Code Grant Type [9]

Die dabei übermittelten Parameter haben folgende Bedeutung:

- *access_token*: Der vom Authorization Server ausgestellte Access Token.
- *token_type*: Gibt an, um was für eine Art Token es sich handelt.
- *expires_in*: Gibt die Gültigkeitsdauer eines Access Tokens seit seiner Ausstellung in Sekunden an.
- *refresh_token*: Der vom Authorization Server ausgestellte Refresh Token.
- *example_parameter*: Weitere Parameter können beliebig durch den Authorization Server an den Client übermittelt werden.

3.3.2. Implicit Grant Type

Der Implicit Grant Type wird für öffentliche Clients verwendet. In diesem Zusammenhang bedeutet das, dass der Client innerhalb des Browsers des Resource Owners unter Verwendung von JavaScript, Flash u.a. ausgeführt wird. Dadurch besitzt der Client eine geringere Vertrauensstellung als beim Authorization Code Grant Type, da der Client nicht in einer vom Entwickler kontrollierten Umgebung ausgeführt wird. Dementsprechend besitzt der Client keine Authentifizierungsinformationen (Client Secret) und kann sich auch nicht gegenüber dem Authorization Server authentifizieren. Ein großer Sicherheitsaspekt liegt auf der vorherigen Registrierung der Redirection URI. Im Gegensatz zum Authorization Code Grant Type wird auf die Verwendung eines Authorization Codes verzichtet und der Access Token direkt innerhalb des Authorization Response zurück an den Client gesendet. Daher kann nur eine vorherige Registrierung der Redirection URI sicherstellen, dass der Access Token an den richtigen Client gesendet wird.

Weiterhin werden beim Implicit Grant keine Refresh Tokens eingesetzt. Dementsprechend muss der Autorisierungsprozess, wenn die Gültigkeitsdauer des Access Tokens abgelaufen ist, von neuem begonnen werden. Der gesamte Protokollablauf ist in Abbildung 3.4 dargestellt.

Authorization Request (Schritte 1-8)

Der Protokollfluss innerhalb des Authorization Request gleicht dem des Authorization Code Grant Types und wird dadurch initiiert, dass der Client den User Agent an den Authorization Endpoint weiterleitet. Dabei übersendet er die (teilweise optionalen) Parameter wie beim Authorization Code Grant Type: „response_type, client_id, redirection_uri, scope und state“. Der Wert des *response_type* Parameter ist in diesem Fall jedoch auf „token“ gesetzt.

Authorization Response / Access Token Response (Schritte 9-12)

Falls der Resource Owner die Autorisierungsanfrage bestätigt hat, sendet der Authorization Server den Access Token und weitere Parameter an den Client (siehe Listing 3.5).

Der Access Token und die weiteren Parameter werden dabei innerhalb der Fragment Komponente der Redirection URI, welche durch das Doppelkreuz Zeichen („# “) eingeleitet wird, angegeben. Da der Fragment Teil normalerweise zur Referenzierung eines bestimmtes Teils innerhalb einer Ressource verwendet wird, wird er bei HTTP Requests nicht mitgesendet und nur durch den Browser verarbeitet [39].

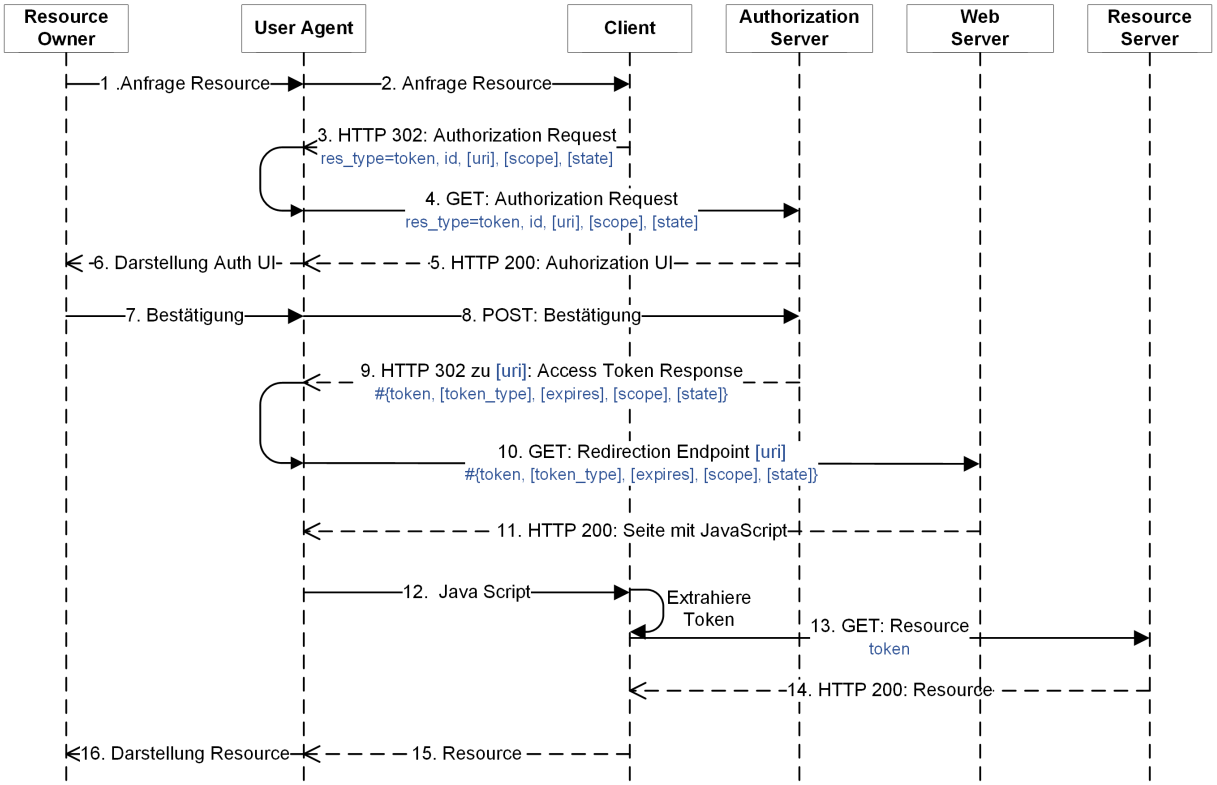

Abbildung 3.4.: Implicit Grant Type[9]

```
HTTP/1.1 302 Found
Location: http://example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA&state=
    xyz&token_type=example&expires_in=3600
```

Listing 3.5: Access Token Response beim Implicit Grant Type [9]

Dadurch wird verhindert, dass der Access Token serverseitig verarbeitet werden kann. Es ist lediglich möglich mittels eines clientseitigen Scriptes, welches z.B. als JavaScript vom Redirection Endpoint geladen wird, den Access Token aus der URL zu extrahieren und somit für die weitere Verarbeitung zu nutzen.

3.3.3. Resource Owner Password Credentials Grant Type

Dieser Grant Type sollte nur eingesetzt werden, wenn ein Vertrauensverhältnis zwischen Resource Owner und Client besteht, da die Credentials (in den meisten Fällen Benutzername und Passwort) des Resource Owners direkt an den Client übermittelt werden. Daher wird empfohlen den Resource Owner Password Credentials Grant Type nur bei offiziellen Anwendungen des Dienstanbieters, wobei dieses dem Nutzer explizit bekannt sein muss, zu verwenden. Als Beispiel sei hier die Mobile Anwendung von Facebook für Smartphones genannt. Der Grant Type bietet jedoch gegenüber der normalen HTTP Basic Authentification den Vorteil, dass die Credentials nicht für jede Anfrage vorgehalten, also gespeichert, werden

müssen, sondern nur einmalig an den Authorization Server übermittelt werden müssen und anschließend wieder gelöscht werden können. Der gesamte Protokollablauf ist in Abbildung 3.5 dargestellt.

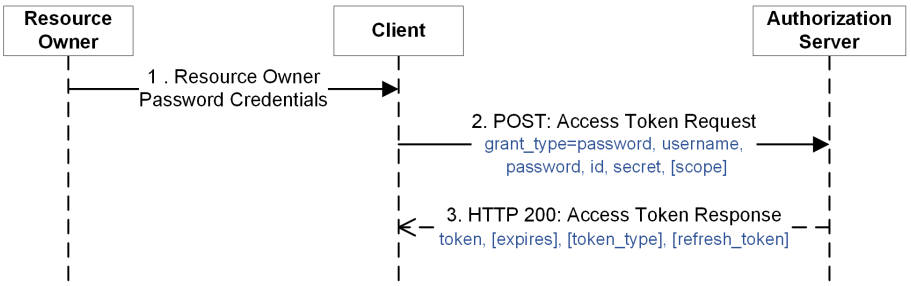

Abbildung 3.5.: Resource Owner Password Credentials Grant Type [9]

Authorization Request / Response (Schritt 1)

Die OAuth Spezifikation geht nicht darauf ein, wie der Client die Credentials des Resource Owners erhalten soll. Es wird lediglich darauf verwiesen, dass der Client nach Erhalt eines Access Tokens die Credentials des Resource Owners löschen muss.

Access Token Request / Response (Schritt 2-3)

Um einen Access Token zu erhalten, stellt der Client eine Anfrage an den Token Endpoint, wobei der Parameter *grant_type* den Wert „password“ besitzt. Weiterhin werden mittels der Parameter *username* und *password* der Benutzername und das Passwort des Resource Owners an den Authorization Server übermittelt. Innerhalb der Anfrage muss sich der Client mit den in Abschnitt 3.2.1 beschriebenen Methoden authentifizieren. Ist die Anfrage gültig, sendet der Authorization Server einen Access Token und optional einen Refresh Token an den Client zurück.

3.3.4. Client Credentials Grant Type

Bei dem Client Credentials Grant Type handelt es sich um das einfachste, der beschriebenen, Verfahren zur Anforderung eines Access Token, da es lediglich aus einem Access Token Request und dem dazugehörigen Response besteht, welches in Abbildung 3.6 dargestellt ist. Dieser Grant Typ kann zur Server-zu-Server Kommunikation eingesetzt werden, wobei der Client selbst der Besitzer der Ressourcen ist und somit z.B. keine delegierte Autorisierung durch einen Resource Owner benötigt [1].

Authorization Request / Response

Da die Client Authentifizierung als Authorization Grant verwendet wird, benötigt dieser Protokollablauf keinen vorhergehenden Authorization Request.

Access Token Request / Response (Schritt 1-2)

Um einen Access Token zu erhalten, stellt der Client eine Anfrage an den Token Endpoint und authentifi-

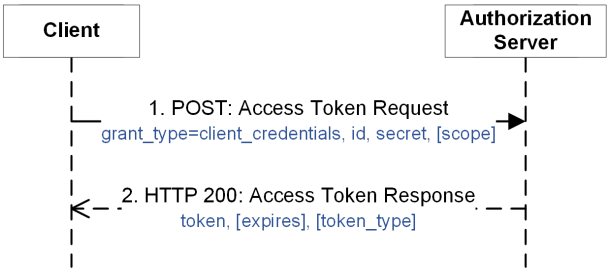

Abbildung 3.6.: Client Credentials Grant Type [9]

ziert sich gegenüber dem Authorization Server (siehe Listing 3.6). Des Weiteren setzt der Client lediglich den Parameter *grant_type* auf den Wert „client_credentials“.

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

Listing 3.6: Access Token Request beim Client Credentials Grant Type [9]

Falls der Access Token Request gültig ist, übermittelt der Authorization Server einen Access Token an den Client. Ein Refresh Token sollte bei diesem Verfahren nicht eingesetzt werden.

3.4. Extension Grant Types

Die OAuth 2.0 Spezifikation bietet die Möglichkeit zusätzliche Grant Types zu spezifizieren, um weitere Client Typen zu unterstützen oder eine Verbindung zwischen OAuth und anderen Frameworks, welche im Zusammenhang mit Authentifizierung und Autorisierung stehen, herzustellen. Daher werden nachfolgend zwei Möglichkeiten zur Übersendung des Authorization Grants mittels Assertions [32] an den Authorization Server vorgestellt. Da Assertions an eine Gültigkeitsdauer gebunden sind, sollte die Lebenszeit des Access Token entsprechend oder niedriger gewählt sein. Weiterhin wird durch den Authorization Server kein Refresh Token ausgestellt, da mittels diesem die Lebenszeit des Access Token erneuert werden kann und somit die Gültigkeitsdauer der Assertion übersteigen könnte.

3.4.1. JWT Bearer Token Grant Type

Der JWT Bearer Token Grant Type beruht auf dem Entwurf „JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants“ [35, sec. 2.1] und setzt einen JWT Bearer Token als Authorization Grant ein, um einen Access Token anzufordern. Der Protokollablauf wird in Abbildung 3.7 dargestellt.

Abbildung 3.7.: JWT Bearer Token Grant Type

Access Token Request / Response (Schritt 1-2)

Um einen Access Token vom Authorization Server zu erhalten, sendet der Client den im Listing 3.7 dargestellten Request an den Token Endpoint. Eine Authentifizierung des Clients (siehe Kapitel 3.2.1) ist bei diesem Grant Type optional.

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt-bearer
&assertion=eyJhbGciOiJFUzI1NiJ9.
eyJpc3Mi[...].
J9l-ZhwP[...]
```

Listing 3.7: Access Token Request beim JWT Bearer Token Grant Type [35]

Die in diesem Request verwendeten bzw. möglichen Parameter haben folgende Bedeutungen:

- *grant_type*: Der grant_type Parameter muss in diesem Fall auf den Wert „urn:ietf:params:oauth:grant-type:jwt-bearer“ gesetzt werden und gibt an, dass der JWT Bearer Token Grant Type verwendet werden soll.
- *assertion*: Beinhaltet den base64url codiereten JWT Bearer Token, welcher von der Client Anwendung erzeugt wurde.
- *scope*: Der optionale scope Parameter kann dazu verwendet werden, um die anzufragenden Berechtigungsfelder anzugeben.

Nach Erhalt des Access Token Requests validiert der Authorization Server den JWT Bearer Token. Dazu überprüft er, ob der JWT Bearer Token Abgaben über den Aussteller, das Subjekt, den Adressaten und die Gültigkeit beinhaltet. Sind diese Werte gültig, sendet er einen Access Token an den Client zurück.

Bildung des JWT Bearer Token

Ein JWT Bearer Token [35] [34] besteht aus drei Teilen, welche mittels eines Punktes („.“) zu einer Zeichenkette zusammengefasst werden. Der erste Teil ist der sogenannte JWT Header. Innerhalb dieses

Headers werden kryptografische Verfahren sowie weitere optionale Parameter für die Verwendung des JWTs angegeben. Der zweite Teil des JWT Bearer Token beinhaltet die Informationen, welche mittels des Tokens übertragen werden sollen. Abschließend werden die ersten beiden Teile durch Angabe einer Signatur oder eines Keyed-Hash Message Authentication Codes im dritten Teil gegen Veränderungen abgesichert.

Zur Bildung eines JWT Bearer Token wird zunächst der JWT Header festgelegt. Als Beispiel dafür sei der im Listing 3.8 dargestellte JWT Header genannt. Der typ Parameter gibt den MIME Media Type an und ist grundsätzlich auf den Wert „JWT“ gesetzt. Der Algorithmus Parameter legt in diesem Fall fest, dass als kryptografische Funktion der HMAC SHA-256 Algorithmus verwendet wird.

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

Listing 3.8: JWT Header [34]

Anschließend werden mittels des sogenannten Claims Informationen festgelegt, welche an den Authorization Server übertragen werden sollen. Ein beispielhafter Claim Set, welches als ein JSON Object angelegt wird, wird im Listing 3.9 gezeigt.

```
{"iss":"client-id-xyz1234",
 "sub":"mike@example.com",
 "aud":"https://auth.example.net",
 "nbf":1300815780,
 "exp":1300819380}
```

Listing 3.9: JWT Claim Set [35]

Die in diesem Claim Set verwendeten bzw. möglichen Claims haben nachfolgende Bedeutungen:

- *iss*: Der Issuer Claim identifiziert den Aussteller des JWT und kann z.B. zur Übertragung der Client ID verwendet werden.
- *sub*: Der Subject Claim definiert den Gegenstand des JWTs und bezieht sich z.B. auf den Resource Owner.
- *aud*: Der Audience Claim teilt den Adressaten des JWTs mit und wird z.B. zur Adressierung des Authorization Servers eingesetzt.
- *exp*: Der Expiration Time Claim gibt die Gültigkeitsdauer des JWT durch Festlegung eines Zeitpunktes an.
- *nbf*: Der Not Before Claim bestimmt ab welchem Zeitpunkt der JWT gültig ist.
- *iat*: Der Issued At Claim nennt den Ausstellungszeitpunkt des JWTs.
- *jti*: Der JWT ID Claim legt einen eindeutigen Identifizierer für den JWT fest.

Es ist jedoch zu beachten, dass es sich bei den vorgestellten Claims nur um die innerhalb des Standards erläuterten Claims handelt. Das Claim Set kann jedoch um beliebige eigene Claims erweitert werden, solange daraus ein valider JWT Bearer Token entsteht.

Zur letztendlichen Zusammensetzung des JWT Bearer Token werden die UTF-8 Repräsentation beider JSON Objects (JWT Header und JWT Claim Set) jeweils base64url codiert und mittels eines Punktes zusammengefügt. Anschließend wird auf dieser Zeichenkette die im JWT Header angegebene kryptografische Funktion angewendet. Die resultierende Zeichenkette wird wieder base64url codiert und als dritter Teil durch einen Punkt separiert an die vorher erzeugte Zeichenkette angehängt. Alle drei Teile zusammen ergeben nun den JWT Bearer Token, welcher in einem Access Token Request verwendet werden kann (siehe dazu Listing 3.7).

3.4.2. SAML 2 Bearer Assertion Grant Type

Der SAML 2 Bearer Assertion Grant Type beruht auf dem Entwurf „SAML 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants“ [36, sec. 2.1] und setzt eine SAML 2 Assertion als Authorization Grant ein, um einen Access Token anzufordern. Der Protokollablauf wird in Abbildung 3.8 dargestellt.

Abbildung 3.8.: SAML 2 Bearer Assertion Grant Type

Access Token Request / Response (Schritt 1-2)

Um einen Access Token vom Authorization Server zu erhalten, sendet der Client den im Listing 3.10 dargestellten Request an den Token Endpoint. Eine Authentifizierung des Clients (siehe Kapitel 3.2.1) ist bei diesem Grant Type optional.

```
POST /oauth2/token HTTP/1.1
Host: authz.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Asaml2-bearer&
assertion=PHNhbWxwOl[...]
```

Listing 3.10: Access Token Request beim SAML 2 Bearer Assertion Grant Type [36]

Die in diesem Request verwendeten bzw. möglichen Parameter haben folgende Bedeutungen:

- *grant_type*: Der grant_type Parameter muss in diesem Fall auf den Wert „urn:ietf:params:oauth:grant-type:saml2-bearer“ gesetzt werden und gibt an, dass der SAML 2 Bearer Assertion Grant Type verwendet werden soll.
- *assertion*: Beinhaltet die base64url codierete SAML 2 Assertion, welche von der Client Anwendung erzeugt wurde oder von einem SAML Identity Provider ausgestellt wurde.
- *scope*: Der optionale scope Parameter kann dazu verwendet werden, um die anzufragenden Berechtigungsfelder anzugeben.

```
<Assertion IssueInstant="2010-10-01T20:07:34.619Z"
  ID="ef1xsbZxPV2oqjd7HTLRLIBlBb7"
  Version="2.0"
  xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
  <Issuer>https://saml-idp.example.com</Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    [...aus Uebersichtsgruenden gekuerzt...]
  </ds:Signature>
  <Subject>
    <NameID
      Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
     mike@example.com
    </NameID>
    <SubjectConfirmation
      Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
     <SubjectConfirmationData
       NotOnOrAfter="2010-10-01T20:12:34.619Z"
       Recipient="https://authz.example.net/oauth2/token"/>
     </SubjectConfirmation>
  </Subject>
  <Conditions>
    <AudienceRestriction>
       <Audience>https://saml-sp.example.net</Audience>
    </AudienceRestriction>
  </Conditions>
  <AuthnStatement AuthnInstant="2010-10-01T20:07:34.371Z">
    <AuthnContext>
      <AuthnContextClassRef>
        urn:oasis:names:tc:SAML:2.0:ac:classes:X509
      </AuthnContextClassRef>
    </AuthnContext>
  </AuthnStatement>
</Assertion>
```

Listing 3.11: SAML 2 Assertion [36]

Die SAML 2 Assertion, welche beispielhaft im Listing 3.11 veranschaulicht wird, muss wie der JWT Bearer Token Angaben über den Aussteller, das Subjekt, den Adressaten und die Gültigkeit ent-

halten. Damit der Authorization Server die SAML 2 Assertion als Bearer Assertion annehmen kann, muss das *Subject* Element ein *SubjectConfirmation* Element beinhalten, welches ein *Method* Attribute mit dem Wert „urn:oasis:names:tc:SAML:2.0:cm:bearer“ besitzt. Weiterhin muss innerhalb des Kindelements *SubjectConfirmationData* mittels des *NotOnOrAfter* Attributs die Gültigkeitsdauer der Assertion angegeben werden.

Im Fall einer gültigen Validierung der SAML 2 Assertion, sendet der Authorization Server einen Access Token an den Client zurück.

3.5. Sicherheitsempfehlungen und -vorkehrungen

Die OAuth 2.0 Spezifikation [9, sec. 10] sowie der RFC 6819 [11] führen mehrere Empfehlungen und Vorkehrungen auf, welche die Sicherheit gegenüber Risiken und Angriffen auf das OAuth 2.0 Protokoll gewähren sollen. Nachfolgend werden einige von ihnen beschrieben.

Allgemein

SR 1. *Vertraulichkeit von Nachrichten*: Die Vertraulichkeit von Anfragen, welche vom Client ausgehen, sollten durch den Einsatz von TLS sichergestellt werden. Durch die Maßnahme können Angriffe, welche auf das Abfangen oder Wiedereinspielen von Nachrichten gerichtet sind, verhindert werden. Des Weiteren sollten der Authorization Code, Tokens sowie Credentials grundsätzlich nur gesichert mittels TLS übertragen werden.

SR 2. *Informationen gegenüber dem Resource Owner*: Der Resource Owner sollte in Bezug auf die Autorisierung einer Dritt-Anwendungen mit ausreichenden Informationen versorgt werden, damit dieser eine kontrollierte Entscheidung treffen kann. Die Einbindung des Resource Owners und die dadurch erreichte Transparenz können nach Ansicht der Autoren des RFCs zu einer besseren Aufdeckung von bestimmten Angriffen als durch den Authorization Server führen.

SR 3. *Sichere Speicherung von Geheimnissen jeglicher Art*: Alle beteiligten Parteien sollten Geheimnisse jeglicher Art (z.B. Token, Credentials, Authorization Code) sicher speichern. Dementsprechend sollten Credentials nicht im Klartext gespeichert werden, sondern möglichst als Hash oder verschlüsselt abgelegt werden. Des Weiteren sollten Credentials des Clients nicht direkt in der Anwendung gespeichert werden, sondern aus einem Keystore oder einen Datenbank bezogen werden.

Tokens (Access, Refresh, Code)

SR 4. *Komplexität der Tokens*: Der Authorization Server muss sicherstellen, dass die Tokens eine gewisse Komplexität aufweisen. So darf es für einen Angreifer in einer angemessenen Zeitspanne nicht möglich sein, einen gültigen Token zu genieren oder durch iteratives Ausprobieren zu raten.

SR 5. *Gültigkeitsdauer*: Ein Token sollte nach einer nachvollziehbaren Zeitdauer ablaufen. Dabei sollte die Zeitdauer so gering wie nötig gehalten werden, um z.B. die Ausmaße bei einem Verlust oder einer Offenlegung gering zu halten.

SR 6. *Limitierung von Anfragen*: Die Anzahl an Anfragen, die mit einem Token durchgeführt werden und/oder die an einen Endpunkt gesendet werden können, sollten durch den Authorization Server und den Resource Server festgelegt sein, um Replay oder Brute Force Angriffe einzuschränken. In Bezug auf den Authorization Code sollte es nur möglich sein, damit einen Access Token anzufragen. Erhält der Authorization Server eine Anfrage mit einem schon einmal übersendeten Authorization Code, dann sollte er diese Anfrage verwerfen oder vorher ausgestellte Access Token widerrufen.

SR 7. *Limitierung des Scopes*: Der Client sollte bei dem Authorization Request nur die minimale Anzahl an Scopes anfragen, die er zur Erfüllung seiner Tätigkeit benötigt. Auch sollte der Authorization Server nur aufgrund vorher definierter Regelwerke bestimmte Scopes erlauben können. Die Grundlage für diese Regelwerke sollten der Client Typ sowie der angebotene Dienst des Clients sein. Dementsprechend sollte es für einen Client, bei dem die möglichen Scopes im Vorfeld festgelegt wurden, nicht möglich sein darüber hinaus gehende Scopes anzufragen und somit auf die gesamte API des Dienstanbieters zugreifen zu können.

SR 8. *Bindung der Tokens an eine Client ID*: Ein Token sollte an die entsprechende ID des verwendeten Clients gebunden sein und bei allen Anfragen validiert werden. Folglich sollte es durch einen Angreifer mit einem eigenen Client (anderer Client ID) nicht möglich sein einen Access Token mit einem abgefangenen Authorization Code anzufragen. Die Validierung der Client ID sollte möglichst auch eine Authentifizierung des Client zur Folge haben.

SR 9. *Widerrufen von Tokens*: Der Authorization Server sollte sowohl für den Client als auch für den Resource Owner eine Möglichkeit bieten, um Tokens zu widerrufen.

SR 10. *Rotation von Refresh Tokens*: Um Missbrauchsmöglichkeiten bei Refresh Tokens einzuschränken, sollte bei Anfragen nach der Neuausstellung eines Access Tokens mittels eines Refresh Tokens auch ein neuer Refresh Token übersendet werden. Dadurch wird verhindert, dass Refresh Tokens auf unterschiedlichen Geräten parallel verwendet werden können.

SR 11. *Übersendung von Access Tokens*: Wird der Access Token an den Resource Server übertragen, sollte er möglichst innerhalb des Authorization Headers des HTTP Requests übermittelt werden. Da der Authorization Header von HTTP Proxies und Servern gesondert verarbeitet wird, senkt es die Wahrscheinlichkeit des Datenverlustes bzw. die unbeabsichtigte Speicherung des autorisierten Requests und somit auch des Access Token im Allgemeinen.

Authentizität

SR 12. *Client Authentifizierung*: Der Authorization Server muss den Client, wenn immer es möglich ist, authentifizieren. Im Fall eines öffentlichen Clients, welcher sich nicht durch ein geheimes Authentifizierungsmerkmal authentifizieren kann, muss mindestens eine Redirection URI beim Authorization Server registriert werden.

SR 13. *Validierung der registrierten Redirection URI*: Grundsätzlich sollte der Authorization Server verlangen, dass sich Clients bei ihm im Vorfeld mit einer oder mehreren Redirection URIs registrieren. Bei Anfragen sollte die Redirection URI gegen die gespeicherte genau überprüft werden. Falls sie nicht genau übereinstimmt, sollte davon ausgegangen werden, dass sie von einem Angreifer stammt und die Anfrage verworfen werden. Weiterhin sollte die Redirection URI an eine Client ID gebunden sein, so dass sicher gestellt ist, dass der Client nur an die vorher registrierte Redirection URI weiter geleitet werden kann.

SR 14. *Verhinderung von Open Redirectors*: Bei der Angabe der Endpunkte (siehe Abschnitt 3.2.2) sollte darauf geachtet werden, dass diese nicht durch einen Angreifer als Open Redirector genutzt werden können. Bei einem Open Redirector handelt es sich um eine automatische und unvalidierte Weiterleitung des User Agents an einen Ort, welcher innerhalb der URL mittels eines Parameters bestimmt werden kann. Als Beispiel sei z.B. folgende URL genannt: `http://example.com/login.php?url=welcome.php`. Hierbei wird ein Nutzer nach dem erfolgreichem Login auf die Willkommensseite weitergeleitet. Open Redirectors können von Angreifern genutzt werden, um einen Benutzer unter Vortäuschung einer vertrauten Adresse auf eine bösartige Webseite zu locken.

4. Durchführung der manuellen Sicherheitsanalyse

Nachfolgend wird die Vorgehensweise zur Durchführung der Sicherheitsanalyse beschrieben. Die Grundlage für die manuelle Sicherheitsanalyse ist ein Untersuchungskatalog, der iterativ durch experimentelles Ausprobieren von OAuth 2.0 Implementationen entstanden ist. Der gesamte Verlauf der Durchführung der manuellen Sicherheitsanalyse ist in Abbildung 4.1 dargestellt. Die folgenden Unterkapitel beziehen sich auf einzelne Schritte innerhalb des Aktivitätsdiagrammes.

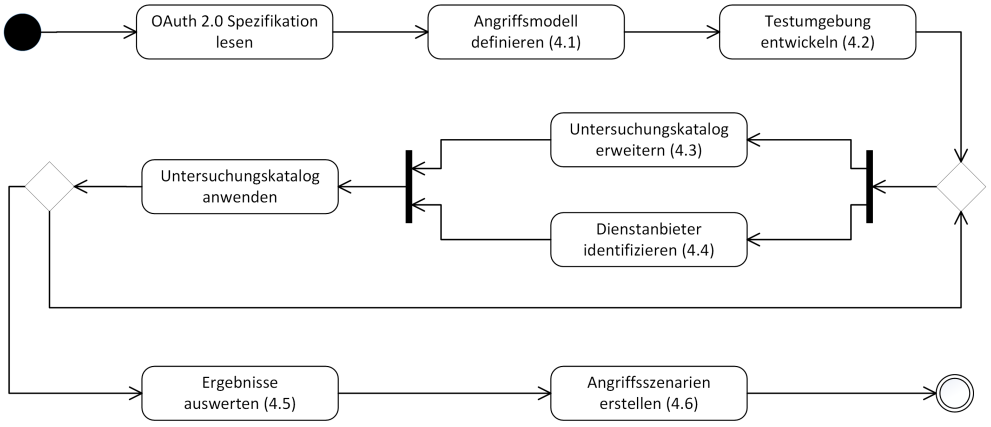

Abbildung 4.1.: Aktivitätsdiagramm der Durchführung der manuellen Sicherheitsanalyse

4.1. Angriffsmodell

Im folgenden Abschnitt wird das Angriffsmodell für die späteren Analysen detailliert beschrieben. Dazu wird zunächst das Ziel des Angreifers dargelegt, sowie grundlegende Annahmen erläutert, welche bei der Durchführung des Protokolls, bei der Implementierung durch einen Dienstanbieter bzw. bei der Analyse gelten. Anschließend werden für das Angriffsmodell die Fähigkeiten eines Angreifers aufgezeigt sowie dargestellt, wie sich ein Benutzer im Allgemeinen verhält.

4.1.1. Ziele des Angreifers

Ein grundlegendes Ziel, welches der Angreifer verfolgt, ist den unautorisierten Zugriff auf die Daten des Resource Owners beim Identity bzw. Resource Provider zu bekommen. Dies bedeutet in Bezug auf

das OAuth Protokoll, dass der Angreifer in Besitz eines Access Token eines Opfers gelangen muss oder eine Client Anwendung so täuschen muss, dass diese ihm den Zugriff auf die Ressourcen eines Opfers gewährt.

4.1.2. Annahmen

Grundsätzlich wird davon ausgegangen, dass der Browser sowie der Computer des Benutzers nicht kompromittiert sind. Das bedeutet, dass ein Angreifer keine Sicherheitslücken im Zusammenhang mit der Benutzung des User Agents durch den Benutzer bzw. des darunterliegenden Betriebssystems ausnutzt.

Des Weiteren werden sowohl der Resource Server als auch der Authorization Server aufgrund der fehlenden Dokumentation und des fehlenden Zugriffs auf den Source Code als Black Box gehandhabt. Interne Strukturen, Prozesse sowie Reaktionen auf bestimmte Eingaben sind dem Angreifer nicht bekannt und können nur teilweise durch eigene Tests und Analysen nachgestellt werden. Auch besitzt der Angreifer keinen Zugriff auf die Netzwerkkommunikation zwischen dem Authorization Server und Resource Server. Daher liegen Prozesse des OAuth 2.0 Protokolls, welche durch einen Webserver durchgeführt werden, nicht im Geltungsbereich von Angriffen. Dazu gehören unter anderem der Austausch des Authorization Codes gegen einen Access Token oder der Zugriff auf Ressourcen beim Resource Server mittels eines Access Token beim Authorization Grant Type. Es ist lediglich möglich im Vorfeld Daten, die innerhalb dieser Prozesse verarbeitet werden, im Rahmen der Fähigkeiten des Angreifers zu manipulieren und die resultierenden Auswirkungen für sich zu nutzen.

Weiterhin kann angenommen werden, dass falls TLS zur Absicherung des Kommunikationskanals eingesetzt wird, dieses grundsätzlich sicher ist. Ein Angreifer wird sich in diesem Zusammenhang keine Angriffe wie z.B. den BEAST Angriff auf TLS [40] zu Nutze machen, um die Kommunikation zu entschlüsseln und somit Zugriff auf einen Access Token zu erhalten. Jedoch sei hier noch einmal erwähnt, dass die Sicherheit des OAuth 2.0 Protokolls hauptsächlich auf die Verschlüsselung der Kommunikation mittels TLS beruht: „*No matter what you do, if for any reason the SSL/TLS layer is broken, the entire security architecture of OAuth bearer tokens falls apart*.“[41] Außerdem weist Eran Hammer innerhalb seiner Artikel [41] [4] darauf hin, dass es sich dabei lediglich um eine Ende-zu-Ende Verschlüsselung handelt, jedoch der Benutzer durch Angriffe auf den Client oder User Agent z.B. mittels Phishing, Cross Site Scripting oder falschen TLS Zertifikaten getäuscht werden kann. Diese Art von Angriffen ist jedoch nicht Bestandteil der Analyse.

Eine weitere Annahme für die Analyse ist, dass die Umsetzungen von Sicherheitsrichtlinien des „OAuth Thread Modell“ [11] durch die Dienstanbieter als nicht gegeben angesehen werden. Das bedeutet in diesem Zusammenhang, dass Risiken, welche in diesem Dokument identifiziert worden sind, nicht unbedingt auch von den Dienstanbietern erkannt worden sind. Daher muss bei der Analyse davon ausgegangen werden, dass die beschrieben Risiken weiterhin auftreten können. Dementsprechend werden sie bei der Analyse berücksichtigt und teilweise auch geprüft.

4.1.3. Fähigkeiten des Angreifers

Innerhalb dieser Arbeit wird als Angreifertyp ein Webangreifer [42] [43] [44] verwendet, welcher im Gegensatz zum Netzwerkangreifer keine beliebigen Netzwerkverbindungen belauschen oder manipulieren kann. Der Webangreifer besitzt daher keine speziellen Netzwerkrechte. Er kann lediglich willkürliche, d.h. auch nicht standardkonforme, HTTP Requests von einem seinerseits kontrollierten Netzwerk Endpunkt an einen Server senden. Ebenfalls muss er Antworten nicht dem Standard gemäß verarbeiten, so dass er z.B. HTTP Weiterleitungen nicht folgen muss.

Des Weiteren kann der Webangreifer eine bösartige Webseite aufsetzen. Das bedeutet, dass der Angreifer mindestens die Kontrolle über einen Webserver besitzt und Anfragen an diesen Webserver mit bösartigen Inhalten beantworten kann. Dementsprechend kann er aber, wenn ein Benutzer eine bösartige Webseite des Angreifers aufsucht, die ihm dadurch zur Verfügung gestellte Browser API des Benutzers nutzen. So kann er z.B. mittels der Funktion *window.open* ein neues Browser Fenster mit einer beliebigen URL öffnen.

Darüber hinaus kann der Webangreifer Kommentare inklusive Links auf beliebigen Webseiten veröffentlichen sowie bösartige Links via E-Mails oder Anzeigen verbreiten. Jedoch wird er keine Schwachstellen wie z.B. Cross-Site Scripting auf den Seiten der jeweiligen Dienstanbieter ausnutzen. Weiterhin wird er die Back-End Server der Identity und Resource Provider nicht kompromittieren sowie keine Angriffe auf den User Agent und das Betriebssystem des Benutzers durchführen.

Im Allgemeinen kann sich jedoch der Webangreifer beliebig viele Accounts beim Identity bzw. Resource Provider erstellen, um über die Account Informationen auf Gemeinsamkeiten und Muster schließen zu können. Ferner kann er Informationen, welche öffentlich auf den Servern zugänglich sind, nutzen.

4.1.4. Verhalten des Benutzers

Über das Verhalten des Benutzers wird die Annahme getroffen [43], dass der Benutzer jede öffentlich zugängliche Webseite inklusive die durch einen Angreifer erstellte Webseite besucht. Dementsprechend klickt der Benutzer auch auf für ihn vertrauenswürdige Links z.B. in E-Mails oder in Foren. Jedoch erkennt er Anzeichen für Phishing, so dass der Benutzer nicht durch Nachbauen einer Webseite zur Eingabe von vertraulichen Daten getäuscht werden kann. Anzeichen können in diesem Sinne Fehler innerhalb der Gestaltung der Webseite oder sprachliche Fehler sein. Demgegenüber kann davon ausgegangen werden, dass ein Benutzer nicht die Richtigkeit von Query Parametern innerhalb einer URL überprüft.

4.2. Entwicklung einer Testumgebung

Zu Beginn der Analyse wurde das OAuth 2.0 Protokoll mittels des Google OAuth 2.0 Playgrounds [45] detailliert betrachtet. Dieser Playground ermöglicht es Entwicklern durch Ausprobieren von Requests das OAuth 2.0 Protokoll sowie entsprechende Google APIs, welche das OAuth 2.0 Protokoll einsetzen, näher kennen zu lernen. So wird der Entwickler Schritt für Schritt durch den Protokollfluss für serverseitige Webanwendungen geführt. Nacheinander können die Autorisierung von Scopes, der Austausch des

Authorization Codes in einen Access Token (siehe dazu Abbildung 4.2), die Erneuerung eines Access Tokens sowie autorisierte API-Requests durchgeführt werden. Zu jedem Schritt wird der entsprechende HTTP Request und Response angezeigt.

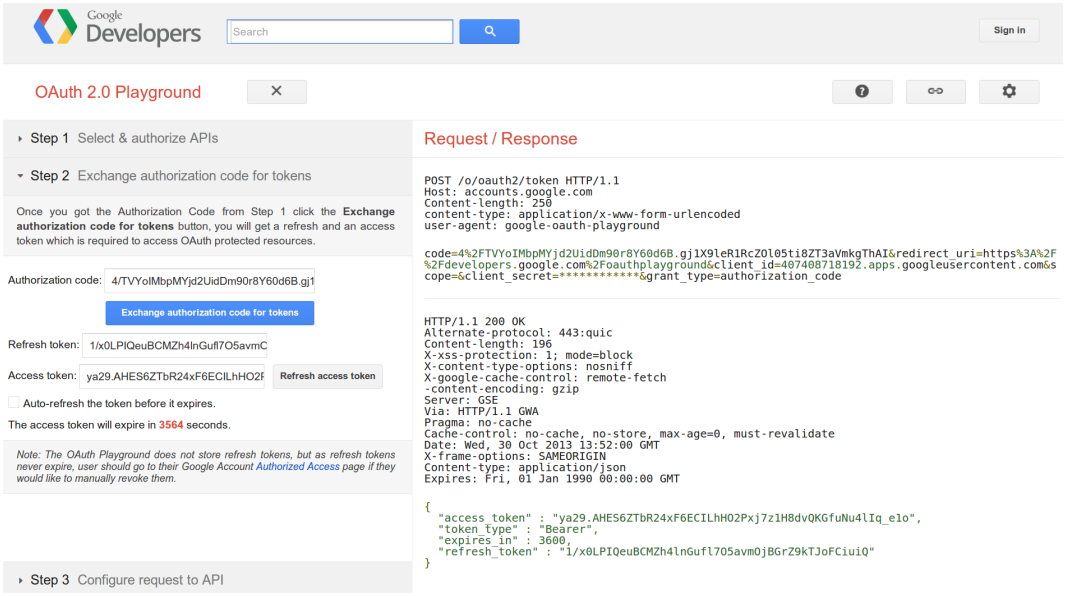

Abbildung 4.2.: Google OAuth 2.0 Playground [45]

Parallel dazu wurde eine eigene Client Anwendung für den Zugriff auf die Google API entwickelt. Als Programmiersprache für die Client Anwendung wurde PHP gewählt, da diese sowohl bezüglich der Client und Server Implementationen [46] sowie innerhalb der Dokumentationen der Anbieter [47, 48] weit verbreitet ist. Zunächst wurden alle Clientfunktionalitäten für das tiefer gehende Verständnis selbst programmiert, jedoch wurde im Laufe der Entwicklung auf eine vorhandene Client Bibliothek umgestellt. Dabei handelt es sich um die Bibliothek „Light PHP wrapper for the OAuth 2.0 protocol“ [49]. Im Gegensatz zu anderen Client Bibliotheken [46] bietet die verwendete Bibliothek alle benötigten Funktionen innerhalb eines kompakten und einfach zugänglichen Codes an.

Um weitere OAuth-Implementierungen zu analysieren, wurde die Google Client Anwendung zu einer universellen Test Client Anwendung ähnlich dem Google Playground weiterentwickelt. Mittels modularen Konfigurationsdateien kann eine entsprechende Client Anwendung eines Dienstanwenders emuliert werden. Dazu muss eine entsprechende Anwendung bei dem Dienstanbieter registriert werden und die zugehörigen Daten (Name des Anbieters, Authorization Endpoint, Token Endpoint, Registrierte Redirection URI, Client ID, Client Secret) in der globalen Konfigurationsdatei der Test Client Anwendung hinterlegt werden. In einer für jeden Provider separaten Konfigurationsdatei können weitere Angaben wie Scopes und zugehörige API Aufrufe angegeben werden.

Wird die Test Client Anwendung durch einen Anwender besucht, hat dieser zu Beginn die Möglichkeit zwischen den vorher konfigurierten Anbietern auszuwählen. Anschließend werden alle notwendigen Daten aus der Konfigurationsdatei für den Authorization Request geladen und in den entsprechenden

Textfeldern der Test Client Anwendung angezeigt (siehe Abbildung 4.3). Der Anwender kann dabei alle Parameterwerte beliebig verändern. Bezüglich des Scopes kann der Anwender ihn selber definieren oder ihn aus einer Liste auswählen. Ein Feld für weitere Parameter erlaubt es auch nicht spezifizierte Parameter an den Authorization Server zu senden.

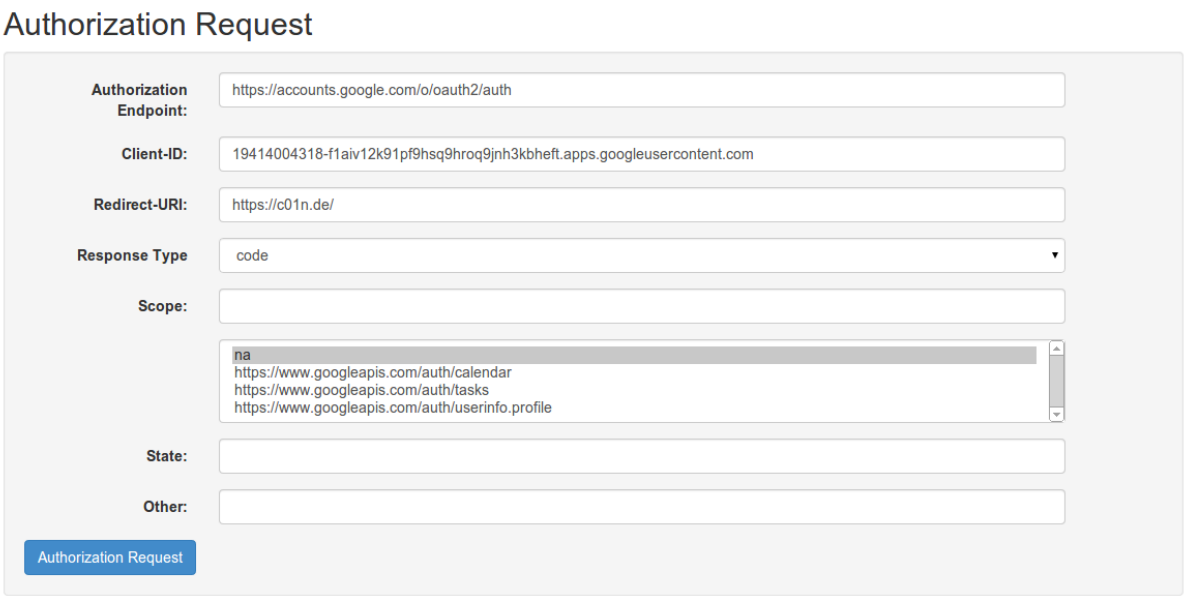

Abbildung 4.3.: Test Client Anwendung - Authorization Request

Nach Absenden des Authorization Request wird der User Agent entsprechend der OAuth 2.0 Spezifikation an den Authorization Endpunkt weiter geleitet. Dort autorisiert der Anwender die entsprechende Anwendung mit den vorher definierten Scopes. Anschließend wird der User Agent, falls die Redirection URI nicht verändert wurde, zurück zur Test Client Anwendung geleitet. Da alle Werte des Authorization Request in einer Session Variable gespeichert wurden, werden diese wieder innerhalb des Authorization Request Bereiches angezeigt und können für weitere Testzwecke verändert und erneut abgeschickt werden. Unterhalb dieses Bereiches wird der Authorization Response mit dem Authentication Code angezeigt. Weiterhin werden alle notwendigen Informationen für den Access Token Request geladen und im zugehörigen Bereich in einem Formular angezeigt (siehe Abbildung 4.4). Auch hier ist es für den Anwender wieder möglich sämtliche Parameterwerte zu verändern.

Wird der Access Token Request abgeschickt, wird intern mittels cURL [50] die Anfrage an den Access Token Endpoint gesendet. Anschließend wird der Access Token Response im dazugehörigen Bereich angezeigt. Falls der Access Token Request gültig war und ein Access Token an die Anwendung gesendet wurde, wird der Bereich für den Resource Request angezeigt. Der Anwender kann nun entweder die Ressource URL frei eingeben oder, falls es vorher konfiguriert wurde, sie aus einer der beiden Listen auswählen (siehe Abbildung 4.5). In der ersten Liste werden Ressource URLs angezeigt, welche dem im Authorization Request angegebenen Scope entsprechen. Die zweite Liste umfasst Ressource URLs, welche außerhalb des Scopes sind. Dadurch kann der Anwender testen, ob der Anbieter eine wirkliche

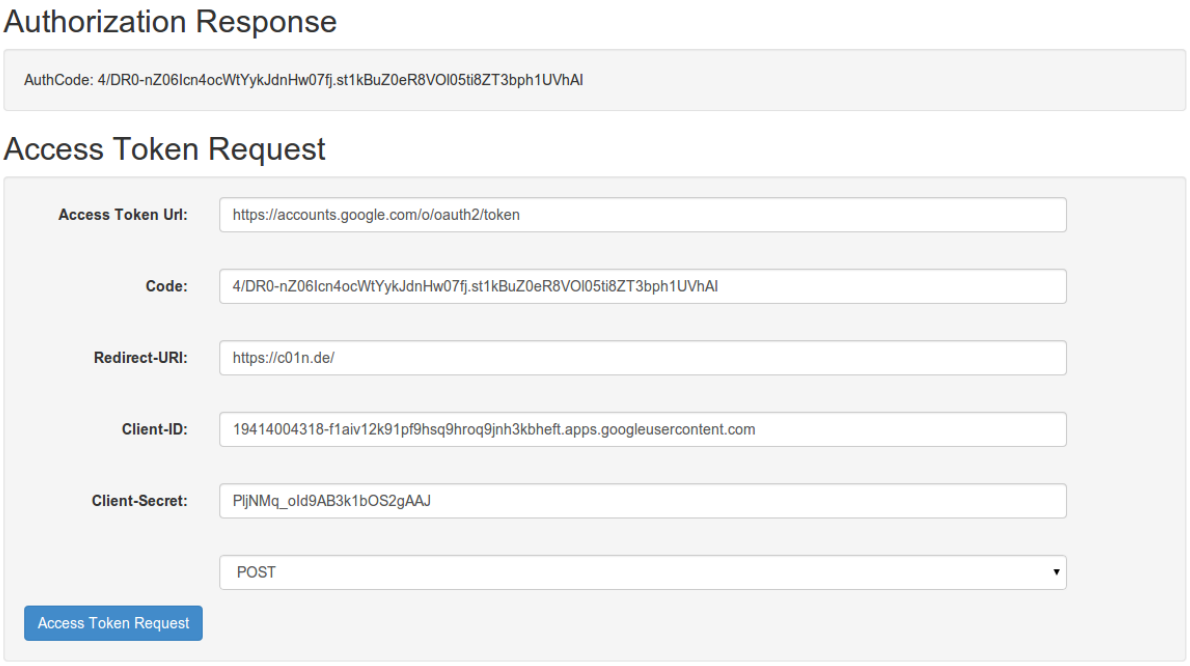

Abbildung 4.4.: Test Client Anwendung - Authorization Response und Access Token Request

Rechtetrennung auf Basis des autorisierten Scopes vornimmt und somit nur Ressourcen zurück liefert, welche der Resource Owner autorisiert hat. Die Grundlage für die Abbildung 4.5 war, dass innerhalb des Authorization Request Rechte für die Verwaltung des Kalenders angefordert wurden. Dementsprechend enthält die erste Liste die Resource URL zur Abfrage der Kalenderliste. Da keine Rechte für die Verwaltung von Aufgaben angefordert wurden, beinhaltet die zweite Liste die Resource URL zur Abfrage der Aufgabenliste.

Enthält der Access Token Response einen Refresh Token wird ein separater Bereich für die Erneuerung des Access Tokens dargestellt. Der Anwender hat jederzeit die Möglichkeit den Access Token zu erneuern.

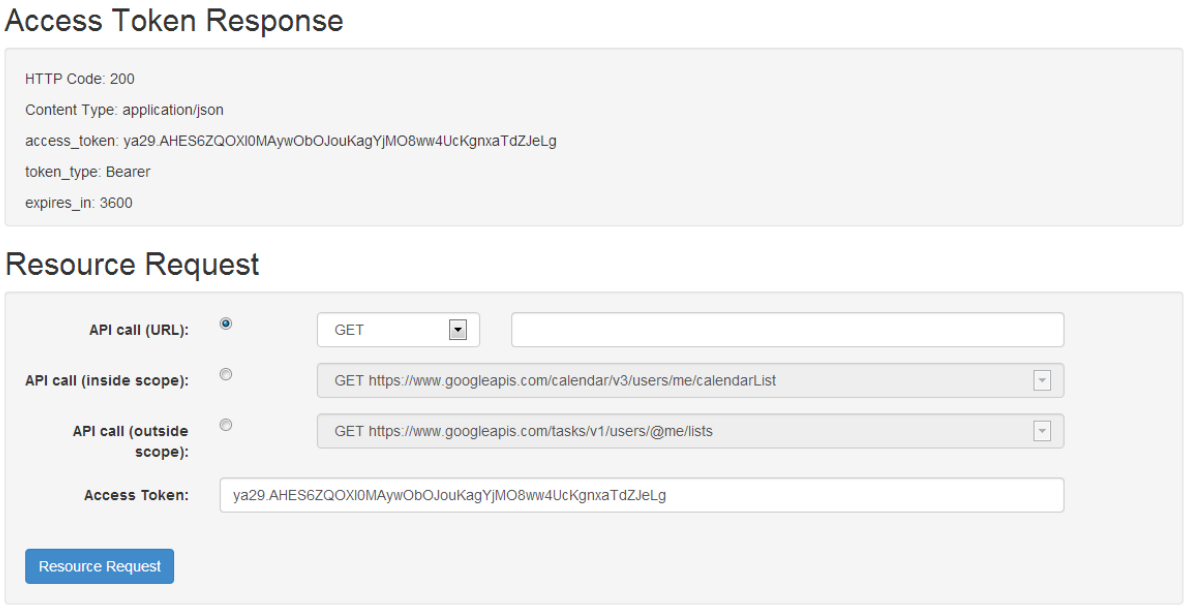

Abbildung 4.5.: Test Client Anwendung - Access Token Response und Resource Request

4.3. Untersuchungsaspekte

Für die manuelle Sicherheitsanalyse bei den einzelnen Dienstanbietern wurde ein Katalog an Untersuchungsaspekten definiert, welcher nachfolgend aufgeführt und erläutert wird. Zu diesen Untersuchungsaspekten gehören unter anderem allgemeine Protokolleigenschaften, Verhalten des Protokolls auf bestimmte Ereignisse sowie die Auswirkungen von Änderungen an bestimmten Protokollparametern. Für jeden Untersuchungsaspekt wird aufgezeigt, warum dieser Aspekt untersucht wird und in der Regel auch welche Auswirkungen er auf die Sicherheit des OAuth 2.0 Protokolls besitzt. Weiterhin werden zu jedem Untersuchungsaspekt die entsprechenden Merkmale innerhalb des OAuth Protokolls, welche es in diesem Zusammenhang bei den einzelnen Dienstanbietern zu untersuchen gibt, in Tabellenform dargestellt. Es wird auch aufgeführt, mit welcher Maßnahme diese Merkmale zu erheben sind. Dabei wird zwischen drei Herkünften unterschieden: die Analyse des Netzwerkverkehrs, die Dokumentation des Dienstanbieters und die Registrierung der Client Anwendung beim Dienstanbieter.

Hauptsächlich bezieht sich der Katalog auf den Authorization Code Grant Type, da dieser im Regelfall mindestens von den Anbietern unterstützt wird. Da die Parameter und das Verhalten beim Implicit Grant Type ähnlich bzw. gleichzusetzen mit dem Authorization Code Grant Type sind, können die Untersuchungsaspekte auch auf den Implicit Grant Type übertragen werden. Weiterhin ist zu beachten, dass nicht jeder Untersuchungsaspekt bei jedem Dienstanbieter bestimmt werden kann, da es sich letztendlich um eine Blackbox Untersuchung handelt sowie nur auf vorhandene Daten innerhalb der Dokumentation des jeweiligen Anbieters zurück gegriffen werden kann.

Untersuchungsaspekt 1 - Grant Types

Innerhalb der OAuth Spezifikation sind vier verschiedene Grant Types sowie die Möglichkeit zur eigenen Erweiterung der Grant Types dokumentiert. Jedoch wird nicht jeder Grant Type von dem einzelnen Dienstanbieter implementiert.

Untersuchungsaspekt	**Beschreibung**	**Herkunft**
Grant Types		
Grant Types	Welche Grant Types werden vom Anbieter implementiert?	Dokumentation

Tabelle 4.1.: Untersuchungsaspekt 1 - Grant Types

Untersuchungsaspekt 2 - Auslegung des Standards durch den Dienstanbieter

Nicht alle Vorgänge des Protokollablaufes werden im OAuth 2.0 Standard genau festgelegt, sondern lassen Freiräumen zur Auslegung durch die Dienstanbieter.

Untersuchungsmerkmal	**Beschreibung**	**Herkunft**
Authorization Request		
Implementierung		
Abweichende Parameter vom Standard	Werden durch den Anbietern weitere Parameter eingesetzt?	Dokumentation
Anforderung Refresh Token	Wird standardmäßig ein Refresh Token mit gesendet bzw. in welcher Weise kann ein Refresh Token angefragt werden ?	Dokumentation
Authorization Response		
Authentication Code		
Lebensdauer	Wie lange ist die Lebensdauer des Authorization Codes?	Dokumentation
Access Token Request		
Authentifikation		
Methode	Mit welcher Methode kann sich der Client beim Authorization Server authentifizieren?	Dokumentation
Access Token Response		
Access Token		
Übermittlungsformat	In welchem Format wird der Access Token übermittelt?	Analyse
Felder	Welche Felder werden übermittelt?	Analyse
Gültigkeitsdauer	Wie lange ist die Gültigkeitsdauer des Access Tokens?	Analyse, Dokumentation
Refresh Token		
Gültigkeitsdauer	Wie lange ist die Gültigkeitsdauer des Access Tokens?	Analyse, Dokumentation
Erneuerung des Refresh Tokens	Wird bei Aktualisierung eines Access Tokens mittels eines Refresh Tokens auch ein neuer Refresh Token mit gesendet?	Analyse
Ungültigkeit alter Tokens	Werden durch die Erneuerung alte Tokens (Access Token bzw. Refresh Token) ungültig?	Analyse
API Call		
Access Token		
Übermittlungsmethode	Welche Methoden werden zur Übermittlung des Access Tokens angeboten?	Dokumentation

Tabelle 4.2.: Untersuchungsaspekt 2 - Auslegung des Standards durch den Dienstanbieter

Untersuchungsaspekt 3 - Absicherung des Kommunikationskanals durch den Einsatz von TLS

Die Sicherheit des OAuth 2.0 Protokolls basiert, wie in Kapitel 2 erläutert wurde, auf der Absicherung des Kommunikationskanals durch den Einsatz von TLS. Dementsprechend gilt es zu überprüfen, ob alle Endpunkte nur mittels HTTPS oder aber auch mittels HTTP, d.h. ohne den Einsatz von TLS, erreicht werden können. Wird keine Verschlüsselung eingesetzt, ist es letztendlich möglich die Kommunikation abzuhören und so unter Umständen leichter in den Besitz eines Authorization Codes oder eines Access Tokens zu gelangen.

Untersuchungsmerkmal	**Beschreibung**	**Herkunft**
Authorization Request		
Authorization Endpoint		
URL	URL des Authorization Endpoints	Dokumentation
Erreichbarkeit über HTTP	Ist der Authorization Endpoint über HTTP ohne TLS Verschlüsselung zu erreichen?	Analyse
HTTP Methoden	Welche HTTP Methoden werden bei Anfragen an den Authorization Endpoint unterstützt bzw. sind erlaubt?	Analyse
Redirection URI		
Registrierung einer HTTP URL	Ist es möglich eine HTTP URL zu registrieren?	Registrierung
Weiterleitung an HTTP URI bei Nichtregistrierung	Ist es möglich eine HTTP Redirection URI zu übersenden, wenn keine Registrierung der Redirection URI verlangt ist?	Analyse
Access Token Request		
Token Endpoint		
URL	URL des Token Endpoints	Dokumentation
Erreichbarkeit über HTTP	Ist der Token Endpoint über HTTP ohne TLS Verschlüsselung zu erreichen?	Analyse

Tabelle 4.3.: Untersuchungsaspekt 3 - Absicherung des Kommunikationskanals durch den Einsatz von TLS

Untersuchungsaspekt 4 - Änderung der Redirection URI

Eine Änderung des Redirection URI Parameters hat schwerwiegende Auswirkungen auf den Protokollverlauf. Im eigentlichen Sinn des Protokolls soll der User Agent des Resource Owner mittels dem Redirection URI Parameter im Anschluss an die Autorisierung zurück an den Client geleitet werden. Handelt es sich bei der geänderten Redirection URI um eine vom Angreifer kontrollierte Webseite, so wird im Fall

des Authorization Code Grant Types der Authorization Code an den Angreifer gesendet. Die Möglichkeit zur Änderung des Redirection URI Parameters kann auf zwei Ausgangssituationen basieren. Erstens der Authorization Server validiert die Redirection URI nicht korrekt, so dass eine geänderte URL als gültig erkannt wird oder zweitens es wurde keine Redirection URI registriert, so dass eine beliebige URL übermittelt werden kann.

Beim Implicit Grant Type sind die Ausmaße der Weiterleitung gravierender, da innerhalb der Weiterleitung der Access Token mit gesendet wird. Da der Besitzer des Access Tokens nicht auf den implementierten Funktionsumfanges einer Client Anwendungen angewiesen ist, sondern die gesamten Funktionen der bereitgestellten API im Rahmen der Autorisierung ausnutzen und so gezielte Requests stellen kann.

Untersuchungsmerkmal	**Beschreibung**	**Herkunft**
Authorization Request		
Redirection URI		
Vorherige Registrierung	Ist die vorherige Registrierung der Redirection URI notwendig?	Registrierung
Erforderlichkeit	Ist die Angabe des Parameters redirect_uri notwendig?	Analyse
Striktheit der Überprüfung	In welcher Weise wird die angegebene URL mit der vorher registrierten URL verglichen?	Analyse
Domain des Dienstanbieters	Ist es möglich die Domain bzw. Subdomain des Dienstanbieters als Redirection URI zu übersenden?	Analyse
Access Token Request		
Redirection URI		
Änderbarkeit	Ist es möglich den Redirection URI Parameter abzuändern?	Analyse

Tabelle 4.4.: Untersuchungsaspekt 4 - Änderung der Redirection URI

Untersuchungsaspekt 5 - Änderung des Response Types

Eine Änderung des Response Type ist unter der Voraussetzung, dass der Dienstanbieter weitere Grant Types implementiert hat, grundsätzlich beliebig möglich und hat einen Wechsel des Authorization Grant Types sowie eine Veränderung des Protokollverlaufes zur Folge. So führt eine Änderung des Parameters von dem Wert „code“ auf den Wert „token“ dazu, dass ein Access Token anstatt des Authorization Codes direkt an den angegebenen Redirection Endpoint gesendet wird. Da der Access Token nun nicht mehr serverseitig verarbeitet wird, besteht die Chance für einen Angreifer, durch hier nicht näher beschriebene Angriffe wie Cross Site Scripting [38], in den Besitz des Access Tokens zu gelangen und somit uneingeschränkten Zugriff auf die autorisieren Ressourcen zu bekommen.

Untersuchungsmerkmal	Beschreibung	Herkunft
Authorization Request		
Response Type		
Erforderlichkeit	Ist die Angabe des response_type zwingend erforderlich oder gibt es einen Standard Grant Type?	Analyse, Dokumentation
Änderbarkeit	Kann vom Authorization Code Grant Type (response_type=code) zum Implicit Grant Type (response_type=token) durch Abänderung des Response Type Parameters gewechselt werden?	Analyse, Dokumentation

Tabelle 4.5.: Untersuchungsaspekt 5 - Änderung des Response Types

Untersuchungsaspekt 6 - Komplexität der Client ID

Obwohl die Client ID eine öffentlich zugängliche Information ist und infolgedessen nicht besonders abgesichert sein muss, sollte sie eine gewisse Komplexität ausweisen. Dieses begründet sich dadurch, dass ein Angreifer ansonsten durch automatisiertes Ausprobieren aller möglichen ID Werte (z.B. wenn die Client Id nur aus einer Zahlenfolge besteht) alle Client Anwendungen des jeweiligen Dienstanbieters für seine Tests nutzen kann [8].

Untersuchungsmerkmal	Beschreibung	Herkunft
Authorization Request		
Client ID		
Komplexität	Welche Komplexität besitzt die Client ID?	Registrierung

Tabelle 4.6.: Untersuchungsaspekt 6 - Komplexität der Client ID

Untersuchungsaspekt 7 - Änderung des Scopes

Die Berechtigungsfelder für die Client Anwendung sind im Vorfeld nicht festgelegt, sondern werden über den Scope Parameter innerhalb des Authorization Requests angegeben. Daher führen Änderungen an diesem Parameter zu einer Reduzierung bzw. Erweiterungen des Berechtigungsumfanges für den Client. Da der Resource Owner während der Autorisierung diesem Umfang zustimmt, besitzt er eine Kontrollfunktion über diesen Parameter. Jedoch ist anzumerken, dass der Resource Owner nicht die Kenntnis darüber besitzt, welche Rechte der Client zur Ausführung wirklich benötigt. Ferner würde er, um die Anwendung nutzen zu können, auch einen Vollzugriff auf seine Daten gewähren und somit allen Scopes zustimmen.

Es ist beim Authorization Code Grant Type zu beachten, dass der Zugriff auf die Ressourcen durch eine serverseitige Anwendung durchgeführt wird. Dementsprechend kann nur mittels der implementierten Funktionen des Clients auf die entsprechenden Ressourcen zugegriffen werden. Eine Erweiterung des Berechtigungsumfanges kann daher dazu führen, dass trotzdem nicht auf die zugehörigen Ressourcen zugegriffen werden kann.

Untersuchungsmerkmal	Beschreibung	Herkunft
Authorization Request		
Scope		
Anzahl	Wie viele verschiedene Scopes werden vom Anbieter angeboten?	Dokumentation
Unterteilte Zugriffsberechtigungen	Werden Scopes auch genutzt um Zugriffsberechtigungen zu unterteilen (z.B. in readonly / write) ?	Dokumentation
API Call		
Scope		
Zugriff außerhalb des autorisierten Scopes	Ist es möglich auf Ressourcen, welche eigentlich außerhalb des Scopes liegen, zu zugreifen?	Analyse

Tabelle 4.7.: Untersuchungsaspekt 7 - Nutzung und Änderung des Scopes

Untersuchungsaspekt 8 - Einsatz des State Parameters

Damit nicht willkürliche Anfragen an den Redirection Endpoint gesendet werden können, sollte der Client innerhalb des Authorization Requests den State Parameter nutzen, um einen beliebigen Wert festzusetzen, welcher beim Authorization Response mit zurückgesendet wird. Stimmen beide Werte des State Parameters überein, so weiß der Client, dass die Anfrage an die Redirection Endpoint zu seinem vorher gestellten Authorization Request gehört.

Untersuchungsmerkmal	Beschreibung	Herkunft
Authorization Request		
State		
Erforderlichkeit	Ist die Angabe des state Parameters erforderlich bzw. vorgeschrieben?	Analyse, Dokumentation

Tabelle 4.8.: Untersuchungsaspekt 8 - Einsatz des State Parameters

Untersuchungsaspekt 9 - Fehlermeldungen

Der Resource Owner sollte im Fall von Fehlern im Protokollverlauf ausreichend informiert werden. Dazu hat der OAuth Standard entsprechende Fehlermeldungen spezifiziert, welche jedoch durch die Dienstanbieter erweitert werden können. Diese Fehlermeldungen können durch einen Angreifer genutzt werden, um sich einen Überblick zu verschaffen, welches Fehlverhalten durch die jeweilige Dienstanbieter identifiziert werden. Dementsprechend können diese Fehlermeldungen auch Ausgangspunkte für Angriffe darstellen. So kann durch Gegenüberstellung eines herbei geführten Fehlerfalles und der zurückgelieferter Fehlermeldung überprüft werden, ob die Fehlererkennung richtig implementiert ist. Weiterhin können dadurch Lücken aufgedeckt werden, wenn keine Fehlermeldung zurück gesendet wird, obwohl es sich um einen Fehlerfall handelt.

Untersuchungsmerkmal	Beschreibung	Herkunft
Error Response		
Error		
Definition	Werden in der Entwickler Dokumentation Error Meldungen angegeben? Können diese als Hinweise dienen, welche fehlerhaften Anfragen der Anbieter erkennt und herausfiltert?	Dokumentation

Tabelle 4.9.: Untersuchungsaspekt 9 - Fehlermeldungen

Untersuchungsaspekt 10 - Erneute Autorisierungsanfrage

Bei diesem Aspekt ist zu untersuchen, wie sich der Authorization Server bei einem erneuten Authorization Request mit den gleichen Parametern d.h. insbesondere mit gleichem Scope verhält. Da eine Autorisierung des Resource Owners in diesem Fall schon vorliegt, kann es sein, dass der Authorization Server eine andere Handlungsweise zu Grunde legt.

Untersuchungsmerkmal	Beschreibung	Herkunft
Authorization Request		
Implementierung		
Authentifizierung des Nutzers	Wie wird der Nutzer bei erneuten Autorisierungsanfragen authentifiziert? D.h. muss sich der Nutzer bei jeder Anfrage neu authentifizieren?	Analyse
Autorisierung bei jedem Authorization Request	Muss der Resource Owner bei jeder Autorisierungsanfrage zustimmen oder wird dieses automatisiert?	Analyse, Dokumentation
Access Token Response		
Access Token		
Ausstellung eines neues Access Token	Wird bei einem erneuten Authorization Request ein neuer Access Token ausgestellt?	Analyse
Gültigkeit eines alten Access Token	Bleibt bei einem erneutem Authorization Request, wenn ein neuer Access Token ausgestellt wird, der alte Access Token weiterhin gültig?	Analyse

Tabelle 4.10.: Untersuchungsaspekt 10 - Erneute Autorisierungsanfrage

Untersuchungsaspekt 11 - Bindung des Authorization Codes

Innerhalb des Authorization Response wird der Authorization Code, welcher die Autorisierung des Resource Owners wiedergibt, an den Client gesendet. Für den Client ist dieser Authorization Code ein zufälliger String und steht gemäß dem OAuth 2.0 Standard in keinem Zusammenhang zu anderen vorher ausgetauschten Parametern. Da der Client den Authorization Code im nächsten Schritt gegen einen Access Token eintauscht, muss es sich lediglich um einen gültigen Authorization Code handeln. Ansonsten kann dem Client innerhalb des Authorization Responses ein beliebiger Authorization Code übermittelt

werden. Aufgrund dessen, dass sich der Resource Owner nicht gegenüber dem Client authentifiziert, besteht für den Client auch keine Möglichkeit zu überprüfen, ob der Benutzer der Client Anwendung derselbe ist, wie der Resource Owner, welche die Client Anwendung autorisiert hat. Das bedeutet in diesem Zusammenhang, dass wenn der Client einen gültigen Authorization Code einer anderen Person erhält und diesen gegen einen Access Token eintauschen kann, dann kann der Benutzer der Client Anwendung auf die autorisierten Ressourcen der anderen Person zu greifen. Jedoch ist hierbei zu beachten, dass nur im Umfang der implementierten Client Funktionalität auf die Ressourcen zu gegriffen werden kann.

Untersuchungsmerkmal	**Beschreibung**	**Herkunft**
Authorization Reponse		
Authorization Code		
Änderbarkeit	Wird der Authorization Code gegen Änderungen geschützt?	Analyse

Tabelle 4.11.: Untersuchungsaspekt 11 - Bindung des Authorization Codes

Untersuchungsaspekt 12 - Wiederholtes Übersenden des Authorization Codes

Werden Access Token Requests, welche wiederholt gleiche Authorization Codes übersenden, nicht durch den Authorization Server erkannt, so wird für jede Übersendung ein neuer Access Token ausgestellt. Dieses bedeutet, dass eine Autorisierung mehrere Zugriffsmöglichkeiten mittels unterschiedlichen Access Tokens zur Folge hat. Die Kontrolle über die Verwendung und deren Ausmaße wird dadurch erschwert.

Untersuchungsmerkmal	**Beschreibung**	**Herkunft**
Access Token Request		
Code		
Mehrmalige Verwendung	Ist es möglich mittels eines Authorization Codes mehrere Access Tokens anzufragen?	Analyse
Gültigkeit alter Tokens	Falls mit einem Authorization Code mehrere Access Tokens angefragt werden können, werden die älteren Access Tokens ungültig oder sind diese zur weiteren Nutzung gültig?	Analyse

Tabelle 4.12.: Untersuchungsaspekt 12 - Wiederholtes Übersenden des Authorization Codes

Untersuchungsaspekt 13 - Bindung des Access Token an den Client

Innerhalb des Resource Requests wird der Access Token als Repräsentant der Resource Owner Autorisierung übersendet. Der Resource Server überprüft, ob es sich um einen gültigen Access Token handelt und sendet bei einer gültigen Validierung die entsprechenden Ressourcen zurück an den Client. Jedoch kann der Resource Server im Fall eines Bearer Tokens nicht überprüfen, ob es sich dabei um den vom

Resource Owner autorisierten Client handelt. Dieses bedeutet, dass jemand der im Besitz eines Access Tokens ist, diesen unabhängig von der vorherigen Teilnahme im OAuth Protokoll einsetzen kann, um die autorisierten Ressourcen des Resource Owners zu erhalten. Daher ist zu untersuchen, welcher Token Typ von den Dienstanbietern verwendet wird.

Untersuchungsmerkmal	**Beschreibung**	**Herkunft**
Access Token Response		
Token		
Typ	Von welchem Typ (z.B. Bearer) ist der Access Token?	Analyse, Dokumentation

Tabelle 4.13.: Untersuchungsaspekt 13 - Bindung des Access Token an den Client

Untersuchungsaspekt 14 - Widerrufung von Tokens

Der Authorization Server sollte eine Möglichkeit bieten, dass Token, welche eigentlich noch gültig sind, widerrufen werden können. Hierbei ist zu untersuchen, wie diese Möglichkeit implementiert ist.

Untersuchungsmerkmal	**Beschreibung**	**Herkunft**
Revoke Token		
URL	Gibt es eine URL zur Widerrufung von Tokens?	Dokumentation
Parameter	Welche Parameter müssen übermittelt werden?	Dokumentation
HTTP Methode	Welche HTTP Methoden werden unterstützt?	Dokumentation
Zusammenhang zwischen den Tokens	Reicht es einen der beiden Tokens (Access Token und Refresh Token) zu widerrufen oder müssen beide Tokens widerruft werden?	Analyse

Tabelle 4.14.: Untersuchungsaspekt 14 - Widerrufung von Tokens

4.4. Identifizierung und Auswahl von Anbietern

Als Quellen für die Identifizierung von Dienstanbietern, welche das OAuth 2.0 Protokoll implementiert haben, wurde auf mehrere Webseiten zurückgegriffen:

- *Wikipedia.org* [51]: Innerhalb des englischen Artikels zu OAuth wird eine Liste von Dienstanbietern mit der implementierten OAuth Version aufgeführt.
- *Webseite des Bloggers Namachivayam* [52]: Auf seiner Webseite präsentiert Namachivayam eine „oAuth light-weight class demo page“. Dazu bietet er für eine Vielzahl von Dienstanbietern jeweils eine Demo Client Anwendung an. Weiterhin beschreibt er für einen Teil der aufgeführten Dienstanbieter in einer Schritt-für-Schritt Anleitung die Registrierung und Einrichtung einer Client-Anwendung, sowie gibt er deren Endpunkte an.

- *OAuth.io* [53] [54]: Oauth.io bietet für Entwickler eine einheitliche API zur Nutzung einer großen Anzahl an OAuth Dienstanbietern Implementationen (80+). Die Nutzung des Services bietet dem Entwickler den Vorteil, dass er so nicht jede einzelne SDK Entwicklerdokumentation eines Dienstanbieters lesen muss, um an einen Access Token zu gelangen.

Bei allen ausgewählten Anbietern wurde durch Kontrolle der aktuellsten Entwicklerdokumentation verifiziert, dass sie das OAuth 2.0 Protokoll implementiert haben. Bei der Auswahl der Dienstanbieter wurde Wert auf eine Mischung der angebotenen Funktionalität sowie auf die Größe des Dienstanbieters und der damit wahrscheinlich einhergehenden Größe des Entwicklerteams gelegt.

Name	**Url**	**Beschreibung**	**Developer URL**
Box	`https://www.box.com/`	Cloud Speicher	`http://developers.box.com/oauth/`
Cheddar	`https://cheddarapp.com/`	Task List	`https://cheddarapp.com/developer/authentication`
Dailymile	`http://www.dailymile.com`	Workout Tracking	`http://www.dailymile.com/api/documentation/oauth`
Dailymotion	`http://www.dailymotion.com/`	Video Plattform	`http://www.dailymotion.com/doc/api/authentication.html`
Facebook	`http://www.facebook.com`	Soziales Netzwerk	`https://developers.facebook.com/docs/facebook-login/`
Foursquare	`https://foursquare.com/`	Soziales Netzwerk	`https://developer.foursquare.com/overview/auth`
Google	`http://www.google.com`	Suchmaschine	`https://developers.google.com/accounts/docs/OAuth2`
Salesforce	`http://www.force.com/`	Cloud Plattform	`http://help.salesforce.com/apex/HTViewHelpDoc?id=remoteaccess_authenticate.htm`
Stackexchange	`https://stackexchange.com`	FAQ Plattform	`http://api.stackexchange.com/docs/authentication`

Tabelle 4.15.: Ausgewählte Dienstanbieter

4.5. Ergebnisse der Sicherheitsanalyse

Grundsätzlich kann festgehalten werden, dass das OAuth 2.0 Protokoll sehr unterschiedlich von den Dienstanbietern implementiert wird. Eine Übersicht der Ergebnisse der manuellen Sicherheitsanalyse wird in der Tabelle 4.16 präsentiert.

	Box	**Cheddar**	**Dailymile**	**Dailymotion**	**Facebook**	**Foursquare**	**Google**	**Salesforce**	**Stackexchange**
Grant Types	AC	AC	AC, I	AC, I, ROPC	AC, I, CC	AC, I	AC, I, JWT, DF	AC, I, JWT, SAML, ROPC	AC, I
Anforderung Refresh Token	autom.	-	-	autom.	-	-	Param.	Scope	-
Registrierung Redirection URI / HTTP	optional / nein	ja / ja	optional / ja	ja / ja	ja / ja	ja / ja	ja / ja	ja / nein	ja / ja
Änderbarkeit der Redirection URI im Authorization Request	Hinzu. Subdomain + TLD	Hinzu. TLD	nein	Hinzu. TLD	Hinzu. Subdomain	Hinzu. TLD	nein	nein	Hinzu Subdomain
Änderbarkeit der Redirection URI im Access Token Request	ja	ja	nein	nein	nein	ja	nein	nein	nein
Änderbarkeit Response Type	-	-	ja	ja	ja	ja	ja	ja	nein
Unzureichende Client ID Komplexität	nein	nein	nein	nein	ja	nein	nein	nein	ja
Anzahl Scopes	0	0	0	10	100+	0	100+	7	4
Nutzung State Parameter	optional	optional	optional	optional	optional	optional	optional	optional	optional
Automatik bei erneuter Autorisierung	nein	nein	nein	nein	ja	nein	ja	ja	nein
Mehrmaliges Übersenden des Authorization Codes	ja	nein	nein	nein	nein	nein	nein	nein	ja
Gültigkeitsdauer Access Token	1h	n/a	n/a	10h	60d	∞	1h	n/a	1d
Token Revokation	ja	-	-	ja	-	-	ja	ja	-

Tabelle 4.16.: Ergebnissübersicht der manuellen Analyse
Legende: AC = Authorization Code Grant Type, I = Implicit Grant Type, ROPC = Resource Owner Password Credential Grant Type, CC = Client Credential Grant Type, JWT = JWT Bearer Token Grant Type, SAML = SAML 2 Bearer Assertion Grant Type, DF = Device Flow, Param. = Parameter, Hinzu. = Hinzufügen von, TLD = Top-Level Domain

Zwar implementieren alle Anbieter den Authorization Grant Type, jedoch in den unterschiedlichsten Ausführungen. So werden Refresh Tokens teilweise automatisch erstellt, teilweise müssen sie extra angefordert werden oder werden auch gar nicht eingesetzt. Die Gültigkeitsdauern der ausgestellten Access Token variieren sehr stark. Bei manchen Anbietern sind sie nur eine Stunde gültig, bei anderen 60 Tage oder sogar unendlich lang gültig. Die Möglichkeit Scopes zu verwenden, um eine umfangreiche Funktionstrennung zu erreichen, wird nur von wenigen Anbietern eingesetzt. Kritisch ist jedoch der Umgang mit der Redirection URI zu betrachten. Bei einem Großteil der Anbieter war es möglich eine HTTP Redirection URI zu registrieren. Weiterhin lassen sich teilweise registrierte Redirection URIs um Subdomains oder sogar um Top-Level Domain Teile erweitern. Demzufolge lässt sich die Verifikation der registrierten Redirection URI umgehen, in dem die registrierte URL `https://domain.de` mit einer beliebigen Top-Level Domain z.B. `https://domain.de.to` erweitert und so der Anwender auf eine beliebige Webseite weitergeleitet wird. Weiterhin wird bei allen Dienstanbietern der Authorization Code nicht vor Änderungen geschützt, so dass ein Angreifer einen abgefangenen Authorization Code als seinen eigenen ausgeben kann und so auf die autorisierten Ressourcen des Opfers zu greifen kann.

Im anschließenden Teil werden die Ergebnisse der manuellen Sicherheitsanalyse bei den einzelnen Dienstanbietern präsentiert. Es werden dabei jedoch nur die charakteristischen Eigenschaften sowie analysierte Schwachstellen, welche sich auf den Authorization Code Grant Type beziehen, der jeweiligen OAuth Implementierung des Dienstanbieters entsprechend der Untersuchungsaspekte aufgezeigt. Des weiteren werden Referenzen auf die Sicherheitsempfehlungen- und vorkehrungen (siehe Kapitel 3.5) gegeben. Alle erhobenen Daten der Sicherheitsanalyse lassen sich im Anhang A einsehen.

Box

UA 1: Es wird lediglich der Authorization Code Grant Type unterstützt.

UA 2: Im Access Token Response wird automatisch ein Refresh Token mit gesendet, welcher vierzehn Tage gültig ist. Der Access Token ist nur eine Stunde gültig.

UA 4: Die vorherige Registrierung einer Redirection URI für eine Client Anwendung ist optional. Wird keine Redirection URI registriert, so kann ein Angreifer eine beliebige Redirection URI während des Authorization Requests setzten. Ist jedoch eine registriert, so wird bei der Überprüfung der Redirection URI lediglich die gespeicherte URL als Präfix der angegeben URL verglichen. Das bedeutet, dass die gespeicherte URL als Subdomain einer anderen Domain als gültig verifiziert wird. Einen Angreifer ermöglicht dies, die Redirection URI zu verändern und so den Protokollfluss zu steuern.

UA 7: Scopes werden durch den Dienstanbieter nicht definiert (Nichtbeachten von SR 7). Erhält ein Angreifer einen Access Token eines Opfers, so kann er ohne Einschränkung auf alle Daten des Opfers zugreifen.

UA 10: Bezüglich der Authentifikation des Resource Owner gegenüber dem Authorization Server

wird nicht auf eine bestehende Session zurückgegriffen, sondern der Resource Owner muss sich bei jeder Autorisierungsanfrage neu authentifizieren.

UA 12: Die Gültigkeit des Authorization Codes ist auf 30 Sekunden beschränkt, jedoch lässt sich der Authorization Code in dieser Zeit nutzen, um mehrmalig einen Access Token anzufordern. Alle ausgestellten Access Token sind gültig und können für den Zugriff auf autorisierte Ressourcen genutzt werden (Verletzung von SR 6).

UA 14: Es wird ein Endpunkt bereit gestellt, an dem Tokens widerruft werden können. Es genügt einen Token (Access Token oder Refresh Token) zu übersenden, um beide zu widerrufen.

Cheddar

UA 1: Es wird lediglich der Authorization Code Grant Type unterstützt.

UA 4: Eine Redirection URI muss im Vorfeld registriert werden. Jedoch ist es dabei möglich eine HTTP URL zu registrieren (Nichtbeachtung von SR 1). Dieses führt wiederum dazu, dass der Authorization Code über eine unverschlüsselte Kommunikationsverbindung übertragen wird.

UA 4: Der Redirection URI Parameter ist innerhalb des Authorization Request nicht erforderlich. Wird er jedoch übertragen, so wird bei der Überprüfung der Redirection URI lediglich die gespeicherte URL als Präfix der angegeben URL verglichen. Das bedeutet, dass die gespeicherte URL als Subdomain einer anderen Domain als gültig verifiziert wird. Ein Angreifer kann dieses ausnutzen, um in den Besitz eines Authorization Codes zu gelangen. Da innerhalb des Access Token Request die Redirection URI nicht validiert wird, kann der Angreifer den abgefangenen Authorization Code gegen einen Access Token eintauschen (siehe Kapitel 4.6.1).

UA 7: Scopes werden durch den Dienstanbieter nicht definiert (Nichtbeachten von SR 7). Erhält ein Angreifer einen Access Token eines Opfers, so kann er ohne Einschränkung auf alle Daten des Opfers zugreifen.

Dailymile

UA 1: Es werden sowohl der Authorization Code als auch der Implicit Grant Type unterstützt.

UA 3-4: Die vorherige Registrierung einer Redirection URI für eine Client Anwendung ist optional. Wird keine Redirection URI registriert, so kann ein Angreifer eine beliebige Redirection URI während des Authorization Requests setzten. Es ist sowohl möglich eine HTTP URL zu registrieren als auch im Fall der Nichtregistrierung eine HTTP URL innerhalb des Redirection URI Paramaters anzugeben (Nichtbeachtung von SR 1). Dies führt dazu das sensible Daten (wie der Authorization Code oder Access Token) unverschlüsselt übertragen werden. Wenn jedoch eine Redirection URI registriert wurde, wird diese strikt überprüft.

UA 7: Scopes werden durch den Dienstanbieter nicht definiert (Nichtbeachten von SR 7). Erhält ein Angreifer einen Access Token eines Opfers, so kann er ohne Einschränkung auf alle Daten des Opfers zugreifen.

Dailymotion

UA 1: Es werden neben dem Authorization Code Grant Type auch der Implicit und der Resource Owner Password Credential Grant Type unterstützt.

UA 2: Ein Refresh Token wird automatisch im Access Token Response mit gesendet. Die Gültigkeitsdauer des Access Token ist auf 10 Stunden begrenzt.

UA 3-4: Eine Redirection URI muss zwar im Vorfeld registriert werden, jedoch ist es dabei möglich seine HTTP URL anzugeben (Nichtbeachtung von SR 1). Bei der Überprüfung der Redirection URI während des Authorization Requests wird lediglich geprüft, ob die übermittelten URL mit der gespeicherten URL beginnt. Dieses kann durch einen Angreifer ausgenutzt werden, um den Auhtorization Code bzw. den Access Token an eine beliebige Webseite zu leiten.

UA 7: Der Anbieter definiert zehn verschiedene Scopes, welche hauptsächlich für die Funktionstrennung eingesetzt werden.

UA 10: Im Gegensatz zu anderen Anbietern muss sich der Resource Owner bei jeder Autorisierungsanfrage gegenüber dem Authorization Server authentifizieren.

UA 14: Zur Widerrufung von Tokens bietet Dailymotion einen entsprechenden Endpunkt an. Die Übermittlung des Access Token führt auch zu einer Widerrufung des Refresh Tokens.

Facebook

UA 1: Facebook bietet neben dem Authorization Code Grant Type und dem Implicit Grant Type auch eine Mischung von beiden Verfahren an. Als Response Type Parameter wird dabei der Wert „code%20token“ übermittelt. Dieses führt dazu, dass neben dem Access Token auch ein Authorization Code innerhalb des URL Fragments direkt als Authorization Response an den Client gesendet wird. Weiterhin hat Facebook zur Wartung von Anwendungen den Client Credential Grant Type implementiert.

UA 2: Die Gültigkeitsdauer eines Access Token beträgt 60 Tage. Die Nutzung von Refresh Tokens wurde von Facebook nicht implementiert.

UA 4: Neben der Übermittlung der registrierten Redirection URI, welche um Subdomains erweitert werden kann, ist auch die Angabe der durch Facebook erzeugten Canvas Seite `https://apps.facebook.com/<app-name>` möglich.

UA 6: Die Client ID besteht nur aus einer Zahlenfolge, so dass eine Iteration über alle Client IDs möglich ist (Verletzung von SR 4).

UA 7: Facebook bietet mit über 100 verschiedenen Scopes ein ausführliches Berechtigungskonzept.

UA 10: Hat der Resource Owner einmal einer bestimmten Scopekombination zugestimmt, muss er bei erneuten Autorisierungsanfragen nicht erneut zustimmen. Der Authorization Code bzw. Access Token wird direkt an die Client Anwendung gesendet.

Foursquare

UA 1: Es werden sowohl der Authorization Code Grant Type als auch der Implicit Grant Type unterstützt.

UA 2: Access Tokens besitzen keine Gültigkeitsbeschränkung, d.h. sie laufen nicht ab. Auch besteht für die Client Anwendung keine Möglichkeit einen Access Token zu widerrufen (Verletzung von SR 5 und SR 9).

UA 4: Im Vorfeld muss eine Redirection URI für einen Client registriert werden. Die Angabe der Redirection URI innerhalb des Authorization Requests ist nicht notwendig. Bei Übermittlung der Redirection URI wird jedoch nur überprüft, ob die übersendete URI mit der gespeicherten URI beginnt. Dies kann ein Angreifer ausnutzen, um in den Besitz des Authorization Codes bzw. Access Token zu gelangen (siehe Kapitel 4.6.2). Da die Redirection URI im Access Token Request nicht korrekt validiert wird, kann der Angreifer einen abgefangen Authorization Code gegen einen Access Token eintauschen (siehe Kapitel 4.6.1)

UA 7: Scopes werden durch den Dienstanbieter nicht definiert (Nichtbeachten von SR 7). Erhält ein Angreifer einen Access Token eines Opfers, so kann er ohne Einschränkung auf alle Daten des Opfers zugreifen.

Google

UA 1: Zusätzlich zu dem Authorization Code Grant Type und dem Implicit Grant Type hat Google für Server zu Server Anwendungen den JWT Bearer Token Grant Type implementiert. Ferner bietet Google für Geräte mit eingeschränkten Eingabemöglichkeiten (z.B. Spielekonsolen oder Kameras) einen speziellen Geräte Protokollfluss (Device Flow) an. Zum Start dieses Protokollflusses wird eine Anfrage mit der Client ID und den Scopes an einen speziellen OAuth Device Endpoint gesendet. Als Antwort erhält der Anwender eine URL und einen sogenannten User Code. Diese genannte URL muss der Anwender nun in einem separaten Browser öffnen und den User Code dort angeben. Anschließend erhält der Nutzer das gewohnte Autorisierungsformular und kann die Client Anwendung autorisieren.

UA 2: Über den Parameter *access_type=offline* innerhalb des Authorization Requests kann ein Refresh Token angefordert werden.

UA 2: Die Lebensdauer eines Access Token ist bei Google auf eine Stunde beschränkt.

UA 7: Mit über 100 verschiedenen Scopes verfügt Google über ein weitreichendes Berechtigungskonzept. Für ein Projekt (Gruppe von Client Anwendungen) müssen im Vorfeld einzelne Scope Gruppen freigeschaltet werden, so dass gezielt der Zugriff auf die Google API gesteuert werden kann.

UA 10: Default mäßig muss der Resource Owner erneuten Autorisierungsanfragen nochmals zustimmen. Jedoch kann dies durch den Parameter *approval_prompt=force* innerhalb des Authorization Requests erzwungen werden, so dass dem Resource Owner das gewohnte Autorisierungsformular angezeigt wird.

UA 14: Zur Widerrufung von Tokens bietet Google einen entsprechenden Endpunkt an. Die Übermittlung des Access Token führt auch zu einer Widerrufung des Refresh Tokens.

Salesforce

UA 1: Salesforce unterstützt die größte Menge an Grant Types. Neben dem Authorization Code und dem Implicit Grant Type, hat Salesforce den Resource Owner Password Grant Type sowie den JWT Bearer Token und SAML Bearer Assertion Grant Type implementiert.

UA 2: Ein Refresh Token kann über den Scope Wert „refresh_token“ angefordert werden.

UA 14: Zur Widerrufung von Tokens bietet Salesforce einen entsprechenden Endpunkt an. Im Gegensatz zu anderen Anbietern bleibt der Refresh Token bei Widerrufung des Access Token gültig. Jedoch wird bei Übermittlung des Refresh Token sowohl dieser als auch der Access Token für ungültig erklärt.

Stackexchange

UA 1: Es werden sowohl der Authorization Code als auch der Implicit Grant Type unterstützt. Jedoch kann durch den Client nur jeweils einer der beiden Grant Types verwendet werden, da dieser während der Registrierung festgelegt werden muss.

UA 2: Die Gültigkeitsdauer des Access Token ist auf einen Tag festgelegt.

UA 3: Eine Redirection URI muss im Vorfeld registriert werden. Jedoch wird dabei nicht geprüft, ob es sich um eine HTTPS URI handelt (Nichtbeachtung von SR 1).

UA 4: Das Hinzufügen einer Subdomain zu der registrierten Redirection URI wird als gültig verifiziert. Dadurch ist es für einen Angreifer möglich die Manipulation der Redirection URI zu

seinem Vorteil zu nutzen und so in den Besitz des Access Tokens zu gelangen (siehe Kapitel 4.6.2).

UA 6: Die Komplexität der Client ID ist sehr gering, da sie in unserem Analyse Fall nur aus einer vierstelligen Zahlenfolge bestand (Verletzung von SR 4).

UA 12: Mit dem Authorization Code ist es möglich mehrmalig einen Access Token anzufordern. Alle ausgestellten Access Token sind gültig und können für den Zugriff auf autorisierte Ressourcen genutzt werden (Verletzung von SR 6).

4.6. Angriffsszenarien

Im Folgenden werden konkrete Angriffsszenarien erläutert, welche dem definierten Angriffsmodell entsprechen und die erzielten Ergebnisse der manuellen Sicherheitsanalyse aufgreifen.

4.6.1. Angriffsszenario 1: Erhalt des Authorization Code durch Redirection URI Manipulation

Bei diesem Angriffsszenario wird der Authorization Response eines Opfers zu einer vom Angreifer kontrollierten Webseite geleitet. Der dadurch erlangte Authorization Code wird innerhalb eines neuen Protokolllaufes vom Angreifer als sein eigener ausgegeben, umso mittels der Client Anwendung auf die Ressourcen des Opfers zugreifen zu können.

Vorgehen des Angreifers

Das im Folgenden beschriebene Vorgehen bezieht sich auf den in Abbildung 4.6 dargestellten Verlauf:

Schritt 1-2 Der Angreifer lässt dem Opfer einen gefälschten Authorization Request, welcher als Redirection URI Parameter eine vom Angreifer kontrollierte Webseite besitzt, zukommen (siehe Hervorhebung in Abbildung 4.6). Die Adresse der Angreiferwebseite ist zwar als Parameter in der URL enthalten, jedoch so durch den Angreifer konstruiert, dass es dem Opfer wahrscheinlich nicht auffällt.

Schritt 3-8 Das Opfer klickt auf den Link und startet damit das OAuth Protokoll. In den folgenden Schritten autorisiert er die für ihn vertrauenswürdige Client Anwendung.

Schritt 9-10 Als Authorization Response erhält der User Agent des Resource Owners die Weiterleitung zu der Webseite des Angreifers, wodurch der Authorization Code an den Angreifer gesendet wird.

Schritt 4*-8* Der Angreifer startet einen eigenen Protokollablauf mit der gleichen Client Anwendung. Jedoch lässt er die Redirection URI unverändert.

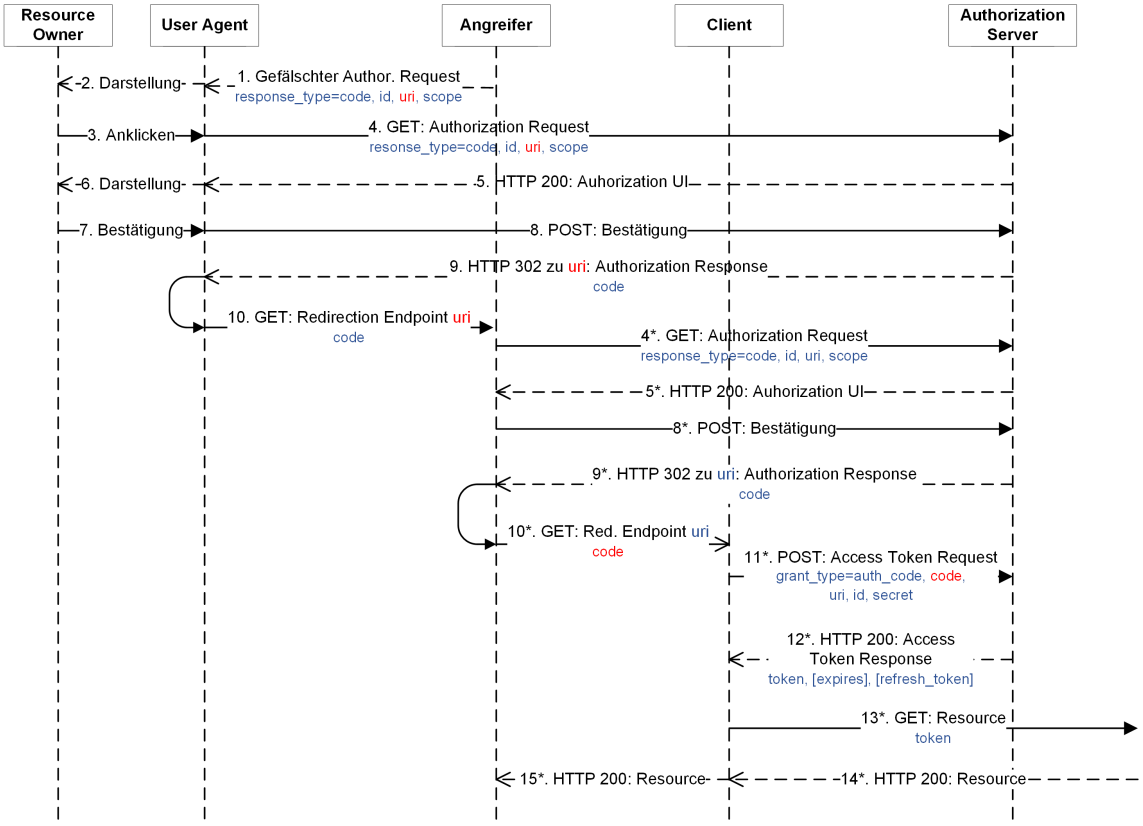

Abbildung 4.6.: Erhalt des Authorization Code durch Redirection URI Manipulation beim Authorization Code Grant Type. Aus Übersichtsgründen wurden die Schritte 6* und 7* weggelassen, sowie der Angreifer, dessen User Agent und die von ihm kontrollierte Webseite in einer Entität zusammengefasst.

Schritt 9*-10* Der Angreifer erhält als Authorization Response die Weiterleitung zur Client Anwendung. Dieser folgt er nicht direkt, sondern tauscht den Authorization Code gegen den vom Opfer übermittelten Authorization Code aus.

Schritt 11*-12* Die Client Anwendung tauscht den Authorization Code des Opfers gegen einen Access Token ein.

Schritt 13*-15* Der Angreifer kann im Rahmen der Client Funktionalitäten auf die Ressourcen des Opfers zugreifen.

Voraussetzung

Eine zwingende Voraussetzung für diesen Angriffsvektor ist, dass der Redirection URI Parameter innerhalb des Authorization Requests so durch einen Angreifer abgeändert werden kann, dass nach der Autorisierung durch das Opfer dessen User Agent an eine vom Angreifer definierte Webadresse weitergeleitet wird (siehe Sicherheitsaspekt 4.3). Eine weitere Voraussetzung ist, dass die im Access Token Request angegebene Redirection URI nur gegen die vorher registrierte Redirection URI validiert wird

und nicht gegen die während des Authorization Requests übermittelte. Denn ansonsten würde die Validierung feststellen, dass es sich aufgrund der unterschiedlichen Protokollläufe auch um unterschiedliche Redirection URIs handelt.

Gegenmaßnahmen

Unterschiedliche Gegenmaßnahmen würden dieses Angriffsszenario verhindern. So würde eine strenge Validierung der Redirection URI im Authorization Request durch den Authorization Server verhindern, dass der User Agent des Resource Owners an eine beliebige Webseite weitergeleitet wird und somit der Angreifer den Authorization Code nicht erhält. Eine einfache Überprüfung, ob die übersendete URI mit der gespeicherte URI beginnt, reicht in diesem Fall nicht aus, da ansonsten die Redirection URI als Subdomain einer anderen Domain akzeptiert werden würde. Weiterhin darf die übermittelte Redirection URI innerhalb des Access Token Request nicht nur mit der gespeicherten Redirection URI verglichen werden. Sondern sie muss, wie es in der Spezifikation beschrieben ist, mit der im Authorization Request übergebenen Redirection URI abgeglichen werden: „*[...] the authorization server MUST ensure that the redirection URI used to obtain the authorization code is identical to the redirection URI provided when exchanging the authorization code for an access token*.“ [9]

Bei diesem Angriffsszenario wird ausgenutzt, dass die Redirection URI nicht nur eine fremde URI enthalten kann, sondern

4.6.2. Angriffsszenario 2: Erhalt des Access Token durch Redirection URI Manipulation

Bei diesem Angriffsszenario wird der Authorization Response eines Opfers zu einer vom Angreifer kontrollierten Webseite geleitet. Den mit gesendeten Access Token kann der Angreifer extrahieren und für den Zugriff auf die autorisierten Ressourcen des Opfers mittels eigener Anfragen an den Resource Server nutzen. Dieses Angriffsszenario kann unter auch[1] aus einem Authorization Request, welcher auf dem Authorization Code Grant Type beruht, hervorgehen.

Vorgehen des Angreifers

Das im Folgenden beschriebene Vorgehen bezieht sich auf den in Abbildung 4.7 dargestellten Verlauf:

Schritt 1-2 Der Angreifer lässt dem Opfer einen gefälschten Authorization Request, welcher als Redirection URI Parameter eine vom Angreifer kontrollierte Webseite besitzt, zukommen (siehe Hervorhebung in Abbildung 4.7). Die Adresse der Angreiferwebseite ist zwar als Parameter in der URI enthalten, jedoch so durch den Angreifer konstruiert, dass es dem Opfer wahrscheinlich nicht auffällt.

Schritt 3-8 Das Opfer klickt auf den Link und startet damit das normale OAuth Protokoll. In den folgenden Schritten autorisiert er die Client Anwendung.

[1] unter der Bedingung, dass der Dienstanbieter einen Wechsel des Response Types zulässt

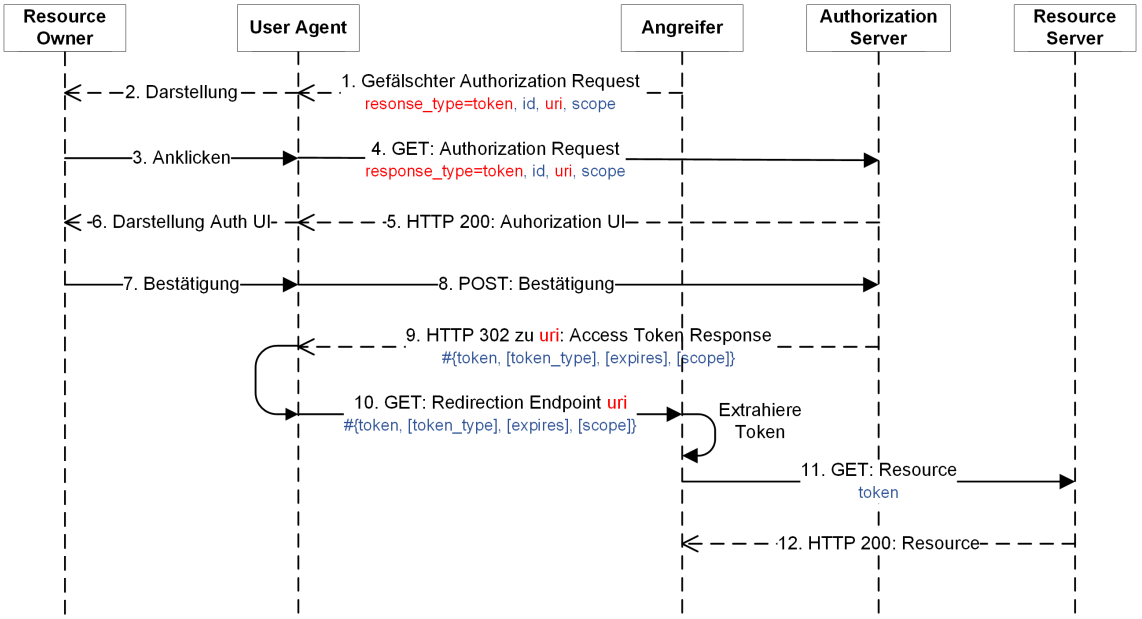

Abbildung 4.7.: Redirection URI Manipulation beim Implicit Grant

Schritt 9-10 Als Access Token Response erhält der User Agent des Resource Owners die Weiterleitung zu der Webseite des Angreifers, wodurch der Access Token an den Angreifer gesendet wird. Der Angreifer kann den Access Token mittels JavaScript extrahieren.

Schritt 11-12 Der Angreifer kann den Access Token nutzen um auf die API des Dienstanbieters zu zugreifen, um so an die autorisierten Ressourcen des Opfers zu gelangen.

Voraussetzung

Eine notwendige Voraussetzung ist, dass es sich um einen clientseitigen Protokollfluss (Implicit Grant Type) handelt. Dazu kann entweder ein vorhandener clientseitiger OAuth 2.0 Protokollfluss verwendet werden oder unter der Bedingung, dass der Anbieter sowohl den Authorization Code Grant Type als auch den Implicit Grant Type unterstützt, der Protokollverlauf durch Änderung des Response Type Parameters in einen Implicit Grant Type geändert werden. Weiterhin muss es möglich sein den Redirection URI Parameter so zu ändern, dass der User Agent des Resource Owner im Access Token Response an eine unter Kontrolle des Angreifers stehenden Webseite geleitet wird.

Gegenmaßnahmen

Implementiert ein Dienstanbieter den Implicit Grant Type, so muss er im Vorfeld bei der Registrierung der Client Anwendung verlangen, dass eine HTTPS Redirection URI festgelegt wird. Weiterhin muss während des Authorization Requests die übermittelte Redirection URI strikt gegen die gespeicherte Redirection URI validiert werden.

5. Konzept zur Durchführung von automatisierten Penetrationstests

Unter der Testautomatisierung wird der Einsatz von Softwarewerkzeugen zur Durchführung oder Unterstützung von Testaktivitäten, wie z.B. Testmanagement, Testentwurf, Testausführung und Soll/Ist-Vergleich verstanden [55].

Die Vorteile des automatisierten Testens bestehen darin, dass die Testqualität gegenüber dem manuellen Testen verbessert werden kann. Manuelle Tests, welche ständig wiederholt werden müssen, können bei einem Tester zu einer Monotonität führen, so dass auf Grund von Unkonzentriertheit Schritte nicht vollständig ausgeführt oder sogar ausgelassen werden. Bei automatisierten Tests werden die Schritte eines Tests nicht manuell, d.h. durch eine menschliche Person, sondern durch eine Maschine durchgeführt. Dieses führt nicht nur zu einer Zeitersparnis, sondern auch zu einer Erhöhung der Testgenauigkeit, da die einzelnen Schritte strikt und vollständig ausgeführt werden. Aufgrund dessen, dass die Tests immer gleich ausgeführt werden, lassen sie sich miteinander vergleichen und die Qualität messen. Dementsprechend lassen sich automatisierte Tests auch zur Qualitätsmessung einsetzen. In diesem Zusammenhang besteht auch die Möglichkeit die Tests unter gleichen Bedingungen zu wiederholen und sie so z.B. zur Kontrolle von Korrekturmaßnahmen einzusetzen. Weiterhin werden bei jedem Testdurchlauf die Ergebnisse festgehalten und erlauben so im Anschluss eine Nachverfolgung des Testergebnisses sowie die automatische Erstellung einer Testdokumentation [56].

Innerhalb dieses Kapitels wird daher ein Konzept entwickelt sowie eine prototypische Implementierung vorgestellt, wie automatisierte Tests auf das OAuth Protokoll angewendet werden können. Zuvor wird jedoch darauf eingegangen, welche Schwierigkeiten des OAuth Protokolls innerhalb des Konzepts für das automatisierte Testen berücksichtigt werden müssen.

5.1. Schwierigkeiten in Bezug auf das automatisierte Testen

Automatisiertes Testen kann, wie in der Einleitung des Kapitels, eine Verbesserung des manuellen Testens darstellen. Jedoch weisen Dustin et al. [56] darauf hin, dass nicht erwartet werden kann, dass sich alle manuellen Tests auch automatisieren lassen. Daher wird im Folgenden auf die Schwierigkeiten des automatisierten Testens des OAuth 2.0 Protokolls eingegangen.

Eine Herausforderung beim automatisierten Testen besteht in der Verarbeitung von Aktionen, die sich durch Veränderungen von Parametern ergeben. Als Beispiel sei hier das Verhalten des Authorization Servers auf die Abänderung des Redirection URI Parameters genannt. Um dieses zu verdeutlichen wurde

in einem Beispielszenario bei vier Dienstanbietern der Redirection Parameter auf die Webseite `https://heise.de` geändert. Diese Webseite ist bei allen vier Dienstanbietern nicht als Redirection URI registriert. Demnach müsste der Protokollverlauf während der Validierung des Authorization Requests anhalten und der Anwender über den Fehler informiert werden:

> *If the request fails due to a missing, invalid, or mismatching redirection URI, or if the client identifier is missing or invalid, the authorization server SHOULD inform the resource owner of the error and MUST NOT automatically redirect the user-agent to the invalid redirection URI.* [9]

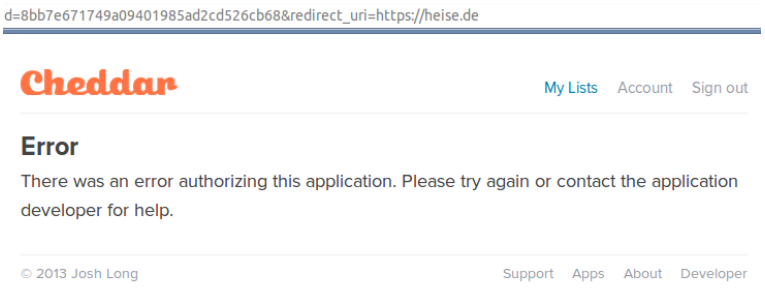

Abbildung 5.1.: Verhalten des Anbieters Cheddar bei Änderung des Redirection URI Parameters

Abbildung 5.2.: Verhalten des Anbieters Box bei Änderung des Redirection URI Parameters

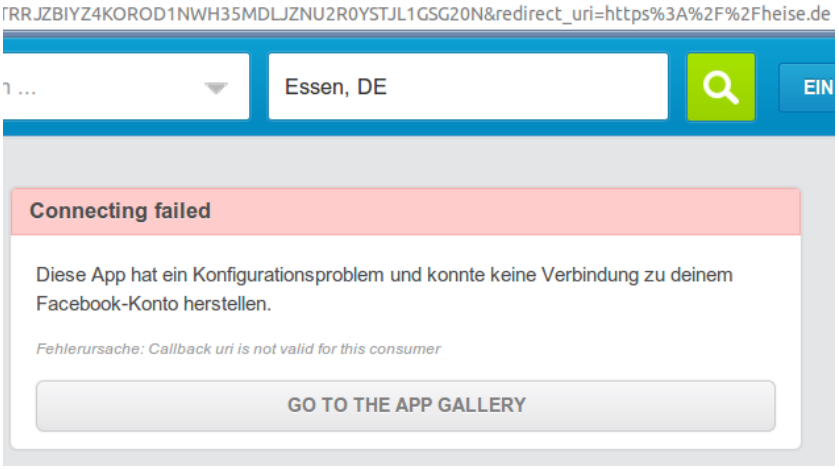

Abbildung 5.3.: Verhalten des Anbieters Foursquare bei Änderung des Redirection URI Parameters

Abbildung 5.4.: Verhalten des Anbieters Dailymotion bei Änderung des Redirection URI Parameters

Bei der Analyse war zu beobachten, dass Cheddar (Abbildung 5.1) lediglich über einen aufgetretenen Fehler informiert. Es ist dabei jedoch nicht ersichtlich, um welchen Fehler es sich handelt. Demgegenüber wird bei Box (Abbildung 5.2) deutlich genannt, dass es sich um einen Redirection URI Mismatch handelt. Bei Foursquare (Abbildung 5.3) wird zwar mit der Fehlermeldung das Gleiche ausgesagt, jedoch mit einer für OAuth unkonventionellen Sprache. Anstatt des Redirection URI Begriffs wird der Begriff Callback URI verwendet und der Client wird als Comsumer bezeichnet. Kritisch hat sich in diesem Zusammenhang Dailymotion (Abbildung 5.4) verhalten, da dieser Dienstanbieter den User Agent an die nicht registrierte und abgeänderte Redirection URI weitergeleitet hat. Die Fehlermeldung wird zwar als Parameter übertragen, kann aber auf Grund der Weiterleitung nicht angezeigt werden.

Dieses aufgezeigte Beispielszenario zeigt deutlich, welche Schwierigkeiten sich für das automatische Testen ergeben. Einer menschlichen Person ist es zu mindestens in drei Fällen (ausgenommen wird hier der Fall von Cheddar) möglich den Fehler auf die Änderung des Redirection URI Parameters zurück zuführen. Bei einem automatisierten Test müsste für jeden Anbieter das Fehlverhalten und entsprechende Kennzeichen explizit definiert werden. Dieses würde jedoch einer langfristigen manuellen Analyse vorausgehen, welches jedoch nicht gewollt ist.

Eine weitere Schwierigkeit, welche sich im Zusammenhang mit dem Testen des OAuth Protokolls ergibt, ist, dass Änderungen innerhalb des Authorization Requests durch den Resource Owner autorisiert werden müssen, um aktuelle und gültige Authorization Codes oder Access Tokens zu erhalten. Da teilweise CSRF-Schutzmechanismen innerhalb der Bestätigungsformulare des Authorization Servers eingebettet sind, müssen entsprechende CSRF Tokens innerhalb der Bestätigung mitgeschickt werden. Ein wieder einspielen von alten Nachrichten kann daher nicht in Betracht gezogen werden.

Zusammenfassend ergeben sich also Schwierigkeiten für das automatisierte Testen dadurch, dass das OAuth Protokoll unterschiedlich durch die Dienstanbieter ausgelegt wird sowie dass bei Änderungen innerhalb des Authorization Requests die Bestätigung des Resource Owners eingeholt werden muss und dieses schwierig zu automatisieren ist.

5.2. Konzept

Das Konzept, welches nachstehend erläutert wird, entgeht beiden beschriebenen Schwierigkeiten. Dazu wird zum einen ausgenutzt, dass bei Änderungen des Protokollverhaltens nicht der Fehlerfall betrachtet wird, sondern der reine Erfolgsfall und zum anderen wird ein Test Framework eingesetzt, welches eine Browserautomatiserung bietet und so die einzelnen Autorisierungsanfragen automatisch bestätigt werden können.

Der Grundgedanke des Konzepts basiert darauf einmalig manuell festzustellen, wie der Protokollverlauf im Normalfall, d.h. ohne Abänderung von Parameters oder sonstigem, bei einem Dienstanbieter mit einer bestimmten Client Anwendung ablaufen würde. Der Protokollverlauf führt während des Authorization Requests grundsätzlich[1] dazu, dass dem Resource Owner ein Bestätigungsformular, dem sogenannten OAuth Consent (siehe Abbildung 5.5), für die Autorisierungsbestätigung angezeigt wird. Dieses Bestätigungsformular ist generell gleich aufgebaut. Dem Resource Owner wird in einer Liste angezeigt, welche Berechtigungen (Scopes) er an die Client Anwendung vergibt. Weiterhin beinhaltet das Bestätigungsformular einen Bestätigungsbutton sowie einen Button zur Ablehnung.

Werden während des Authorization Requests an den Parametern Veränderungen vorgenommen, welche durch den Authorization Server akzeptiert werden, führt dies weiterhin zur Anzeige des Bestätigungsformulars. Dieses kann am besten durch Überprüfung der Darstellung des Bestätigungsbuttons verifiziert werden. Werden Protokollveränderungen vorgenommen, welche zu einem Fehler im Protokollverlauf führen, wird die entsprechende Fehlerroutine des Dienstanbieters ausgeführt. Dabei ist es für das Konzept nicht von Interesse, wie mit dem Fehler umgegangen wird. Letztendlich ist nur relevant, dass im Fehlerfall kein Bestätigungsbutton für den Authorization Request angezeigt wird.

Da mit dieser Vorgehensweise nur reine Ja/Nein Entscheidungen getestet werden können, ist es wichtig zu beachten, dass immer nur eine Veränderung im Vergleich zum festlegten Normalzustand vorgenommen wird. Nach Abschluss der Test können die gewonnenen Testergebnisse dann zu entsprechenden Angriffsszenarien zusammen gesetzt werden.

[1] unter der Bedingung, dass der Dienstanbieter keinen automatisierten Prozess bei erneuten und schon einmal gewährten Autorisierungsanfragen implementiert hat

Abbildung 5.5.: OAuth Consent

Zur automatischen Überprüfung lassen sich Assertions [57] nutzen. Eine Assertion ist in diesem Zusammenhang ein boolescher Ausdruck, welcher im Vorfeld eine notwendige Bedingung für eine korrekte Ausführung definiert. Die notwendige Bedingung ist in diesem Fall z.B. die Darstellung des Autorisierungsbuttons. Zur genauen Identifizierung dessen kann der Text des Buttons, die ID oder eine bestimmte CSS Klasse verwendet werden.

Ein weiteres Element des Konzeptes ist die Automatisierung des Authorization Requests und der Autorisierungsbestätigungen mittels eines Browserautomatisierungs-Frameworkes. Anstatt maschinell direkte HTTP Requests zu verschicken und HTTP Responses zu analysieren, wird durch das Browserautomatisierungs-Framework das menschliche Verhalten emuliert. Dabei wird für jeden Test ein Browser geöffnet und in diesem der Authorization Request ausgeführt. Im Fall eines gültigen Authorization Requests wird der Autorisierungsbutton, welcher in der manuellen Analyse ermittelt wurde, automatisiert angeklickt. Durch diese Verfahrensweise wird gewährleistet, dass zu jedem Authorization Requests gültige Authorization Codes oder Access Token erzeugt werden. Weiterhin müssen so keine CSRF Schutzmechanismen umgangen werden.

5.3. Prototypische Implementierung

Als Kern der prototypischen Implementierung des Konzepts wird das Selenium Framework [58], welches eine Sammlung von Tools zur Automatisierung von Browsern auf verschiedenen Plattformen bietet,

verwendet. Browserautomatisierung bedeutet in diesem Zusammenhang, dass der Browser aus der Sicht eines Anwenders verwendet wird, jedoch aber Aktionen, wie eine bestimmte Seite aufrufen oder ein Formular absenden, maschinell durchgeführt werden können. Die zugrunde Architektur von Selenium wird in Abbildung 5.6 dargestellt.

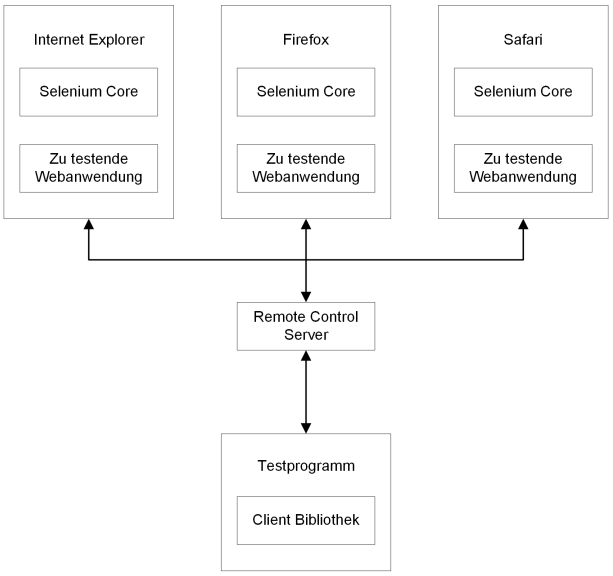

Abbildung 5.6.: Selenium Architektur [59]

Die zentrale Komponente ist der Selenium Remote Control (RC) Server. Dieser hat einerseits die Aufgabe die Browser zu starten bzw. zu schließen und andererseits die Selenium Kommandos, welche vom Testprogramm übermittelt werden, an den Selenium Core weiter zu senden. Damit das Testprogramm mit dem RC Server kommunizieren kann, muss es eine Selenium Client Bibliothek implementiert haben. Der Selenium Core, bestehend aus einer Sammlung von JavaScript Funktionen, wird beim Starten des Browser automatisch als Erweiterung dem Browser hinzugefügt. Er dient dazu die Selenium Befehle zu interpretieren und mit Hilfe des Browser internen JavaScript Interpreters auszuführen. Die Ergebnisse werden anschließend über den RC Server zurück an das Testprogramm gesendet [59].

Zur Vereinfachung des Testprogrammes wurde die Authentifizierung des Resource Owners gegenüber dem Authorization Server ausgelagert. Dazu wird ein vorgeschalteter Burp Proxy [60] mit aktivierten Session Handling verwendet. Dies führt dazu, dass Cookies von besuchten Seiten in einer sogenannten Cookie Jar Datei gespeichert werden und bei erneuten Anfragen an die Webseite dem Request hinzugefügt werden. Authentifiziert sich der Resource Owner nun vor Beginn der Tests in einem beliebigen Browser mit dem konfigurierten Burp Proxy bei dem Dienstanbieter, so wird dementsprechend bei den Authorization Requests des Testprogrammes das entsprechende Cookie mit gesendet.

Als Selenium Client Bibliothek wurde für die prototypische Implementierung des Konzeptes der PHP-Webdriver [61] gewählt. Diese bietet auf Grund der PHP Umgebung den Vorteil, dass die in Kapitel 4.2 verwendete OAuth Client Bibliothek wieder verwendet werden kann. Dadurch ist es möglich Client Funktionalitäten, wie den Austausch des Authorization Codes gegen einen Access Token oder Anfra-

gen für den Resourcenzugriff an die API der Dienstanbieter, aus dem Testprogramm heraus zu nutzen. Die Client Anwendung wird dementsprechend lokal in der Testumgebung des Testers ausgeführt. Um Aspekte bezüglich der Redirection URI unter realen Bedingungen[2] zu testen, wird eine externe Webseite benötigt. Dieses hat jedoch zur Folge, dass die Redirection URI nicht mehr zurück zu dem Client führt und so der Protokollablauf unterbrochen wird. Daher muss sicher gestellt sein, dass die lokal laufende Client Anwendung auf die Informationen (Authorization Code bzw. Access Token), welche an die Redirection URI gesendet werden, zugreifen kann. Dazu konnte als einzige Lösung für dieses Probleme evaluiert werden, dass diese Informationen auf der Webseite der Redirection URI angezeigt und mittels des Selenium Webdrivers abgegriffen werden.

Zur Durchführung von einzelnen Tests, werden die Selenium Client Bibliothek und die OAuth Client Bibliothek von PHPUnit [62], einem Testframework für PHP, umgeben. Die gesamte Architektur der prototypischen Implementierung ist in Abbildung 5.7 dargestellt.

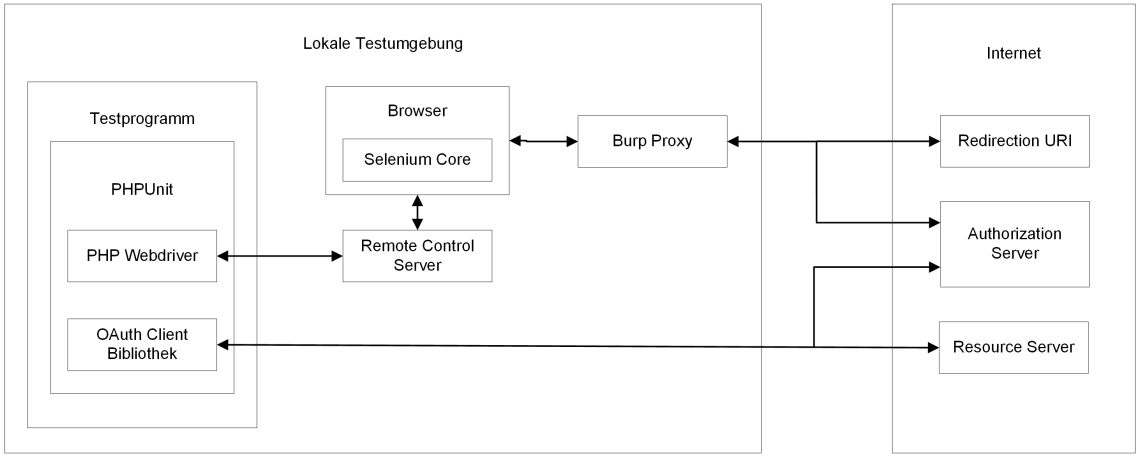

Abbildung 5.7.: Architektur der prototyischen Implementierung

Zur Verdeutlichung des Zusammenwirkens der einzelnen Komponenten wird im folgenden der Ablauf eines beispielhaften Testcases erläutert. Der im Listing 5.1 gezeigte Test überprüft, ob der Token Type im Access Token Response den Wert „Bearer“ besitzt.

Zeile 2: Die OAuth Client Bibliothek wird verwendet, um die Authorization Request URL mit den einzelnen Parametern zusammenzusetzen. Es werden als Query Parameter der Response Type sowie die Redirection URI übergeben.

Zeile 3: Mittels des PHP Webdrivers wird der Authorization Request im Browser ausgeführt.

Zeile 4: Innerhalb des Browsers wird der Bestätigungsdialog für den Authorization Request angezeigt. Da im Vorfeld über die Variable *control_element* die ID des Bestätigungsbuttons festgelegt wurde, kann dieser automatisiert angeklickt werden.

[2] Für Testzwecke wäre es bei manchen Dienstanbietern auch mögliche eine lokale (z.B. 127.0.0.1) Adresse für die Redirection URI registrieren

```
1  public function testBearerToken(){
2    $this->url = $this->client->getAuthenticationUrl($this->auth_endpoint,
         array('response_type' => 'code','redirect_uri' => $this->
         redirect_endpoint));
3    $this->webDriver->get($this->url);
4    $this->webDriver->findElement(WebDriverBy::id($this->contol_element))->
         click();
5    $parameters=array();
6    $parameters['grant_type']='authorization_code';
7    $parameters['redirect_uri']=$this->redirect_endpoint;
8    $parameters['code']=$this->webDriver->findElement(WebDriverBy::id('code')
         )->getText();
9    try {
10     $response=$this->client->getAccessToken($this->token_endpoint, $
           parameters, 'POST');
11   }catch (Exception $e){ }
12   $this->assertEquals('Bearer', $response['result']['token_type']);
13 }
```

Listing 5.1: Beispielhafter Testcase

Zeile 5-8: Die Parameter für den Access Token Request werden festgelegt.

Ziele 8: Da die Webseite der Redirection URI den Authorization Code innerhalb eines Div Tags mit der ID *code* anzeigt, kann der Authorization Code mit Hilfe des Webdrivers extrahiert werden.

Zeile 10: Mittels der OAuth Client Bibliothek wird ein Access Token beim Token Endpoint des Authorization Servers angefragt. Dazu werden die Parameter *grant_type*, *redirection_uri* und *code* übersendet. Der Response des cURL Requests wird in der Variable *response* gespeichert.

Zeile 12: Mit Hilfe einer Assertion kann überprüft werden, ob der Token Type Parameter des Access Token Responses den Wert „Bearer“ besitzt.

6. Reflexion der Ergebnisse der Arbeit

6.1. Verwandte Arbeiten

Eine Vielzahl von Arbeiten haben sich im Vorfeld schon mit der Sicherheit von Sigle Sign-On Lösungen oder Authentifizierungsprotokollen (wie z.B. SAML oder OpenID) beschäftigt [63] [64] [65] [66]. Gleichermaßen existieren auch Arbeiten, welche sich mit der Sicherheit [67] und der formalen Verifikation [68, 69] des OAuth 2.0 Protokolls auseinander gesetzt haben. Daher werden nachfolgend verwandte Arbeiten zum Themengebiet „Sicherheitsanalyse von OAuth 2.0“ vorgestellt.

Innerhalb ihrer Arbeit stellen Bansal et al. [70] einen Ansatz vor, mittels dem sie durch formale Analyse des OAuth 2.0 Protokolls konkrete Angriffe entdeckt haben. Dazu haben sie im Vorfeld verschiedene Konfigurationen des Protokolls modelliert und unter der Verwendung des ProVerif Tools [71] formal analysiert. Als Grundlage für ihre Angriffe haben sie einen Netzwerk Angreifer gewählt, welcher die Kommunikation über nicht private Kommunikationskanäle abfangen und verändern kann. Durch ihre Analyse haben sie mehrere sogenannte Social CSRF Angriffe innerhalb von Client Anwendungen und Authorization Servern entdeckt.

Yang und Manoharan [72] analysieren die Sicherheit des OAuth 2.0 Protokolls mittels eines unlimitierten Netzwerkangreifers. Dabei konzentriert sich ihr Angreifer auf das Abfangen von Nachrichten zwischen dem User Agent des Resource Owners und der Client Anwendung, umso den übermittelten Authorization Code zu erhalten. Anschließend können sie entweder den gesamten Response wieder einspielen oder einen eigenen Protokollfluss initialisieren, um den Authorization Code gegen einen Access Token einzutauschen und so auf die geschützten Ressourcen des Opfers zu greifen zu können.

Wang et al. [73] konzentrieren sich bei ihrer Analyse auf die von den Dienstanbietern bereitgestellten SDKs, welche eine sichere Authentifizierung und Autorisierung bieten sollen. Dabei gehen sie der Frage nach, ob Anwendungen sicher sind, wenn Entwickler die SDKs in einem angemessenen Rahmen verwenden. Alle untersuchten Anbieter verwenden das OAuth Protokoll. Die Untersuchung des Protokolls ist jedoch nicht Bestandteil ihrer Arbeit. Bei der Analyse haben die Autoren herausgefunden, dass die SDKs keine direkten Fehler enthalten, jedoch aber nur unter bestimmten nicht dokumentierten Annahmen zu einer sichereren Umgebung führen. Diese Annahmen wurde mittels manuellem Aufwand und automatisierter formeller Überprüfung identifiziert.

Sun und Beznosov [38] analysieren die Sicherheit des OAuth Protokolls anhand von reellen Implementationen. Dazu überprüfen sie, wie bekannte Webschwachstellen (XSS, CSRF, Browser Schwachstellen u.a.) genutzt werden können um das OAuth Protokoll zu kompromittieren. In diesem Zusammenhang untersuchen sie die Gründe, die zur Ausnutzung geführt haben und deren Konsequenzen. Weiterhin ge-

ben sie Empfehlungen zur Behebung der Schwachstellen an. Verwunderlich an der Arbeit ist jedoch, dass sie OAuth als ein Single Sign-On System betrachten, obwohl das OAuth Protokoll nur zur Autorisierung einer Dritt-Parteien Anwendung konzipiert wurde [9].

6.2. Fazit

In der vorliegenden Arbeit wurde das Themengebiet „OAuth-basierte Autorisierung und Authentifizierung“ übersichtlich aufgearbeitet. Dafür wurde zunächst aufgezeigt, welche RFCs und aktive Internet Drafts dem OAuth 2.0 Framework zugeordnet werden. Anschließend wurde das zugrunde liegende Konzept erläutert, sowie die unterschiedlichen Grant Types mit ihren zugehörigen Nachrichten erklärt. Die Vergleichbarkeit der einzelnen Grant Types wurde durch eine einheitliche Struktur erreicht. Weiterhin wurde dargestellt, welche Sicherheitsvorkehrungen das OAuth 2.0 Protokoll besitzt, sowie welche Sicherheitsempfehlungen für die Implementierung ausgesprochen werden.

Im zweiten Teil der Arbeit wurde eine manuelle Sicherheitsanalyse des OAuth 2.0 Protokolls durchgeführt. Dafür wurde ein ausführlicher Untersuchungskatalog erarbeitet, welcher Ansatzpunkte für mögliche Schwachstellen des OAuth Protokolls und dessen Implementierung überprüft. Bei der manuellen Sicherheitsanalyse konnte festgestellt werden, dass obwohl innerhalb des OAuth 2.0 Standards und dem beliegendem OAuth Threat Model Sicherheitsempfehlungen gegeben werden, diese ungenügend von den Dienstanbietern umgesetzt werden. So wird in vielen Fällen die Redirection URI unzureichend validiert, so dass ein Angreifer diese modifizieren und dadurch den Protokollfluss zu seinem Vorteil verändern kann. Weiterhin wurde festgestellt, dass auf Grund der Tatsache das der Authorization Code nicht gegen Änderungen abgesichert ist, unter bestimmten Bedingungen ein Angreifer einen abgefangenen Authorization Code als sein eigener ausgeben kann. Dadurch ist es einem Angreifer auch im Fall des Authorization Code Grant Type mittels des Clients möglich auf die Ressourcen eines Opfers zu zugreifen.

Abschließend wurde ein Konzept sowie eine prototypische Implementierung für das automatisierte Testen des OAuth 2.0 Protokolls vorgestellt. Dazu wurde gezeigt, wie die Schwierigkeit der manuellen Zustimmung durch den Resource Owner automatisiert durchgeführt werden kann.

6.3. Ausblick

Da der Schwerpunkt der manuellen Sicherheitsanalyse auf Grund der hohen Verbreitung bei dem Authorization Grant Type sowie dem Implicit Grant Type lag, lassen sich in zukünftigen Arbeiten die weiteren Grant Types auf mögliche Schwachstelle untersuchen. So kann z.B. überprüft werden ob bekannte Angriffe auf SAML, wie z.B. Signature Wrapping Attack [66], sich auch auf den SAML 2 Bearer Assertion Grant Type übertragen lassen.

Des weiteren wurde für das Konzept zur Durchführung von automatischen Penetrationstests lediglich ein Prototyp entwickelt. Dieser lässt sich grundlegend hinsichtlich seiner Funktionalität und Umfang erweitern. Derzeit müssen alle Parameter des Clients von Hand innerhalb des Testprogrammes gesetzt werden. Dieses könnte über das Einlesen einer Konfigurationsdatei vereinfacht werden. Weiterhin sind

nicht alle Untersuchungsaspekten der manuellen Sicherheitsanalyse in automatisierte Test umgewandelt worden, so dass hier weiterer Handlungsbedarf besteht. Auch wäre es möglich das Konzept bzw. das Testprogramm als ein Plugin in ein bestehendes Pentesting Framework oder Programm zu integrieren. Unter Umständen müsste dafür jedoch die zugrunde liegende Programmarchitektur (insbesondere die Programmiersprache) gewechselt werden.

Literaturverzeichnis

[1] R. Boyd, *Getting started with OAuth 2.0*, 1st ed. Sebastopol and CA: O'Reilly, 2012.

[2] E. Hammer-Lahav. (2010) Rfc 5849: The oauth 1.0 protocol. [Online]. Available: http://tools.ietf.org/html/rfc5849

[3] G. Brail and S. Ramji. (2012) Oauth - the big picture. [Online]. Available: http://pages.apigee.com/rs/apigee/images/oauth-ebook-2012-02.pdf

[4] E. Hammer. (2010) Oauth 2.0 (without signatures) is bad for the web. [Online]. Available: http://hueniverse.com/2010/09/oauth-2-0-without-signatures-is-bad-for-the-web/

[5] ——. (2012) Oauth 2.0 and the road to hell. [Online]. Available: http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/

[6] A. Labunets. (2013) A story of $9500 bug in facebook oauth 2.0. [Online]. Available: http://isciurus.blogspot.de/2013/04/a-story-of-9500-bug-in-facebook-oauth-20.html

[7] E. Homakov. (2013) How we hacked facebook with oauth2 and chrome bugs. [Online]. Available: http://homakov.blogspot.co.uk/2013/02/hacking-facebook-with-oauth2-and-chrome.html

[8] N. Goldshlager. (2013) How i hacked facebook oauth to get full permission on any facebook account (without app "allow" interaction). [Online]. Available: http://www.breaksec.com/?p=5734

[9] D. Hardt. (2012) Rfc 6749: The oauth 2.0 authorization framework. [Online]. Available: http://tools.ietf.org/html/rfc6749

[10] S. Haiges, *OAuth 2.0: Client & Server*, 1st ed., ser. shortcuts. Entwicker.press, 2013.

[11] T. Lodderstedt, M. McGloin, and P. Hunt. (2013) Rfc 6819: Oauth 2.0 threat model and security considerations. [Online]. Available: http://tools.ietf.org/html/rfc6819

[12] J. Behrendt and K. Zeppenfeld, *Web 2.0*, 1st ed. Berlin and Heidelberg: Springer-Verlag, 2008.

[13] T. O'Reilly. (2005) What is web 2.0? design patterns and business models for the next generation of software. [Online]. Available: http://www.oreilly.de/artikel/web20.html

[14] A. Parecki. (2012) An introduction to oauth 2. [Online]. Available: http://oreillynet.com/pub/e/2184

[15] Google. (2012) Authsub for web applications. [Online]. Available: https://developers.google.com/accounts/docs/AuthSub?hl=de

[16] Yahoo. (2013) Browser-based authentication. [Online]. Available: http://developer.yahoo.com/bbauth/

[17] E. Hammer. (2011) The oauth 1.0 guide - history. [Online]. Available: http://hueniverse.com/oauth/guide/history/

[18] OAuth Core Workgroup. (2007) Oauth core 1.0. [Online]. Available: http://oauth.net/core/1.0

[19] E. Hammer-Lahav. (2009) Oauth security advisory: 2009.1. [Online]. Available: http://oauth.net/advisories/2009-1/

[20] oauth.net. (2009) Oauth security advisory: 2009.1. [Online]. Available: http://oauth.net/advisories/2009-1/

[21] E. Hammer-Lahav. (2010) Introducing oauth 2.0. [Online]. Available: http://hueniverse.com/2010/05/introducing-oauth-2-0/

[22] R. Shirey. (2007) Rfc4949: Internet security glossary: Version 2. [Online]. Available: http://tools.ietf.org/html/rfc4949

[23] M. Benantar, *Access Control Systems*. New York: Springer Science+Business Media, Inc., 2006.

[24] C. Mezler-Andelberg, *Identity Management - eine Einführung: Grundlagen, Technik, wirtschaftlicher Nutzen*, 1st ed. Heidelberg: dpunkt-Verl., 2008.

[25] D. Baier. (2012) Oauth2 - ready or not? [Online]. Available: https://www.troopers.de/wp-content/uploads/2012/12/TROOPERS13-OAuth2-Ready_or_not_here_I_come-Dominick_Baier.pdf

[26] E. Hammer. (2009) The oauth 1.0 guide - terminology. [Online]. Available: http://hueniverse.com/oauth/guide/terminology/

[27] M. Jones and D. Hardt. (2012) Rfc 6750: The oauth 2.0 authorization framework: Bearer token usage. [Online]. Available: http://tools.ietf.org/html/rfc6750

[28] IETF. (2013) Web authorization protocol (oauth). [Online]. Available: http://datatracker.ietf.org/wg/oauth/

[29] B. Campbell and H. Tschofenig. (2012) Rfc 6755 - an ietf urn sub-namespace for oauth. [Online]. Available: http://tools.ietf.org/html/rfc6755

[30] M. Mealling, L. Masinter, T. Hardie, and G. Klyne. (2003) Rfc 3553 - an ietf urn sub-namespace for registered protocol parameters. [Online]. Available: http://tools.ietf.org/html/rfc3553

[31] T. Lodderstedt, S. Dronia, and M. Scurtescu. (2013) Rfc 7009 - oauth 2.0 token revocation. [Online]. Available: http://tools.ietf.org/html/rfc7009

[32] B. Campbell, C. Mortimore, M. Jones, and Y. Goland. (2013) Assertion framework for oauth 2.0 client authentication and authorization grants: draft-ietf-oauth-assertions-12. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-assertions-12

[33] J. Richer, J. Bradley, M. Jones, and M. Machulak. (2013) Oauth 2.0 dynamic client registration protocol: draft-ietf-oauth-dyn-reg-14. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-dyn-reg-14

[34] M. Jones, J. Bradley, and N. Sakimura. (2013) Json web token (jwt): draft-ietf-oauth-json-web-token-12. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-12

[35] M. Jones, B. Campbell, and C. Mortimore. (2013) Json web token (jwt) profile for oauth 2.0 client authentication and authorization grants: draft-ietf-oauth-jwt-bearer-06. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer-06

[36] B. Campbell, C. Mortimore, and M. Jones. (2013) Saml 2.0 profile for oauth 2.0 client authentication and authorization grants: draft-ietf-oauth-saml2-bearer-17. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-17

[37] J. Richer, M. Mills, H. Tschofenig, and P. Hunt. (2013) Oauth 2.0 message authentication code (mac) tokens: draft-ietf-oauth-v2-http-mac-04. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-04

[38] S.-T. Sun and K. Beznosov, "The devil is in the (implementation) details: an empirical analysis of oauth sso systems," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS '12. ACM, 2012, pp. 378–390.

[39] T. Berners-Lee. (1997) Uri references: Fragment identifiers on uris. [Online]. Available: http://www.w3.org/DesignIssues/Fragment.html

[40] T. Duong and J. Rizzo. (2011) Here come the ⊕ ninjas. [Online]. Available: http://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf

[41] E. Hammer. (2010) Oauth bearer tokens are a terrible idea. [Online]. Available: http://hueniverse.com/2010/09/oauth-bearer-tokens-are-a-terrible-idea/

[42] S.-T. Sun. (2012) Simple but not secure: An empirical security analysis of oauth 2.0-based single sign-on systems.

[43] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song, "Towards a formal foundation of web security," in *2010 IEEE 23rd Computer Security Foundations Symposium (CSF)*, 2010, pp. 290–304.

[44] K. Fu, E. Sit, K. Smith, and N. Feamster, "The dos and don'ts of client authentication on the web," in *10th USENIX Security Symposium, 2001, Washington, D.C., USA*, Dan S. Wallach, Ed. USENIX, 2001.

[45] Google. (2013) Oauth 2.0 playground. [Online]. Available: https://developers.google.com/oauthplayground/

[46] oauth.net. (2013) Oauth 2.0. [Online]. Available: http://oauth.net/2/

[47] Google. (2013) Using oauth 2.0 to access google apis. [Online]. Available: https://developers.google.com/accounts/docs/OAuth2

[48] Facebook. (2013) Getting started with the facebook sdk for php. [Online]. Available: https://developers.facebook.com/docs/php/gettingstarted/

[49] C. Pierrick and B. Anis. (2013) Light php wrapper for the oauth 2.0. [Online]. Available: https://github.com/adoy/PHP-OAuth2

[50] php.net. (2013) Client url library. [Online]. Available: http://www.php.net/manual/en/book.curl.php

[51] Wikipedia.org. (2013) Oauth. [Online]. Available: http://en.wikipedia.org/wiki/OAuth

[52] G. Namachivayam. (2013) oauth light-weight class demo. [Online]. Available: http://ngiriraj.com/socialMedia/oauthlogin/index.php

[53] OAuth.io. (2013) Providers. [Online]. Available: https://oauth.io/providers

[54] ——. (2013) Wishlist. [Online]. Available: https://oauth.io/wishlist

[55] German Testing Board. (2013) Istqb - /gtb standardglossar der testbegriffe: Deutsch/englisch. [Online]. Available: http://www.german-testing-board.info/fileadmin/gtb_repository/downloads/pdf/glossar/CT_Glossar_DE_EN_V22.pdf

[56] E. Dustin, J. Rashka, and J. Paul, *Software automatisch testen: Verfahren, Handhabung und Leistung*, ser. Xpert.press. Springer, 2001.

[57] R. Binder, *Testing object-oriented systems: models, patterns, and tools*. Boston and Munich: Addison-Wesley, 2000.

[58] Selenium. (2013) Seleniumhq: Browser automation. [Online]. Available: http://www.seleniumhq.org/

[59] ——. (2013) Selenium 1 (selenium rc). [Online]. Available: http://www.seleniumhq.org/docs/05_selenium_rc.jsp

[60] PortSwigger. (2013) Burp suite. [Online]. Available: http://portswigger.net/burp/

[61] Facebook. (2013) php-webdriver – webdriver bindings for php. [Online]. Available: https://github.com/facebook/php-webdriver

[62] S. Bergmann. (2013) Phpunit. [Online]. Available: http://phpunit.de/manual/current/en/index.html

[63] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services," in *2012 IEEE Symposium on Security and Privacy (SP)*, IEEE Computer Society, Ed., pp. 365–379.

[64] S.-T. Sun, K. Hawkey, and K. Beznosov, "Systematically breaking and fixing openid security: Formal analysis, semi-automated empirical evaluation, and practical countermeasures," *Computers & Security*, vol. 31, no. 4, pp. 465–483, 2012.

[65] R. Wang, S. Chen, X. Wang, and S. Qadeer, "How to shop for free online: Security analysis of cashier-as-a-service based web stores," in *IEEE Symposium on Security and Privacy (SP)*, 2011, pp. 465–480.

[66] J. Somorovsky, A. Mayer, J. Schwenk, M. Kampmann, and M. Jensen, "On breaking saml: be whoever you want to be," in *Proceedings of the 21st USENIX conference on Security symposium*, 2012, p. 21.

[67] S. Chari, C. S. Jutla, and A. Roy, "Universally composable security analysis of oauth v2. 0," *IACR Cryptology ePrint Archive*, vol. 2011, p. 526, 2011.

[68] S. Pai, Y. Sharma, S. Kumar, R. M. Pai, and S. Singh, "Formal verification of oauth 2.0 using alloy framework," in *International Conference on Communication Systems and Network Technologies (CSNT)*, 2011, pp. 655–659.

[69] M. Miculan and C. Urban, "Formal analysis of facebook connect single sign-on authentication protocol," in *SOFSEM*, vol. 11, 2011, pp. 22–28.

[70] C. Bansal, K. Bhargavan, and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis," in *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE, 2012, pp. 247–262.

[71] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *14th IEEE Computer Security Foundations Workshop*, 2001, pp. 82–96.

[72] F. Yang and S. Manoharan, "A security analysis of the oauth protocol," in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2013, pp. 271–276.

[73] R. Wang, Y. Zhou, S. Chen, S. Qadeer, D. Evans, and Y. Gurevich, "Explicating sdks: Uncovering assumptions underlying secure authentication and authorization," *22nd USENIX Security Symposium*, 2013.

A. Ergebnisse der manuellen Sicherheitsanalyse

Untersuchungsmerkmal	Box	Cheddar	Dailymile	Dailymotion	Facebook
URL	https://www.box.com/	https://cheddarapp.com/	http://www.dailymile.com	http://www.dailymotion.com/	https://www.facebook.com/
Beschreibung	Cloud Speicher	Task List	Workout Tracking	Video Plattform	Soziales Netzwerk
Devoloper URL	http://developers.box.com/oauth/	https://cheddarapp.com/developer/authentication	http://www.dailymile.com/api/documentation/oauth	http://www.dailymotion.com/doc/api/authentication.html	https://developers.facebook.com/docs/facebook-login/
Grant Types	Authorization Code	Authorization Code	Authorization Code, Implicit	Authorization Code, Implicit, Password Crential	Authorization Code, Implicit, Implit Authorization Code (Eigenentwicklung), Client Credential (Apps)
Authorization Request					
Authorization Endpoint					
URL	https://www.box.com/api/oauth2/authorize	https://api.cheddarapp.com/oauth/authorize	https://api.dailymile.com/oauth/authorize	https://api.dailymotion.com/oauth/authorize	https://www.facebook.com/dialog/oauth
Erreichbarkeit über HTTP	nein	Weiterleitung auf HTTPS	nein	nein	Weiterleitung auf HTTPS
HTTP Methoden	GET, POST	GET, POST, TRACE, PUT, DELETE, PATCH	GET	egal (außer TRACE)	GET, HEAD
Response Type					
Erforderlichkeit	ja	nein	ja	ja	nein (Code default)
Änderbarkeit	nein	n/a	ja	ja	ja
Client ID					
Komplexität	jcwrtha9cz0leibongy8ppx4x85orqfx	8bb7e671749a09401985ad2cd526cb68	Y2NJjUJNq0nYsIbn96PNLA3bKUGFyqtmeFBnWcm	b8abb96a6ab840f595bb	560096010695234
Redirectiom URI					
Vorherige Registrierung	optional	ja	optional	ja	ja
Registrierung einer HTTP URL	nein	ja	ja	ja	ja
Weiterleitung an HTTP URI bei Nichtregistrierung	nein	n/a	ja	n/a	n/a
Domain des Dienstanbieters	nein	n/a	nein	nein	Verwendung der erzeugten Canvas Seite https://apps.facebook.com/<app-name> möglich
Erforderlichkeit	ja	nein	ja	ja	ja

Untersuchungsmerkmal	Box	Cheddar	Dailymile	Dailymotion	Facebook
Striktheit der Überprüfung	Host-name, Subdomain erlaubt, https://c01n.de.to funktioniert	redirect URI als prefix, https://c01n.de.to funktioniert	wenn registriert, dann strikt	redirect URI als prefix, https://c01n.de.to funktioniert	Host-name, Subdomain erlaubt
Scope					
Anzahl	0	0	0	10	100+
Unterteilte Zugriffsberechtigungen	nein	nein	nein	nein	ja
State					
Erforderlichkeit	optional	optional	optional	optional	optional
Implementierung					
Abweichende Parameter vom Standard	nein	nein	nein	ja [display]	ja [display]
Anforderung Refresh Token	automatisch	n/a	n/a	automatisch	n/a
Authentifizierung des Nutzers	Jedesmal	Session	Session	Jedesmal	Session
Autorisierung bei jedem Authorization Request	Ja	Ja	Ja	Ja, durch Authentifizierung	Nein
Authorization Response					
Authentication Code					
Änderbarkeit	ja	ja	ja	ja	ja
Lebensdauer	30s	n/a	n/a	n/a	n/a
Error Response					
Error					
Definition	invalid_request, unauthorized_client, invalid_grant, invalid_client, redirect_uri_mismatch, insecure_redirect_uri, invalid_redirect_uri	n/a	n/a	n/a	n/a
Access Token Request					
Token Endpoint					
URL	`https://www.box.com/api/oauth2/token`	`https://api.cheddarapp.com/oauth/token`	`https://api.dailymile.com/oauth/token`	`https://api.dailymotion.com/oauth/token`	`https://graph.facebook.com/oauth/access_token`
Erreichbarkeit über HTTP	nein	nein	nein	nein	nein
Code					
Mehrmalige Verwendung	ja	nein	nein	nein	nein
Gültigkeit alter Tokens	ja	n/a	n/a	n/a	nein
Redirection URI					

Untersuchungsmerkmal	Box	Cheddar	Dailymile	Dailymotion	Facebook
Änderbarkeit	Top-Level-Domain	ja	nein	nein	nein, strikte Überprüfung gegenüber Angabe im Authorization Request
Authentifikation					
Methode	Query Parameter	Basic Auth	Query Parameter	Query Parameter	Query Parameter
Access Token Response					
Access Token					
Typ	Bearer	Bearer	Bearer	Bearer	Bearer
Übermittlungsformat	json	json	json	json	json
Felder	access_token, expires_in, restricted_to, token_type, refresh_token	access_token, token_type, user[created_at, first_name, has_plus, id, last_name, updated_at, username, url, socket[auth_url, api_key, app_id, channel]]	access_token, token_type	access_token, expires_in, refresh_token	access_token, expires
Gültigkeitsdauer	1h	n/a	n/a	10h	60d
Ausstellung eines neues Access Token	ja	nein	nein	ja	ja
Gültigkeit eines alten Access Token	ja	ja	ja	ja	ja
Refresh Token					
Gültigkeitsdauer	14d	n/a	n/a	n/a	n/a
Erneuerung des Refresh Tokens	ja	n/a	n/a	nein	n/a
Ungültigkeit alter Tokens	ja	n/a	n/a	nein	n/a
API Call					
Scope					
Zugriff außerhalb des autorisierten Scopes	n/a	n/a	n/a	n/a	nein
Token					
Übermittlungsmethode	Header: Bearer	Header: Bearer, Query Parameter, Form Parameter	Query Parameter (oauth_token)	Query Parameter, Header: Oauth	Query Parameter
Revoke Token					
URL	`https://www.box.com/api/oauth2/revoke`	n/a	n/a	`https://api.dailymotion.com/logout`	n/a
Parameter	client_id, client_secret, token	n/a	n/a	access_token	n/a

Untersuchungsmerkmal	Box	Cheddar	Dailymile	Dailymotion	Facebook
HTTP Methode	POST	n/a	n/a	GET	n/a
Zusammenhang zwischen den Tokens	Übersendung eines Tokens reicht	n/a	n/a	access_token widerruft auch refresh token	n/a

Untersuchungsmerkmal	**Foursquare**	**Google**	**Salesforce**	**Stackexchange**
URL	`https://foursquare.com/`	`https://www.google.com`	`http://www.force.com/`	`http://stackexchange.com/`
Beschreibung	Soziales Netzwerk	Content Anbieter	Cloud Plattform	FAQ Plattform
Devoloper URL	`https://developer.foursquare.com/overview/auth`	`https://developers.google.com/accounts/docs/OAuth2`	`http://help.salesforce.com/apex/HTViewHelpDoc?id=remoteaccess_authenticate.htm`	`http://api.stackexchange.com/docs/authentication`
Grant Types	Authorization Code, Implicit	Authorization Code, Implicit, JWT Bearer Token (Server to Server Applications), Device Flow	Authorization Code, Implicit, JWT Bearer Token, SAML Bearer Assertion, Password	Authorization Code, Implicit (es geht immer nur ein Verfahren)
Authorization Request				
Authorization Endpoint				
URL	`https://de.foursquare.com/oauth2/authorize`	`https://accounts.google.com/o/oauth2/auth`	`https://login.salesforce.com/services/oauth2/authorize`	`https://stackexchange.com/oauth`
Erreichbarkeit über HTTP	nein	Weiterleitung auf HTTPS	nein	nein
HTTP Methoden	egal (außer TRACE)	GET, POST	egal	GET, POST, HEAD
Response Type				
Erforderlichkeit	ja	ja	ja	nein
Änderbarkeit	ja	ja	ja	nein
Client ID				
Komplexität	L5AKNTRRJZBIYZ4KORODINWH35MDLJZNU2R0YSTJL1GSG20N	19414004318-f1aiv12k91pf9hsq9hroq9jnh3kbheft.apps.googleusercontent.com	3MVG99qusVZJwhslgP2LEzM2Ppi1Sys54sd0OmYxdBGUiUGwPSnDw1hiyWSR7k35X.X_6vDd487TtpoBMTcQg	2092
Redirectiom URI				
Vorherige Registrierung	ja	ja	ja	ja
Registrierung einer HTTP URL	ja	ja	nein	ja

Untersuchungsmerkmal	**Foursquare**	**Google**	**Salesforce**	**Stackexchange**
Weiterleitung an HTTP URI bei Nichtregistrierung	n/a	n/a	n/a	n/a
Domain des Dienstanbieters	nein	nein	nein	nein
Erforderlichkeit	nein	ja	ja	ja
Striktheit der Überprüfung	Redirect URI als prefix, https://c01n.de.to funktioniert	strikt	strikt	Host-name, Subdomain erlaubt
Scope				
Anzahl	0	100+	7	4
Unterteilte Zugriffsberechtigungen	nein	ja	nein	nein
State				
Erforderlichkeit	optional	optional	optional	optional
Implementierung				
Abweichende Parameter vom Standard	ja [display]	ja [login_hint, approval_prompt, access_type]	ja [immediate, display, prompt]	nein
Anforderung Refresh Token	n/a	Über Parameter access_type=offline	Über Scope „refresh_token“	n/a
Authentifizierung des Nutzers	Session	Session	Session	Session
Autorisierung bei jedem Authorization Request	Ja	Default nein (approval_prompt=auto), kann über Parameter approval_prompt=force erzwungen werden	Nein	Ja (Ausnahme, wenn einmal der Scope private_info gewährt wurde und nur dieser wieder angefragt wird)
Authorization Response				
Authentication Code				
Änderbarkeit	ja	ja	ja	ja
Gültigkeitsdauer	n/a	n/a	n/a	n/a
Error Response				
Error				
Definition	n/a	n/a	unsupported_response_type, invalid_client_id, invalid_request, invalid_request, access_denied, redirect_uri_missing, redirect_uri_mismatch, immediate_unsuccessful, invalid_scope	invalid_request, unauthorized_client, access_denied, unsupported_response_type, invalid_scope, server_error, temporarily_unavailable

Untersuchungsmerkmal	Foursquare	Google	Salesforce	Stackexchange
Access Token Request				
Token Endpoint				
URL	`https://de.foursquare.com/oauth2/access_token`	`https://accounts.google.com/o/oauth2/token`	`https://login.salesforce.com/services/oauth2/token`	`https://stackexchange.com/oauth/access_token`
Erreichbarkeit über HTTP	nein	nein	nein	nein
Code				
Mehrmalige Verwendung	ja	nein	nein	ja
Gültigkeit alter Tokens	nein, da immer der gleiche Access Token ausgestellt wird	n/a	n/a	ja
Redirection URI				
Änderbarkeit	ja, solange redirect_uri prefix bleibt	nein	nein	nein, strikte Überprüfung gegenüber Angabe im Authorization Request
Authentifikation				
Methode	Query Parameter	Query Parameter	Query Parameter	Query Parameter
Access Token Response				
Access Token				
Typ	Bearer	Bearer	Bearer	Bearer
Übermittlungsformat	json	json	json (default), urlencoded, xml ; kann über den Parameter „format“ beim Access Token Request bestimmt werden	json
Felder	access_token	access_token, token_type, expires_in, id_token	access_token, refresh_token, instance_url, id, signature, issued_at	access_token, expires
Gültigkeitsdauer	unendlich	1h	n/a	1d
Ausstellung eines neues Access Token	nein	ja	ja	ja
Gültigkeit eines alten Access Token	ja	ja	ja	ja
Refresh Token				
Gültigkeitsdauer	n/a	n/a	n/a	n/a
Erneuerung des Refresh Tokens	n/a	nein	nein	n/a
Ungültigkeit alter Tokens	n/a	nein	nein	n/a

Untersuchungsmerkmal	**Foursquare**	**Google**	**Salesforce**	**Stackexchange**
API Call				
Scope				
Zugriff außerhalb des autorisierten Scopes	n/a	nein	nein	nein
Token				
Übermittlungsmethode	Query Parameter, Form Parameter	Query Parameter, Header: Bearer	Header: Bearer	Query Parameter
Revoke Token				
URL	n/a	`https://accounts.google.com/o/oauth2/revoke`	`https://login.salesforce.com/services/oauth2/revoke`	n/a
Parameter	n/a	token	token	n/a
HTTP Methode	n/a	GET	GET, POST	n/a
Zusammenhang zwischen den Tokens	n/a	Access Token widerruft auch Refresh Token	Refresh Token widerruft Access Token	n/a