Ostfalia - Hochschule für angewandte Wissenschaften
Fachbereich Wirtschaft
Studiengang Wirtschaftsinformatik

Authentifizierung und Autorisierung im Web mit OAuth 2.0

Bachelorarbeit

Zur Erlangung des Grades eines Bachelor of Science
der Fakultät Wirtschaft
der Ostfalia - Hochschule für angewandte Wissenschaften

eingereicht bei
Dipl.-Inf. Nikolai Alex
Prof. Dr.-Ing. Frank Höppner

von
Andreas Wache
Neulandstraße 19
38154 Königslutter am Elm OT Ochsendorf
Mat.-Nr. 30991667

Königslutter am Elm, den 12. September 2014

Zusammenfassung

Die vorliegende Arbeit befasst sich mit dem Protokoll OAuth 2. OAuth 2 ist ein offener Standard für die Autorisierung von API-Zugriffen im Web. Es ermöglicht verschiedenen Anwendungen den standardisierten und sicher geleiteten Zugriff auf die Ressourcen eines Dienstes im Namen des Eigentümers. Der Eigentümer kann mit Hilfe des Protokolls die Erlaubnis erteilen, ohne sein Passwort gegenüber den Anwendungen bekanntgeben zu müssen. Stattdessen authentifiziert er sich bei dem Dienst, der die Ressourcen anbietet. Die Anwendungen erhalten stellvertretend ein Zugriffstoken, das sie für den eigentlichen Zugriff auf die Ressourcen benötigen. Die Spezifikation des Protokolls definiert dazu verschiedene Abläufe. Im Rahmen der Arbeit werden die unterschiedlichen Abläufe des Protokolls untersucht und Verwendungsmöglichkeiten beschrieben. Es wird außerdem ein Beispiel für den Zugriff auf eine mit OAuth 2 abgesicherte API in der Programmiersprache Java erstellt.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CSRF	Cross-Site-Request-Forgery
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
IETF	Internet Engineering Task Force
Java EE	Java Plattform Enterprise Edition
JAX-RS	Java API for RESTful Web Services
JSON	JavaScript Object Notation
MAC	Message Authentication Code
OAuth	Open Authorization
REST	Representational State Transfer
RFC	Request for Comments
SDK	Software Developement Kit
SSL	Secure Sockets Layer
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

Plattformen wie Facebook, Twitter oder Google bestehen aus Diensten, die sie im Internet ihren Benutzern zur Verfügung stellen.[1] Die Nutzung derartiger Internetdienste ist in den vergangenen Jahren angestiegen und entwickelt sich ständig weiter. Die Interaktion mit anderen Diensten wird immer gebräuchlicher und der Zugriff auf die Ressourcen dieser Dienste, wie beispielsweise persönliche Benutzerdaten oder Nachrichten, geschieht häufig von unterschiedlichen Geräten aus.[2]

Dienstanbieter, die ihre Ressourcen anderen Diensten und Benutzern zur Verfügung stellen möchten, nutzen oftmals ein Application Programming Interface (API), um Zugriffe auf die hinterlegten Ressourcen zu ermöglichen.[3] Derartige Zugriffe erfordern meist eine Authentifizierung der einzelnen Benutzer, bei der ein Berechtigungsnachweis in Form von Benutzername und Passwort (Credentials) verwendet wird. Auf dieser Grundlage können dann weitere Rechte für die Autorisierung von Benutzerzugriffen vergeben werden. In der Vergangenheit gab es für die Umsetzung dieser beiden Vorgänge jedoch keinen einheitlichen Standard, was es zunehmend erschwerte, geeignete Techniken zur Authentifizierung und Autorisierung zu verwenden.[4]

Das Protokoll Open Authorization (OAuth) kann als schützende Schicht für einen Internetdienst wahrgenommen werden. Es stellt einen einheitlichen Weg zur Verfügung, um mit standardisierten Methoden auf geschütze Ressourcen eines Dienstes zuzugreifen, ohne dass sensible Daten wie Credentials an Dritte weitergeben werden müssen.[5] Viele der großen Internetplattformen haben OAuth in ihren Umgebungen integriert, um API-Zugriffe zu verwalten.[6] OAuth erlaubt einen Überblick darüber, wer mit dem System kommuniziert und erleichtert den Authentifzierungsvorgang für die Benutzer. Zusätzlich erhöht es die Sicherheit bei API-Zugriffen, da die angebotenen Dienste nicht mehr unmittelbar an die Authentifizierungs- und Autorisierungsvorgänge gekoppelt sind.[7]

Auch der Nachfolger OAuth 2 hat sich bei API-Zugriffen bewährt, denn der Großteil der Plattformen wie Facebook und Google sind bereits auf die zweite Version des Protokolls umgestiegen. Das Protokoll OAuth 2 ist nicht rückwärtskompatibel, da es zum Teil andere Ansätze verfolgt als sein Vorgänger. Es beschreibt verschiedene Abläufe für den Zugriff auf geschützte Ressourcen mit Web- und Desktopanwendungen sowie mobilen Geräten. Diese unterschiedlichen Anwendungen führen bei OAuth 2 Anfragen an vordefinierte Endpunkte der Dienste aus, um an die geschützten Ressourcen mittels eines Zugriffstokens zu gelangen. Der Zugriff geschieht dabei im Auftrag des Eigentümers der Ressourcen, der in Form des Tokens identfiziert wird.[8]

[1] Vgl. Evans, D.-S. / Hagiu, A. / Schmalensee, R. (2006), S. viii.
[2] Vgl. Hiller, J. / Lodderstedt, T. (2012), S.126.
[3] Vgl. LeBlanc, J. (2011), S. 535.
[4] Vgl. Boyd, R. (2012), S. vii.
[5] Vgl. Spasovski, M. (2013), S. 7.
[6] Vgl. Russel, M.-A. (2014), S. 403-404.
[7] Vgl. Spasovski, M. (2013), S. 11.
[8] Vgl. Spasovski, M. (2013), S. 8-10.

Die Zielsetzung dieser Arbeit ist es, eine Übersicht über das Protokoll OAuth 2 und dessen verschiedene Abläufe zu geben. Dabei soll auf die Unterschiede der Abläufe eingegangen werden, um einige Verwendungszwecke aufzeigen zu können. Es soll außerdem ein praktisches Beispiel für den Zugriff einer Webanwendung auf eine mit OAuth 2 abgesicherte API erstellt werden.

Die Arbeit ist folgendermaßen aufgebaut: Kapitel 2 behandelt die wichtigsten Grundlagen für die Verwendung des Protokolls OAuth 2. Es bringt den Leser der Arbeit auf einen zum Verständnis benötigten Wissensstand, indem auf Authentifizierung und Autorisierung, das Verschlüsselungsverfahen TLS und das Zusammenspiel von OAuth mit dem HTTP-Protokoll eingegangen wird. Dabei werden einzelne Begriffe erklärt und Beispiele im Zusammenhang mit deren Verwendung im Web gegeben. In Kapitel 3 wird das Protokoll OAuth 2 erläutert, dessen Funktionsweise beschrieben und eine Übersicht über die verschiedenen Abläufe des Protkolls gegeben. Weiterhin werden sicherheitsrelevante Aspekte diskutiert und es werden die Vorteile gegenüber der Authentifizierung mit HTTP dargestellt. Kapitel 4 beschreibt den Zugriff einer Webanwendung auf die Ressourcen der API von Dropbox. In Kapitel 5 erfolgt eine abschließende Betrachtung des Protokolls.

2 Grundlagen

2.1 Authentifizierung und Autorisierung

Als Authentifizierung bezeichnet man die nachweisliche Identifikation eines Benutzers in einem System. Im Speziellen dient sie im Web zur Überprüfung der Kommunikationspartner und soll aufzeigen, ob ein Nutzer eine Zugangsberechtigung besitzt oder nicht. Eine Authentifizierung ist besonders dann wichtig, wenn weitere Rechte an die Identität einer Person geknüpft sind, damit eine entsprechende Rechtezuordnung erfolgen kann.[9] Bei der Autorisierung handelt es sich hingegen um die Zuweisung von Zugriffsrechten für einen Benutzer, weshalb beide Begriffe oftmals miteinander verwechselt werden.[10]

> "As an aside, some people confuse authorization with authentication. Authentication deals with the question of whether you are actually communicating with a specific process. Authorization is concerned with what that process is permitted to do."[11]

Im Zusammenhang mit dem Web wird bei einer Authentifizierung die Zugehörigkeit eines Benutzers zu seinem Benutzerkonto überprüft. Dies wird meistens durch die Eingabe von Credentials sichergestellt. Der Benutzername ist dabei stellvertretend für die Indentität des Benutzers. Die Authentifizierung ist erfolgreich, wenn ein Benutzer das zu dem Benutzernamen dazugehörige Passwort verwendet. Obwohl einige Anwendungen ihre Credentials eigenständig verwalten, wird es im Webumfeld immer beliebter, diesen Vorgang an die Benutzerverwaltung von anderen Diensten zu koppeln.

[9] Vgl. Hansen, H.-R. / Neumann, G. (2007), S. 32-33.
[10] Vgl. Stahlknecht, P. / Hasenkamp, U. (2006), S. 303.
[11] Tanenbaum, A.-S. / Wetherall, D.-J. (2011), S. 827-828.

Bei einer Autorisierung innerhalb des Webs wird dagegen überprüft, welche Rechte ein Benutzer vorweisen kann, um bestimmte Aktionen auszuführen. Derartige Aktionen können beispielsweise das Lesen von bestimmten Dokumenten oder der Zugriff auf ein E-Mailkonto sein. Um die Rechte eines Benutzers überprüfen zu können, muss sich ein Benutzer im Vorfeld allerdings authentifizieren. Im Web und bei API-Zugriffen gibt es für diese Zwecke die Protokolle HTTP und OAuth.[12] Auf deren Funktionsweise wird im weiteren Verlauf dieser Arbeit genauer eingegangen.

2.2 Verschlüsselung mit TLS

Transport Layer Security (TLS) ist ein hybrides Verschlüsselungsprotokoll, bei dem sowohl symmetrische als auch asymmetrische Verschlüsselungsverfahren eingesetzt werden.[13] Es wird seitens der Internet Engineering Task Force (IETF) gepflegt und ist eine Weiterentwicklung von Secure Sockets Layer (SSL). Mittels des Protokolls kann ein sicherer Kanal zwischen einem Client und einem Server aufgebaut werden. Im August 2008 wurde TLS in der aktuellen Version 1.2 von der IETF veröffentlicht und in der Request for Comments (RFC) 5246 definiert.[14]

Die Kommunikation in Rechnernetzen geschieht in Form von Protokollen, welche häufig im Zusammenhang mit weiteren Protokollen in einem Protokollstapel dargestellt werden. Innerhalb des Internet-Protokollstapels, wie er in Abbildung 1 ersichtlich ist, wird TLS zwischen der Anwendungsschicht und der Transportschicht angesiedelt. Dadurch werden die Anforderungen eines Webbrowsers zur Verwendung eines sicheren Kanals entgegengenommen und an die Transportschicht übergeben.

Abbildung 1: Schichten und Protokolle bei der Verwendung von TLS im Internet
Quelle: Eigene Darstellung in Anlehnung an Tanenbaum, A.-S. / Wetherall, D.-J. (2011), S. 854.

[12] Vgl. Boyd, R. (2012), S. 4-5.
[13] Vgl. Wind, M. / Kröger, D. (2006), S. 280.
[14] Vgl. Dierks, T. / Rescorla, E. (2008), http://tools.ietf.org/html/rfc5246#section-1

Nachdem eine derartige Verbindung erfolgreich aufgebaut ist, liegt die Hauptaufgabe des Protokolls in der Verschlüsselung von Nachrichten. Die Verwendung von TLS zusammen mit dem Protokoll Hypertext Transfer Protocol (HTTP) wird als Secure HTTP (HTTPS) bezeichnet. Es ist der häufigste Anwendungsfall von TLS und wird von den meisten Webbrowsern wie Firefox, Google Chrome und dem Internet Explorer unterstützt.[15] Ein Benutzer muss sich nicht um die Verwendung des Protokolls kümmern, da eine angewählte HTTP-Adresse automatisch in HTTPS umgewandelt wird, sofern eine gesicherte Verbindung genutzt werden soll.[16]

Bei einer Verschlüsselung werden Informationen unter Verwendung eines Schlüssels unkenntlich gemacht. Verschlüsselte Informationen können anschließend mittels des richtigen Schlüssels zurück in ihre Ausgangsform überführt werden.[17] Bei symmetrischen Verschlüsselungsverfahren besitzen Nachrichtensender und -empfänger den gleichen geheimen Schlüssel. Asymmetrische Verschlüsselungsverfahren verwenden im Gegensatz dazu zwei verschiedene Schlüssel, die seitens des Empfängers bereitgestellt und vom selben Algorithmus erzeugt werden. Ein öffentlicher Schlüssel dient zum Verschlüsseln der Informationen und wird vorab dem Sender vom Empfänger bekannt gemacht. Eine verschlüsselte Nachricht kann nur mit Hilfe des dazugehörigen privaten Schlüssels, der nur dem Empfänger bekannt ist, entschlüsselt werden. In der Praxis wird meist eine Kombination der beiden Verfahren eingesetzt und mittels der asymmetrischen Verschlüsselung ein geheimer Schlüssel ausgetauscht, um damit große Datenmengen symmetrisch zu verschlüsseln.[18]

Die Einrichtung eines sicheren Kanals erfolgt bei TLS daher in zwei Phasen. Zu Beginn wird mittels asymmetrischer Verschlüsselungsverfahren ein gemeinsamer Schlüssel ausgehandelt. Im zweiten Schritt findet auf Grundlage einer symmetrischen Verschlüsselung mittels des Schlüssels die eigentliche Nachrichtenübertragung statt.[19] Die Integrität der Nachrichten wird durch einen Message Authentication Code (MAC) sichergestellt. Ein MAC ist eine Prüfsumme, die durch eine sichere Hash-Funktion errechnet wird.[20] Dies ist ein Algorithmus, der aus einer beliebigen Menge an Daten einen in der Länge festgelegten und eindeutigen Wert generiert.[21]

OAuth 2 baut stark auf TLS auf, weil durch dessen angebotene Verschlüsselungs- und Datenintegritätsmaßnahmen eine gewisse Vertaulichkeit bei dem Austausch von Daten zwischen Client und Server über das Netz gewährleistet wird. Die Spezifikation des Protokolls schreibt vor, TLS bei Anfragen an bestimmte Endpunkte zu nutzen. Bei der Verwendung von OAuth 1 war es dagegen nicht zwingend erforderlich, TLS für diese Zwecke zu verwenden. Stattdessen waren die Entwickler dafür verantwortlich, sich mit kryptografischen Verfahren bei dessen Verwendung auseinanderzusetzen.[22] Sicherheitsrelevante Überlegungen bei der Nutzung von TLS und die verschiedenen Endpunkte von OAuth 2 werden im weiteren Verlauf der Arbeit genauer beschrieben.

[15] Vgl. Tanenbaum, A.-S. / Van Steen, M. (2008), S. 626.
[16] Vgl. Hansen, H.-R. / Neumann, G. (2007), S. 374-375.
[17] Vgl. Hansen, H.-R. / Neumann, G. (2007), S. 424.
[18] Vgl. Wind, M. / Kröger, D. (2006), S. 279-280.
[19] Vgl. Tanenbaum, A.-S. / Van Steen, M. (2008), S. 626-627.
[20] Vgl. Dierks, T. / Rescorla, E. (2008), http://tools.ietf.org/html/rfc5246#section-1
[21] Vgl. Hansen, H.-R. / Neumann, G. (2007), S. 354.
[22] Vgl. Spasovski, M. (2013), S. 83.

2.3 Benennung von Ressourcen

Im Web wird unter Verwendung eines einzigen Benennungssystems auf Ressourcen verwiesen. Sogenannte Uniform Resource Identifier (URI) sind das Identifikationsmerkmal für eine Ressource im Web. Sie sind in der RFC 3986 beschrieben.[23]

Ein URI gibt an, an welchem Ort eine bestimmte Webressource gespeichert ist und wie auf die Ressource zugegriffen werden kann. Ein URI besteht im wesentlichen aus zwei Teilen. Den ersten Teil bildet die Bezeichnung des Kommunikationsprotokolls, also dem Verfahren, das besagt, wie auf eine Ressource zugegriffen wird. Bei der Verwendung von HTTP als Kommunikationsprotokoll wird demnach ein “http” gefolgt von einem Doppelpunkt vorangestellt. Der zweite Teil folgt nach zwei Schrägstrichen und ist der DNS-Name oder die IP-Adresse des Servers, auf dem sich die Ressource befindet. Ein DNS-Name ist der symbolische Name für einen Computer und wird oftmals anstatt der IP-Adresse verwendet. Die IP-Adresse dagegen ist die eindeutige Adresse eines Computers im Internet. Zusätzlich kann nach einem weiteren Doppelpunkt eine Portnummer angegeben werden. Diese Nummer besagt auf welchem Port ein Server auf Anfragen lauschen soll. Wird die Angabe der Portnummer weggelassen, kommt ein Standardport zum Einsatz. Bei HTTP ist dies standardmäßig der Port 80. Bei HTTPS, also HTTP über TSL, ist dies der Port 443. Außerdem enthält ein URI den Namen der Ressource, auf die zugegriffen werden soll.[24]

In Abbildung 2 werden einige Beispiele für häufig verwendete Strukturen von URIs dargestellt. In Beispiel a) wird ein DNS-Name als Hostname angegeben und auf die Angabe der Portnummer verzichtet. Die Beispiele b) und c) unterscheiden sich in der Verwendung von DNS-Name und IP-Adresse als Hostnamen, verfügen aber beide über die Angabe eines Ports.

a)

Verfahren		Hostname	Pfadname
http	://	www.dropbox.com	/home

b)

Verfahren		Hostname		Port	Pfadname
http	://	www.dropbox.com	:	80	/home

c)

Verfahren		Hostname		Port	Pfadname
http	://	108.160.166.13	:	80	/home

Abbildung 2: URI-Strukturen bei HTTP
Quelle: Eigene Darstellung in Anlehnung an Tanenbaum, A.-S. / Van Steen, M. (2008), S. 610.

[23] Vgl. Tanenbaum, A.-S. / Wetherall, D.-J. (2011), S. 683-684.
[24] Vgl. Berners-Lee, T. / Fielding R. / Masinter, L. (2005), http://tools.ietf.org/html/rfc3986#section-1

2.4 Datenübertragung mit HTTP

Für die Datenübertragung zwischen Client und Server ist das HTTP-Protokoll zuständig, welches in der aktuellen Version HTTP 1.1 in den RFCs 7230 - 7235 definiert wurde. HTTP ist ein Protokoll der Anwendungsschicht, das auf der Transportschicht aufsetzt und eng mit dem Web verbunden ist. Das Protokoll gibt an, welche Anfragenachrichten von einem Client aus an einen Server übertragen werden können und welche Antworten dabei auftreten können. Durch HTTP wurde die Verbreitung und Entwicklung des Web maßgeblich vorangetrieben, da es auf einem einfachen Modell beruht. Ursprünglich diente es dazu, Webseiten in einen Webbrowser zu laden. Die Verwendung von HTTP hat sich allerdings im Laufe der Zeit verändert. Es wird häufig auch als ein reines Transportprotokoll genutzt, weil es Anwendungen gestattet, über deren Netzwerksgrenzen hinaus Inhalte auszutauschen. Dabei wird es nicht unbedingt von einem Webbrowser oder -server verwendet, sondern kann beispielweise auch in Media-Playern dazu genutzt werden, um Informationen von verschiedenen Musikalben oder Updates herunterzuladen. Es ist zu erwarten, dass sich weitere derartige Nutzungsmöglichkeiten des Protokolls ergeben werden.[25] HTTP hat zudem die wichtige Eigenschaft, dass es zustandslos ist. Es verfügt über kein Konzept, wie mit Verbindungen, bei denen noch eine Antwort aussteht, umzugehen ist. Ein Server wird daher nicht gezwungen, Informationen über seine Clients aufzubewahren. Allerings können Seiteninhalte von Webservern im Zwischenspeicher der Clients aufbewahrt werden. Daher müssen Ressourcen von Webseiten, wie zum Beispiel Bilder von Navigationselementen, nicht bei jeder Anfrage erneut angefordert werden.

Üblicherweise wird bei einer HTTP-Verbindung zwischen einem Webbrowser und einem Server das Transmission Control Protocol (TCP) der Transportschicht verwendet, das in der RFC 793 definiert ist. Der Einsatz von TCP hat den Vorteil, dass sich weder Browser noch Server um verloren gegangene oder zu lange Nachrichten kümmern müssen. Sie können die Annahme treffen, dass ihre Nachrichten beim Ziel ankommen, denn TCP ist für die Übernahme derartiger Aufgaben verantwortlich. Bei Verbindungsfehlern oder dem Ablauf einer gewissen Zeitspanne bei einer Antwort (Timeout) zwischen Client und Server werden allerdings Meldungen ausgegeben. In älteren Versionen von HTTP war es problematisch, mehrere Anfragen an einen Server zu stellen, weil für jedes Anfrage-Antwort-Paar eine seperate TCP-Verbindung aufgebaut werden musste. Daher wird in der aktuellen Version HTTP 1.1 ein Ansatz verfolgt, bei dem Verbindungen dauerhaft oder bis zum Ablauf einer definierten Zeitspanne aufrechterhalten werden können, um mehrere Anfrage-Antwort-Paare innerhalb der Verbindung zu ermöglichen.[26]

HTTP wurde speziell für die Nutzung im Web entwickelt und unterstützt daher Anfragemethoden mit denen es möglich ist, auf Ressourcen eines Webservers zuzugreifen. Diese Methoden wurden dabei bewusst allgemein gehalten, um es für weitere Nutzungsmöglichkeiten verwendbar zu machen. Es können Ressourcen gelesen, geändert, angelegt oder gelöscht werden. Allerdings sind nicht unbedingt in jedem Webserver alle diese Methoden implementiert. Zudem können sie mit Restriktionen versehen sein, so dass beispielsweise nicht jeder Nutzer die Berechtigung besitzt, Ressourcen eines

[25] Vgl. Tanenbaum, A.-S. / Van Steen, M. (2008), S. 610.
[26] Vgl. Tanenbaum, A.-S. / Van Steen, M. (2008), S. 603-604.

Servers zu löschen oder anzulegen. Derartige Restriktionen werden normalerweise innerhalb eines Servers konfiguriert, daher unterscheiden sich viele Server in ihren nutzbaren Anfragemethoden.[27] In der Spezifikation von HTTP werden im Zusammenhang mit den Methoden, die Eigenschaften safe und idempotent beschrieben. Methoden die als safe deklariert sind, führen auf der Serverseite keine Aktionen aus und liefern daher nur Antworten an einen Client zurück. Idempotente Methoden haben die Eigenschaft, dass ein mehrfaches Ausführen der Anfragen nicht zu einem anderen Ergebnis führt.[28]

Die nachfolgende Tabelle 1 zeigt eine Liste der gebräuchlichsten Anfragemethoden bei HTTP und beschreibt diese. Die Methode GET fordert einen Server auf, eine Ressource abzurufen und an den Client zurückzugeben. Die meisten Anfragen im Webumfeld sind vom Typ GET. Die GET-Methode ist safe und idempotent. Die Methode PUT ist das Gegenstück zu GET und fordert den Server auf eine Ressource zu erzeugen oder zu aktualisieren. Die PUT-Methode ist ebenfalls idempotent. Dabei ist es allerdings nicht von Bedeutung, ob eine Ressource bereits auf der Serverseite vorhanden war oder nicht. Im Falle einer vorher nicht vorhandenen Ressource wird eine eindeutige Nummer vom Server für die neu angelegte Ressource vergeben. PUT benötigt des öfteren im Vorfeld eine Authentifizierung und Autorisierung, damit eine entsprechende Anfrage erfolgreich ausgeführt werden kann. Die Methode POST dient ebenfalls der Erzeugung oder Aktualisierung von Ressourcen, allerdings mit dem Unterschied, dass die eindeutige Nummer der Ressource bei der Anfrage dem Client bekannt sein muss. Die POST-Methode wird im Allgemeinen dafür verwendet, um einer Ressource oder einer Sammlung von Ressourcen weitere Daten hinzuzufügen oder das Versenden von Formularen zu ermöglichen. Eine genaue Aussage über die Funktion der Methode kann nicht getroffen werden, da sie von der jeweiligen Art der Implementierung auf der Serverseite abhängig ist. Die Methode DELETE wird dazu verwendet, Ressourcen zu entfernen. Die DELETE-Methode ist idempotent und ist ebenfalls wie PUT, oftmals mit einem Authentifizerungs- und Autorisierungsvorgang verbunden.[29]

Tabelle 1: Häufig verwendete Anfragemethoden bei HTTP

Methode	**Beschreibung**	**safe**	**idempotent**
GET	Ressource lesen	ja	ja
PUT	Ressource erzeugen oder aktualisieren	nein	ja
POST	Ressource erzeugen oder aktualisieren	nein	nein
DELETE	Ressource entfernen	nein	ja

Die Kommunikation zwischen einem Client und einem Server erfolgt bei HTTP über Nachrichten. Dabei wird zwischen Anfrage- und Antwortnachrichten unterschieden. Eine Anfrage bei HTTP (Request) besteht aus einer Kopfzeile (Header) gefolgt von weiteren meist optionalen Feldern. Nach einer Leerzeile folgt dann der eigentliche Datenteil der Nachricht. Dieser kann bei Anfragen auch leer ausfallen, wenn keine Ressourcen übertragen werden. In der Kopfzeile einer Anfrage steht die Anfragemethode zusammen mit einem URI für die angeforderte Ressource sowie der verwendeten Protokollversion.

[27] Vgl. Gourley, D. u.a. (2002), S. 53.
[28] Vgl. Fielding, R. / Reschke, J. (2014), http://tools.ietf.org/html/rfc7231#section-4.2
[29] Vgl. Fielding, R. / Reschke, J. (2014), http://tools.ietf.org/html/rfc7231#section-4.3

Nach dem Erhalt einer Anfrage versendet ein Webserver seine Antwort (Response). Diese setzt sich ebenfalls aus einer Kopfzeile mit weiteren optionalen Feldern zusammen. Nach einer Leerzeile kann ebenfalls ein Datenteil folgen. Die Kopfzeile einer Antwort enthält einen Statuscode und eine Statusmeldung, welche angeben, wie eine Anfrage verarbeitet werden konnte. Zusätzlich beinhaltet die Kopfzeile auch die verwendete Version des Protokolls für die Antwort.[30]

In Tabelle 2 werden die Statuscodes in Gruppen eingeteilt dargestellt. Statuscodes, die mit einer 1 beginnen, dienen als Information an einen Client, wenn die Bearbeitung einer Anfrage noch andauert. Eine solche Zwischenantwort ist in manchen Fällen sinnvoll, da ein Client ansonsten nach einem Timeout einen Fehlerfall annimmt. Statuscodes, die mit einer 2 beginnen, weisen auf erfolgreich ausgeführte Anfragen hin. Dabei wird die Antwort gegebenenfalls mit einer Repräsentation einer Ressource an einen Client zurückgeliefert. Bei Statuscodes, die mit einer 3 anfangen, sind noch zusätzliche Schritte seitens des Clients nötig. So kann beispelsweise von einem Server mitgeteilt werden, dass an einer anderen Stelle nach den benötigten Informationen gesucht werden muss. Statuscodes, die mit einer 4 beginnen, deuten auf Fehler der Clientseite hin. Dies kann bedeuten, dass eine Ressource nicht mehr verfügbar ist. Statuscodes, die mit der 5 anfangen, zeigen hingegen auf, dass sich die Ursache eines Fehlers auf dem Server befindet. Diese Codes können auf Grund von Überlastungen oder bei Fehlern im Quellcode des Servers auftreten.[31]

Tabelle 2: Gruppen von Statuscodes bei HTTP

Statuscode	**Bedeutung**	**Beispiel**
1xx	Information	100 = Server stimmt zu, die Anfrage des Clients zu bearbeiten
2xx	Erfolg	200 = erfolgreiche Anfrage; 204 = kein Inhalt vorhanden
3xx	Umleitung	301 = Seite wurde verlagert; 304 Seite im Cache noch gültig
4xx	Fehler am Client	403 = Seite ist verboten; 404 Seite nicht gefunden
5xx	Fehler am Server	500 = interner Serverfehler; 503 = versuche es später nochmal

Quelle: Tanenbaum, A.-S. / Wetherall, D.-J. (2011), S. 688.

Die Anfrage- und Antwortnachrichten können bei HTTP zusätzlich zu den Anfragemethoden innerhalb einer Kopfzeile weitere Felder enthalten, die dem Nutzdatenteil vorgelagert sind. Es gibt eine Vielzahl unterschiedlicher Headerfelder, die wichtige Parameter und Argumente für die Datenübertragung enthalten. Einige dienen dazu, einem Server mitzuteilen, welche Arten von Antworten ein Client akzeptiert und daher übertragen werden können. Ein Client, der beispielweise in der Lage ist, Webseiten darzustellen, könnte dies mit einem entsprechenden Accept-Headerfeld einem Server mitteilen. Eine andere Variante könnte sein, dass ein Server einem Client antwortet, dass er die Kommunikation nur unter Verwendung von HTTPS fortsetzt. In diesem Beispiel würde der Server ein Upgrade-Headerfeld versenden. In Tabelle 3 ist ein Teil der verschiedenen Headerfelder aufgeführt. Sie beinhaltet die bereits beschriebenen und einige für die Verwendung mit OAuth geeignete Headerfelder.[32]

[30] Vgl. Hansen, H.-R. / Neumann, G. (2007), S. 171.
[31] Vgl. Tanenbaum, A.-S. / Wetherall, D.-J. (2011), S. 687.
[32] Vgl. Tanenbaum, A.-S. / Van Steen, M. (2008), S. 606-608.

Tabelle 3: Headerfelder bei HTTP

Headerfeld	**Quelle**	**Bedeutung**
Accept	Client	Dokumententypen, mit denen der Client umgehen kann
Authorization	Client	Anmeldeinformationen des Clients
Expires	Server	Zeitdauer, für die die Antwort Gültigkeit behält
Host	Client	Name des Servers, auf dem sich die Ressource befindet
Location	Server	Verweis auf den Ort einer Ressource
Upgrade	Client und Server	Das Protokoll, zu dem der Absender wechseln will
WWW-Authenticate	Server	Sicherheitsaufforderung, die der Client beantworten muss

Quelle: Tanenbaum, A.-S. / Van Steen, M. (2008), S. 607.

Nachdem ein Überblick über mögliche Anfragemethoden, Statuscodes und den Aufbau von Nachrichten gegeben wurde, wird im Folgenden ein Beispiel für ein Anfrage-Antwort-Paar aufgezeigt. In Listing 1 wird die PUT-Methode verwendet, um eine Ressource auf einem Server zu erzeugen.

Listing 1: Beispiel einer HTTP-Anfrage mit der PUT-Methode

```
PUT /resources/1 HTTP/1.1
Host: www.server.com
...
```

Eine mögliche Antwort auf die zuvor versendete Anfrage ist in Listing 2 dargestellt, wobei das Anlegen der Ressource vom Server erfolgreich ausgeführt wurde und der Ort in Form eines URI im Location-Headerfeld zurückgegeben wird.

Listing 2: Beispiel einer Antwortnachricht in HTTP

```
HTTP/1.1 201 Created
Location: http://www.server.com/resources/1
...
```

Die Spezifikation von OAuth 2 schreibt die Nutzung mit HTTP vor und baut daher stark auf den beschriebenen Funktionsweisen dieses Protokolls auf. Es werden unter anderem Headerfelder und Statuscodes genutzt, um die Authentifizierung und Autorisierung mit OAuth 2 zu ermöglichen.[33]

[33] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749

2.5 HTTP-Authentifizierung

Das Protokoll HTTP bietet bereits eine Unterstützung für die Authentifizierung und Autorisierung bei Zugriffen auf Ressourcen im Web, welche in der RFC 7235 beschrieben ist. Wenn eine HTTP-Anfrage auf eine geschützte Ressource stattfindet, kann ein Server mit dem Statuscode 401 “Unauthorized” und dem WWW-Authenticate-Headerfeld antworten. Der Statuscode 401 weist auf eine unautorisierte Anfrage hin, welche die Angabe von Benutzername und Passwort erfordert. Das Headerfeld beschreibt hingegen, welches Schema zur Authentifizierung und Autorisierung vom Server akzeptiert wird.[34]

Basic- und Digest Authentication, welche in RFC 2617 definiert sind, stellen die häufigsten Arten der HTTP-Authentifizierung dar und können somit als ein gültiges Schema genutzt werden. Ein Server kann dabei eine Authentifizierung anfordern, welche in einem Webbrowser dazu führt, dass Credentials für den Zugriff auf eine Ressource benötigt werden und gegebenenfalls ein Dialog für deren Eingabe erscheint. Im Anschluss sendet der Webbrowser die Anfrage erneut ab und fügt darin ein Authorization-Headerfeld ein, welches die Credentials kodiert enthält. Eine Kodierung ist die Zuordnung von Bitkombinationen zu den Zeichen eines Zeichenvorrats oder einem Alphabet. Üblicherweise findet bei Basic Authentication die Kodierung mit dem Verfahren Base64 statt.[35]

Listing 3 zeigt eine Anfrage auf eine geschützte Ressource mit der HTTP-Methode GET. In diesem Fall wird die Repräsentation einer Ressource angefordert, allerdings wird unterstellt, dass diese nur angemeldeten Benutzern zur Verfügung steht, da die Ressource durch die HTTP-Authentifizierung geschützt sein soll.

Listing 3: Anfrage auf eine Ressource mit der GET-Methode

```
GET /resources/1 HTTP/1.1
Host: www.server.com
...
```

Listing 4 zeigt eine mögliche Antwort des Servers mit der HTTP-Statusmeldung 401 und dem WWW-Authenticate-Headerfeld. Der Parameter “Realm” ist dabei stellvertretend für den geschützten Bereich auf dem Server.

Listing 4: Antwort mit dem Statuscode 401

```
HTTP/1.1 401 Unauthorized
www-Authenticate: Basic realm="Realm"
...
```

[34] Vgl. Fielding, R. / Reschke, J. (2014), http://tools.ietf.org/html/rfc7235#section-4.1
[35] Vgl. Allmaraju / S. (2010), S. 218.

Innerhalb eines Webbrowsers wie dem Internet Explorer wird ein Benutzer anschließend zur Eingabe seiner Credentials aufgefordert. Ein solcher Eingabedialog ist in Abbildung 3 ersichtlich.

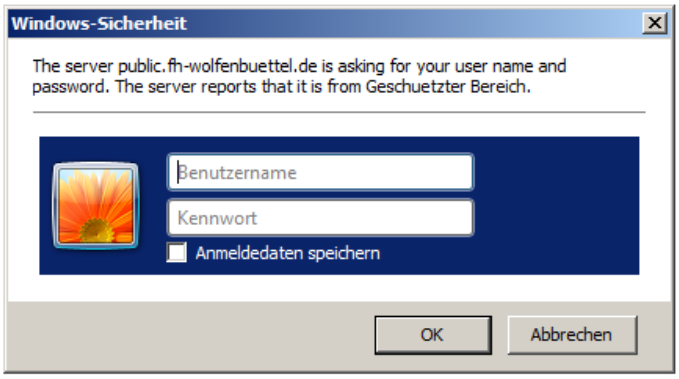

Abbildung 3: Eingabedialog für Benutzername und Passwort bei Basic Authentication

Nach der Eingabe von Benutzername und Passwort werden die Credentials mittels Base64 kodiert und mit dem Authorization-Headerfeld unter Angabe des genutzten Schemas an den Server übertragen. Listing 5 zeigt eine derartige Anfrage.

Listing 5: Anfrage mit dem Authorization-Headerfeld

```
GET /resources/1 HTTP/1.1
Host: www.server.com
Authorization: Basic T0F1dGg=
...
```

Das Verfahren Base64 ist in der RFC 4648 definiert und dient zur Kodierung von 8-Bit Binärdateien in 6-Bit Zeichenfolgen.[36] Es wird überwiegend bei dem Versand von Anlagen in E-Mails verwendet, um Dateien wie beispielsweise Bilder in unveränderter Form zu übertragen, denn das zu Grunde liegende Transportprotokoll Simple Mail Transfer Protocol (SMTP) war ursprünglich nur für den Versand von 7-Bit Textnachrichten mit dem Zeichensatz American Standard Code for Information Interchange (ASCII) ausgelegt.[37]

Bei der Kodierung mit Base64 werden 24 Bits eines Datenstroms in vier Blöcke mit jeweils sechs Bits zerlegt. Jeder dieser Blöcke stellt dezimal einen Wert im Bereich von 0 bis 63 dar. Die Werte werden mittels einer Umsetzungstabelle, wie sie in Tabelle 4 ersichtlich ist, in lesbare Zeichen umgewandelt.[38]

[36] Vgl. Josefsson, S. (2006), http://tools.ietf.org/html/rfc4648#section-4
[37] Vgl. Hansen, H.-R. / Neumann, G. (2007), S. 48.
[38] Vgl. Tanenbaum, A.-S. / Wetherall, D.-J. (2011), S. 634.

Tabelle 4: Das Base64-Alphabet

Zahl	Zeichen	Zahl	Zeichen	Zahl	Zeichen	Zahl	Zeichen
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Quelle: Josefsson, S. (2006), http://tools.ietf.org/html/rfc4648

Die Tabelle enthält zu jedem der 64 Werte ein dazugehöriges Zeichen, das sich aus dem Alphabet in Groß- und Kleinschreibung sowie den Ziffern 1 bis 9 und weiteren Zeichen ergibt. Es können mit drei Bytes des Datenstroms insgesamt vier verschiedene lesbare Zeichen dargestellt werden. Falls der Datenstrom nicht genügend Bytes enthält, um ihn in die vier Blöcke aufzuteilen, wird er am Ende mit Nullbits aufgefüllt. Wenn ein vollständiges Byte mit Nullbits versehen ist, wird dies mit dem Gleichheitszeichen dargestellt.[39]

Tabelle 5 zeigt eine Beispielkodierung einer ASCII-Zeichenfolge mit dem Verfahren Base64. Die Zeichenfolge wird dabei zunächst an Hand seiner dezimalen Werte innerhalb der ASCII-Tabelle in seine binäre Form überführt und anschließend mit Base64 kodiert.

Tabelle 5: Beispielkodierung mit Base64

ASCII	O	A	u	t	h			
dezimal	79	65	117	116	104			
8-Bit	01001111	01000001	01110101	01110100	01101000			
6-Bit	010011	110100	000101	110101	011101	000110	100000	000000
dezimal	19	52	5	53	29	6	32	
Base64	T	0	F	1	d	G	g	=

[39] Vgl. Tanenbaum, A.-S. / Wetherall, D.-J. (2011), S. 634.

Basic Authentication ist ein einfaches Schema, das allerdings Schwächen offenbart. Die Zeichenfolge “T0F1dGg=” vermittelt zwar den Eindruck, dass die zu Grunde liegenden Daten verschlüsselt sind, jedoch lässt sich ein mit Base64 kodierter Text mit Kenntnissen des Verfahrens leicht wieder in seine ursprüngliche Form überführen. Mit dem Einsatz von HTTPS bei der Datenübertagung können die Daten jedoch über einen gesicherten Kanal übertragen werden.[40] Die Verwendung von Basic Authentication führt außerdem dazu, dass die Credentials bei jeder Anfrage im Authorization-Headerfeld von HTTP mitgeschickt werden müssen, was es lauschenden Angreifern einfacher macht, an diese zu gelangen.[41] Das Schema Besic Authentication wird auch bei OAuth 2 eingesetzt, um an ein Access Token zu gelangen.[42]

Die Abfrage von Benutzername und Passwort erfolgt auch bei dem Schema Digest Authentication, welches ebenfalls in der RFC 2617 definiert ist. Allerdings werden die Credentials dabei nicht mehr in kodierter Form versendet, sondern es wird stellvertretend eine Prüfsumme verwendet. Der Nachrichteninhalt beibt auch hierbei unverschlüsselt.[43]

2.6 RESTful APIs

Der Begriff Representational State Transfer (REST) wurde im Jahr 2000 von Roy Fielding eingeführt. Gemeint ist ein Architekturstil, der folgende Kernprinzipien für Webanwendungen identifiziert:

- Eindeutige Identifikation von Ressourcen
- Unterschiedliche Reprasentationsformen von Ressourcen
- Verknüpfung von Inhalten (Hypermedia)
- Zustandslose Kommunikation zwischen Client und Server
- Standardmethoden für den Zugriff auf Ressourcen

Im Web gibt es ein einheitliches Konzept für die Benennung von Ressourcen. Die Verwendung von URIs stellt sicher, dass Ressourcen weltweit eindeutig identifiziert werden können.

Es gibt unterschiedliche Repräsentationsformen von Ressourcen. Ein Client kann bei einer HTTP-Anfrage mit der GET-Methode einem Server mit dem Accept-Headerfeld mitteilen, in welchem Datenformat (Media Type) er die gewünschte Ressource zur Verfügung gestellt bekommen möchte.

Die Verbindung von Ressourcen wird über Verknüpfungen (Links) gesteuert. Diese können auch in den Repräsentationen der Ressourcen enthalten sein, so dass eine Ressource auf eine andere verweisen kann. Anwendungen können die Verknüpfungen auswerten oder dem Benutzer präsentieren.

[40] Vgl. Richardson, L. / Amundsen, M. (2013), S. 250-251.
[41] Vgl. LeBlanc, J. (2011), S. 317.
[42] Vgl. Allmaraju / S. (2010), S. 220.
[43] Vgl. Franks, J. / Hallam-Baker, P. / Hostetler, J. u.a. (1999), http://tools.ietf.org/html/rfc2617#section-3

Jede Repräsentation einer Ressource, die ein Client von einem Server erhält, repräsentiert auch den Status der Benutzerinteraktion innerhalb einer Anwendung. Durch die Verknüpfung der Ressourcen mit Links werden einem Client oft die nächst möglichen Aktionen vorgegeben.

Clients nutzen das HTTP-Protokoll mit den dazugehörigen Methoden für Anfragen auf Ressourcen im Web. Das Protokoll stellt somit ein einheitliches Interface für den Zugriff auf Ressourcen dar.[44]

Die APIs von Diensten im Webumfeld folgen meist den Prinzipien einer REST-Architektur. Innerhalb der Dienste werden mehrere Endpunkte für Anfragen definiert. Diese Endpunkte können abhängig von den zur Verfügung gestellten Ressourcen öffentlich zugänglich oder durch Autorisierungsmechanismen wie dem OAuth-Protkoll geschützt sein. Derartige APIs kommunizieren mit HTTP und basieren daher auf den Funktionalitäten des Protokolls. Ein Client stellt eine Anfrage an einen Dienst, der sich auf einem Server befindet. Dieser verarbeitet den Request und antwortet mit einem HTTP-Statuscode und der Repräsentation einer Ressource.[45]

3 Authentifizierung und Autorisierung mit OAuth 2

3.1 Entstehung und Nutzen des Protokolls

Die Arbeiten am OAuth-Protokoll begannen im Jahr 2006 bei dem Versuch delegierte Zugriffe bei der Nutzung von APIs zu ermöglichen. Für einen Zugriff auf geschützte Ressourcen eines Servers muss sich ein Client am Server authentifizieren. Der Eigentümer der Ressourcen muss bei diesem Vorgang seine Credentials mit der Clientanwendung teilen, damit der Zugriff in seinem Namen stattfinden kann. Da es keinen einheitlichen Standard für API-Zugriffe gab, entschloss sich eine Gruppe von Webentwicklern dazu, selbst eine Spezifikation für diesen Zweck zu entwerfen. Ein erster Spezifikationsentwurf erschien im Juli 2007 und die erste Version der Spezifikation von OAuth wurde im Oktober 2007 veröffentlicht.[46] Auf Grund einer Sicherheitslücke und weiteren Optimierungsnotwendigkeiten wurde das Protokoll weiter überarbeitet.[47] Im April 2010 erschien die Spezifikation von OAuth 1.0 in der RFC 5849.

OAuth hat sich seitdem als ein geeignetes Mittel etabliert, um Anwendungen den API-Zugriff im Auftrag des Eigentümers zu ermöglichen. Duch die Nutzung des Protokolls werden die Credentials der Benutzer nicht unnötig im Web verbreitet und Zugriffe von dritten Anwendungen transparent gehalten. Anstatt die Credentials der Eigentümer von Ressourcen beim Zugriff zu verwenden, erhält die Clientanwendung ein Zugriffstoken. Der Autor der Spezifikation Eran Hammer-Lahav vergleicht das Protokoll mit der Vergabe eines Werkstattschlüssels bei der Nutzung von luxuriösen Autos.[48]

[44] Vgl. Tilkov, S. (2011), S. 9-17.
[45] Vgl. LeBlanc, J. (2011), S. 535.
[46] Vgl. Makice, K. (2009), S. 188.
[47] Vgl. Hammer-Lahav, E. (2009), http://oauth.net/advisories/2009-1/
[48] Vgl. Makice, K. (2009), S. 188.

> “Many luxury cars today come with a valet key. It is a special key you give the parking attendant and unlike your regular key, will not allow the car to drive more than a mile or two. Some valet keys will not open the trunk, while others will block access to your onboard cell phone address book. Regardless of what restrictions the valet key imposes, the idea is very clever. You give someone limited access to your car with a special key, while using your regular key to unlock everything.”[49]

Ein Benutzer dritter Anwendungen wird mittels OAuth zu einer vertrauten Oberfläche von einer ihm bekannten Plattform weitergeleitet und kann dort benötigte Zugriffsrechte auf seine Ressourcen zuteilen. In der Praxis bieten diese Anwendungen ihren Benutzern die Möglichkeit, sich durch OAuth mit ihren Accounts von anderen Plattformen zu authentifizieren und auf dieser Grundlage den Zugriff auf ihre Inhalte zu autorisieren.[50]

Da sich die Implementierung von OAuth 1 allerdings als aufwendig erwies, wurde das Protokoll seitens der IETF ebenfalls überarbeitet. Im Oktober 2012 erschien die aktuelle und zur Version 1 inkompatible Spezifikation von OAuth 2.0 in der RFC 6749, mit der vor allem die Handhabung von Tokens erleichtert wird. Darüber hinaus wurden weitere Protkollabläufe ergänzt.[51] Abbildung 4 zeigt einen Authentifizierungsdialog der Google API unter Verwendung des OAuth-Protokolls.

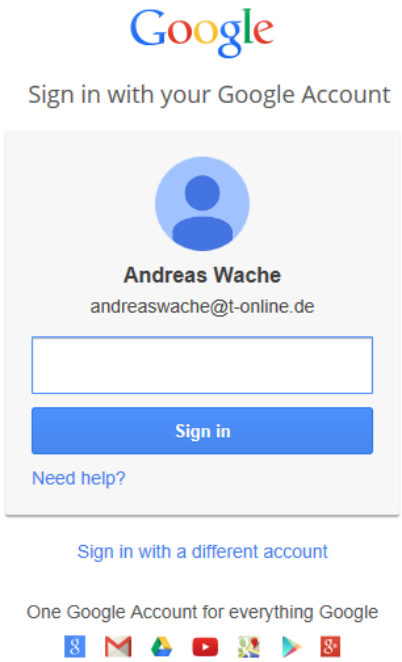

Abbildung 4: Authentifizierungdsdialog der Google API

[49] Hammer-Lahav, E. (2007), http://hueniverse.com/2007/09/05/explaining-oauth/
[50] Vgl. Makice, K. (2009), S. 188.
[51] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-1

3.2 Begriffsdefinitionen

3.2.1 Rollen

Im Zusammenhang mit dem OAuth-Protokoll und dessen Abläufen werden vier verschiedene Rollen definiert, die im Folgenden genauer erläutert werden:

- Resource Owner
- Resource Server
- Client
- Authorization Server

Ein Resource Owner agiert bei OAuth 2 als Endanwender und ist meist der Benutzer einer Anwendung, die auf geschütze Ressourcen zugreifen möchte. Er ist der Eigentümer von geschützten Ressourcen, die sich auf dem Resource Server befinden und ist fähig, den Zugriff auf diese Ressourcen zu gewähren oder zu verweigern.

Der Resource Server ist ein Server, auf dem sich die Ressourcen des Resource Owners befinden. Er verwaltet die Ressourcen und verlangt bei Zugriffen auf diese Ressourcen ein Zugrffstoken.

Ein Client ist eine Anwendung, die Anfragen im Namen des Resource Owners an geschützte Ressourcen auf dem Resource Server stellt. Dazu benötigt der Client ein Zugriffstoken (Access Token). Eine Clientanwendung kann bei OAuth 2 eine Webanwendung, eine Desktopanwendung oder eine mobile Anwendung sein.

Der Authorization Server vergibt bei OAuth 2 die Access Tokens, mit denen ein Zugriff auf geschütze Ressourcen des Resource Servers ermöglicht wird. Ein Resource Owner muss sich dazu am Authorization Server authentifizieren und die jeweiligen Clientanwendungen für Zugriffe auf diese Ressourcen autorisieren.[52]

Ein Client stellt im Namen des Resource Owners HTTPS-Anfragen an den Resource Server. Damit der Client auf die geschützten Ressourcen zugreifen kann, muss dieser in seinen Anfragen ein Access Token mitführen. Das Access Token autorisiert den Client für dessen Zugriffe. Der Resource Server kommuniziert im weiteren Verlauf mit dem Authorization Server und überprüft das Token.[53]

Die Kommunikation zwischen Resource Server und Authorization Server wird nicht genau in der Spezifikation von OAuth 2 festgelegt, weshalb in der Praxis verschiedene Konstellationen vorkommen können. Sie können sich auf dem selben Server oder auf separaten Servern befinden. Zusätzlich gibt es Umgebungen, bei denen ein Authorization Server mit mehreren Resource Servern kommuniziert.[54]

[52] Vgl. Spasovski, M. (2013), S. 13-15.
[53] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-1.1
[54] Vgl. Spasovski, M. (2013), S. 14.

3.2.2 Abstrakter Workflow

Das Protokoll OAuth 2 definiert mehrere Abläufe, auch Flows genannt. Die verschiedenen Flows sind notwendig, da die unterschiedlichen Clientanwendungen bei Oauth 2 zum Teil unterschiedliche Anforderungen an den Protokollablauf stellen und andere Möglichkeiten besitzen. Eine Webanwendung, die sich auf einem Server befindet, kann beispielsweise einen mit dem Authorization Server gemeinsamen Schlüssel so verwalten, dass dieser nicht freigelegt wird. Anders verhält sich dieser Umstand allerdings bei Clientanwendungen, die aus einem Webbrowser heraus aufgerufen werden und eine Skriptsprache wie JavaScript verwenden. Der Quellcode dieser Skriptsprachen kann direkt im Browser aufgerufen werden, weshalb kein gemeinsamer Schlüssel darin abgelegt werden sollte. Innerhalb der Spezifikation von OAuth 2 werden vier verschiedene Flows definiert. Die Gemeinsamkeiten der verschiedenen Flows sind in einem abstrakten Workflow beschrieben, der in Abbildung 5 ersichtlich ist.[55]

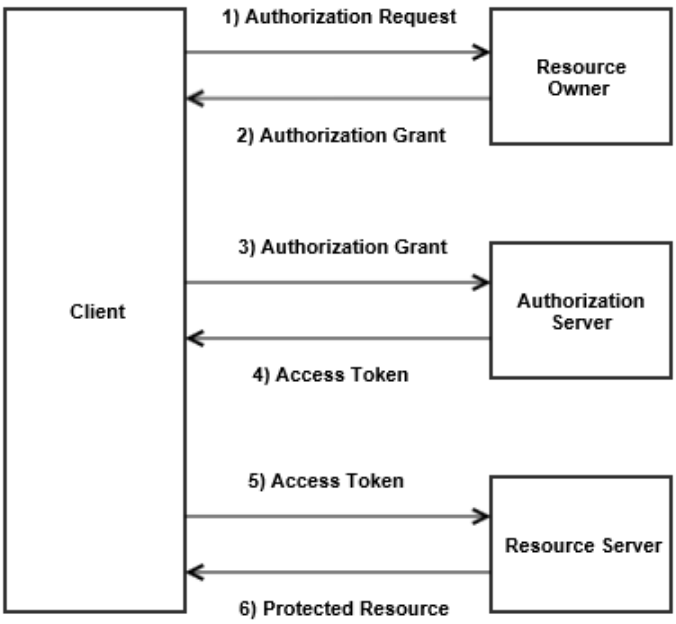

Abbildung 5: Abstrakter Workflow von OAuth 2
Quelle: Hardt, D. (2012), http://tools.ietf.org/html/rfc6749

Der abstrakte Workflow von OAuth 2 beschreibt die Interaktionen zwischen den einzelnen Rollen des Protokolls mittels Anfragen. Er stellt einen abstraktes Ablauf dar und zeigt auf, wie ein Client autorisiert wird, ein Token bekommt und schließlich den Zugriff auf geschützte Ressourcen erhält. Folgende Schritte werden dabei definiert.

[55] Vgl. Spasovski, M. (2013), S. 16-17.

Die ersten beiden Anfragen beziehen sich auf das Zusammenspiel zwischen dem Client und dem Resource Owner:

(1) Der Client beantragt die Autorisierung für den Zugriff auf geschützte Ressourcen vom Resource Owner. Die Anfrage für die Autorisierung kann dabei, wie in Abbildung 5 ersichtlich ist, direkt an den Resource Owner oder an den Authorization Server, welcher als Vermittler agiert, gestellt werden.

(2) Im Anschluß erhält der Client den Authorization Grant, der die Autorisierung des Resource Owners repräsentiert und dessen Form von der Wahl des genutzen Flows abhängig ist.

Die nächsten beiden Anfragen beschreiben die Interaktion zwischen dem Client und dem Authorization Server:

(3) Der Client fragt mittels des Authorization Grants das Access Token beim Authorization Server an.

(4) Der Authorization Server prüft nach, ob der Authorization Grant in geeigneter Form vorliegt und tauscht den Authorization Grant gegebenfalls gegen das Access Token ein. Dabei wird auch die Identität des Clients überprüft.

Die letzten beiden Anfragen beschäftigen sich mit der Kommunikation zwischen Client und Resource Server:

(5) Der Client fragt nach der geschützten Ressource, die sich auf dem Resource Server befindet. Dazu päsentiert er dem Resource Server das Access Token.

(6) Der Resource Server prüft das Access Token und der Zugriff erfolgt.[56]

3.2.3 Authorization Grant

Ein Authorization Grant ist eine Zugriffserteilung und repräsentiert die Autorisierung des Resource Owners zum Zugriff auf geschützte Ressourcen und wird seitens des Clients genutzt, um ein Access Token zu erhalten. Innerhalb der Spezifikation von OAuth 2 werden die nachfolgenden vier Authorization Grants definiert. Zusätzlich ermöglicht die Spezifikation des Protokolls einen Mechanismus für die Definition von weiteren Zuggriffserteilungen.[57]

- Authorization Code
- Implicit
- Resource Owner Passwort Credentials
- Client Credentials

[56] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-1.2
[57] Vgl. Spasovski, M. (2013), S. 17-18.

Ein Authorization Code wird erzeugt, wenn ein Authorization Server als Vermittler zwischen Client und Resource Owner genutzt wird. Anstatt die Autorisierung direkt beim Resource Owner zu beantragen, leitet ein Client den Resource Owner zu einem Authorization Server weiter. Der Resource Owner wird nach einer erfolgreichen Authentifizierung am Authorization Server zurück zum Client geleitet und erhält den Authorization Code. Der Client erhält dadurch keine Einsicht in die Credentials des Resource Owners.

Der Implicit Grant ist eine vereinfachte Variante des Authorization Code Grants und wird bei Clients genutzt, die innerhalb eines Webbrowsers laufen und mit Skriptsprachen wie beispielsweise JavaScript umgesetzt sind. Anstatt dem Client einen Authorization Code auszustellen, um damit an ein Access Token vom Authorization Server gelangen zu können, erhält er direkt nach der Authentifizierung des Resource Owners das Access Token. Der Implicit Grant verbessert die Effektivität derartiger Clients, da die Anzahl von Anfragen zwischen Client und Authorizaton Server reduziert wird.

Resource Owner Passwort Cedentials wie beispielsweise Benutzername und Passwort eines Benutzers können ebenfalls dazu genutzt werden, um ein Access Token zu erlangen. Allerdings sollte dieser Authorization Grant nur dann benutzt werden, wenn der Client als vertrauenswürdig seitens des Resource Owners eingestuft werden kann und die anderen Authorization Grants nicht benutzt werden können. Dies ist beispielsweise der Fall, wenn eine Facebook-Anwendung nach den Zugangsdaten der Benutzer von Facebook fragt. Obwohl die Nutzung dieses Authorization Grants dazu führt, dass der Client direkten Zugang zu den Credentials des Resource Owners bekommt, dient er dazu, nach einer einmaligen Anfrage mit den Credentials ein Access Token vom Authorization Server zu erhalten. Dadurch wird vermieden, dass ein Client die Credentials der Benutzer verwalten muss. Stattdessen kann er die Access Tokens der jeweiligen Benutzer abspeichern und diese gegebenfalls vom Authorization Server erneuern lassen.

Der Client Credentials Grant ermöglicht die Nutzung von Credentials eines Clients für das Erlangen eines Access Token und kann genutzt werden, wenn der Client gleichzeitig der Resource Owner ist und in seinem eigenen Namen auf die geschützen Ressourcen zugreifen möchte. Üblicherweise liegen die Client Credentials in Form von Client ID und Client Secret bei der Nutzung von OAuth 2 vor.[58]

3.2.4 Tokens

In der Spezifikation von OAuth 2 sind folgende zwei Arten von Tokens definiert:

- Access Token
- Refresh Token

[58] Vgl. Spasovski, M. (2013), S. 17-18.

Das Access Token dient als Zugriffstoken und wird seitens der Clients genutzt, um Zugriffe auf geschütze Ressourcen durchzuführen. Nachdem ein Benutzer eine Clientanwendung für den Zugriff auf bestimmte Ressourcen autorisiert hat, bekommt die Anwendung vom Authorization Server ein Access Token geliefert. Das Token ist mit dem Benutzer verknüpft und dient als Ersatz für dessen Credentials beim Ressourcenzugriff, weshalb die Anfragen bei OAuth 2 im Namen des Benutzers stattfinden. Access Tokens sollten immer mit einer Lebensdauer behaftet sein, damit ein Zugriff auf Ressourcen kontrolliert abläuft.[59]

Bei einem versuchten Zugriff auf eine geschützte Ressource mit einem abgelaufenen Access Token, wird seitens des Resource Servers eine Fehlermeldung ausgegeben. Jedoch ist es möglich, neue Access Tokens beim Authorization Server anzufordern, falls die Lebensdauer eines solchen Tokens überschritten ist. Für diesen Zweck gibt es bei OAuth 2 sogenannte Refresh Tokens. Ein Client nutzt ein Refresh Token, um beim Authorization Server ein neues Access Token zu erfragen. Der Flow für die Anforderung eines neuen Access Tokens unter Verwendung eines Refresh Tokens ist in Abbildung 6 dargestellt.[60]

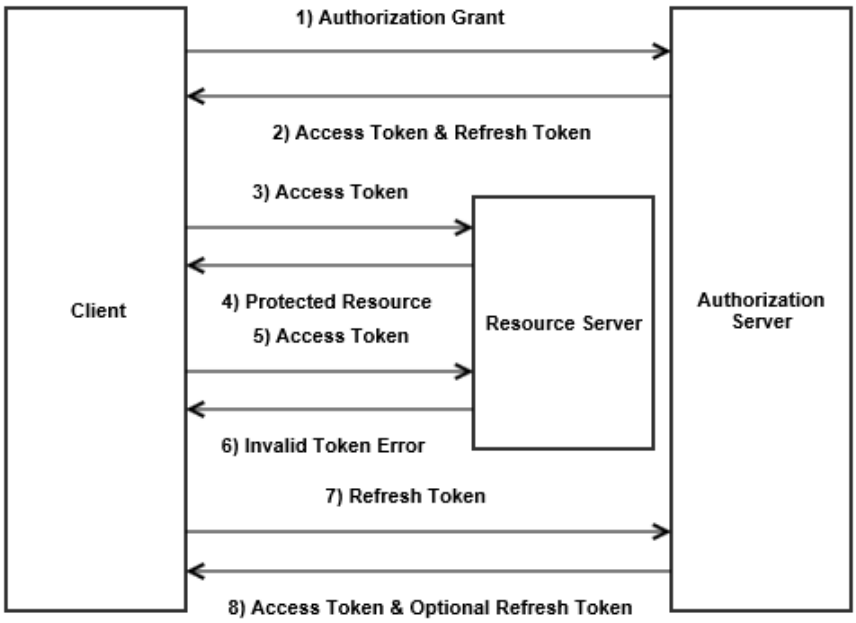

Abbildung 6: Erneuerung eines abgelaufenen Access Tokens
Quelle: Hardt, D. (2012), http://tools.ietf.org/html/rfc6749

[59] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-1.5
[60] Vgl. Spasovski, M. (2013), S. 17-18.

Der Flow für die Anforderung eines neuen Access Tokens beinhaltet folgende Schritte:

(1) Der Client fragt ein Access Token an, indem er sich mit dem Authorization Grant am Authorization Server authentifiziert.

(2) Der Authorization Server überprüft den Authorization Grant und übergibt dem Client ein Access Token und ein Refresh Token, sofern eine Authentifizierung erfolgt ist.

(3) Anschließend versucht der Client auf eine geschütze Ressource des Ressource Servers zuzugreifen.

(4) Der Resource Server prüft das Access Token und führt gegebenenfalls die Anfrage aus.

(5) Ein Zugriff auf geschütze Ressourcen kann solange erfolgen, wie das Acces Tokens gültig ist. Wenn der Client feststellt, dass die Lebensdauer des Access Tokens überschritten ist, fährt er mit Schritt 7 fort.

(6) Der Resource Server antwortet dem Client mit dem HTTP-Statuscode 401 “Unauthorized”, wenn er ein ungültiges Access Token erhält.

(7) Der Client fordert ein neues Access Token mittels des Refresh Tokens beim Authorization Server an.

(8) Der Authorization Server stellt dem Client ein neues Access Token aus. Bei Bedarf erhält der Client ein zusätzliches Refresh Token.[61]

Die Tokens liegen bei OAuth 2 in Form von kodierten Zeichenketten vor, werden innerhalb der Antwortmeldung des Authorization Servers aber meistens mit einem bestimmten Media Type wie JavaScript Object Notation (JSON) ausgeliefert. In Listing 6 wird ein Access Token als JSON-Objekt in der HTTP-Response dargestellt.

Listing 6: Response-Objekt eines Access Tokens im Format JSON

```
HTTP/1.1 200 OK
Content-Type: application/json
...

{
    "access_token":"Access Token",
    "expires_in":3600,
    "state":"Sate",
    "refresh_token":"Refresh Token",
}
```

[61] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-1.5

Das Access Token repräsentiert eine Autorisierung im Namen des Resource Owners. Der zweite Parameter teilt den Clients mit, wie lange ein Access Token noch gültig ist. Die Angabe der Zeiteinheit erfolgt dabei in Sekunden. Der Parameter “state” wird seitens der Clients für Sicherheitsabfragen genutzt und ist oftmals auch eine kodierte Zeichenkette. Ein Client überträgt den Wert des Parameters bei seiner Anfrage an einen Authorization Server und überprüft nach dessen Antwort, ob der selbe Wert zurückgeliefert wird. Das Refresh Token ist ebenfalls eine kodierte Zeichenkette, mit der ein neues Access Token beim Authorization Server angefordert werden kann.[62]

Die meisten APIs, die auf dem Protokoll OAuth 2 basieren, fordern ein Access Token mit dem Schema Bearer Authentication der HTTP-Authentifizierung an. Mit einem Bearer Token kann ein Zugriff auf geschützte Ressourcen stattfinden. Bei einer Anfrage an einen Ressource Server mit einem Bearer Token sind keine weiteren Kenntnisse über kryptografischen Verfahren notwendig. Allerdings wird seitens der Spezifikation von OAuth 2 auch die Verwendung von MACs mit dem Schema MAC Access Authentication der HTTP-Authentifizierung unterstützt.[63]

3.2.5 Clients und Profile

Bevor der Protokollablauf von OAuth 2 gestartet werden kann, muss sich ein Client am Authorization Server registrieren. Die Art und Weise, wie sich ein Client registrieren soll, wird in der Spezifikation des Protokolls nicht genau definiert. Fest steht jedoch, dass die Registrierung ausserhalb des Flows bei einem Dienstanbieter vorzunehmen ist. Der gebräuchlichste Ansatz stellt das Ausfüllen eines Formulares innerhalb einer Webseite dar. Dabei muss ein Client allerdings nicht direkt mit dem Authorization Server kommunizieren, da die Formulare die Angabe von URIs, zum Beispiel für die Weiterleitung vom Authorization Server zum Client ermöglichen. Bei der Registrierung von Clients werden seitens der Spezifikation von OAuth 2 zwei verschiedene Arten von Clients unterschieden:

- confidential
- public

Confidential Clients sind fähig, die Credentials der Benutzer geheim zu halten. Sie laufen meist auf Webservern und persistieren die Credentials innerhalb von Datenbanken.

Public Clients sind nicht in der Lage die Credentials der Benutzer sicher zu verwalten. Sie basieren meist auf Skriptsprachen wie JavaScript, die innerhalb eines Webbrowsers ausgeführt werden oder es handelt sich um Anwendungen für mobile Geräte.

Die Identifikation der einzelnen Clients geschieht über eine eindeutige Zeichenkette, der Client ID. Sie wird seitens des Authorization Servers an die registrierten Clients vergeben. Confidential Clients erhalten zudem ein Passwort, das Client Secret. Es muss zusammen mit der Client ID bei der Authentifizierung am Authorization Server angegeben werden. Die Identifikation von public Clients wird

[62] Vgl. Spasovski, M. (2013), S. 17-18.
[63] Vgl. Boyd, R. (2012), S. 10.

hingegen nicht genau seitens der Spezifikation des Protokolls beschrieben, da diese Clients sich nicht gegenüber einem Authorization Server authentifizieren müssen.[64]

Clients, die im Besitz eines Passwortes sind, können sich über das Authorization-Headerfeld mit dem Schema Basic Autentication der HTTP-Authentifizierung gegenüber einem Authorization Server authentifizieren. Alternativ können Credentials auch als Parameter “client_id” und “client_secret” bei einer Anfrage im Datenteil der Nachricht mitgeschickt werden. Dies ist allerdings eher für den Fall angedacht, wenn das Schema Basic Authentication nicht vom Client verwendet werden kann.[65]

Die Clients werden bei OAuth 2 zusätzlich in drei verschiedene Profile eingeteilt:

- serverbasierte Webanwendungen
- Anwendungen auf Clientseite, die in einem Webbrowser laufen
- native Anwendungen

Serverbasierte Webanwendungen sind Clients, die auf einem Webserver laufen. Sie werden vom Resource Owner genutzt, um ihre Ressourcen von der API einer anderen Anwendung abzufragen. Der Benutzer hat während der Nutzung des Clients keine Einsicht auf das Client Secret oder auf die Access Tokens vom Authorization Server.

Anwendungen auf der Clientseite werden oft innerhalb eines Webbrowsers auf dem Gerät eines Resource Owners ausgeführt. Ein solcher Client wird von einem Webserver bezogen und mit Skriptsprachen ausgeführt. Credentials und Protokolldaten könnten bei der Verwendung dieses Profils mitgelesen werden, da der Quellcode der Skriptsprachen über den Webbrowser zugänglich ist.

Native Anwendungen sind Clients, die den Clients in einem Webbrowser ähneln, da auch hierbei Credentials und Protokolldaten eingesehen werden können. Allerdings werden diese Clients auf dem Gerät des Resource Owners installiert und haben somit keinen Zugriff auf die Ressourcen eines Webbrowsers.[66]

3.2.6 Endpunkte und Scopes

Ein Endpunkt wird bei OAuth 2 als URI angegeben. Endpunkte geben Auskunft darüber, welche Adresse eines Servers bei Anfragen genutzt werden muss. Das Protokoll OAuth 2 unterscheidet drei Endpunkte.

Die serverseitigen Endpunkte werden innerhalb des Quellcodes eines Clients angegeben:

- Authorization Endpoint
- Token Endpoint

[64] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-2
[65] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-1.3
[66] Vgl. Boyd, R. (2012), S. 9-10.

Der Authorization Endpoint wird seitens eines Clients genutzt, um eine Autorisierung vom Resource Owner zu erhalten. Wenn die Authentifizierung des Clients erfolgreich ausgeführt werden konnte, erhält der Client den Authorization Grant. Der Implicit Grant stellt dabei eine Besonderheit dar, denn bei diesem Authorization Grant erhält ein Client das Access Token direkt vom Authorization Server. Ein Client muss dem Authorization Endpoint über den Parameter “response_type” mitteilen, welchen Authorization Grant er als Antwort erwartet.[67]

Der Token Endpoint wird vom Client verwendet, um ein Access Token zu bekommen. Dazu muss ein Authorization Grant übermittelt werden.

Ein weiterer Endpunkt befindet sich auf der Clientseite:

- Redirection Endpoint (Callback)

Der Redirection Endpoint wird bei der Registrierung der Clients am Authorization Server angegeben und genutzt, um einem Client den Authorization Grant oder ein Access Token zu übergeben.

Ein Scope ist ein Parameter, der genutzt werden kann, wenn ein Client Anfragen an die beiden serverseitigen Endpunkte von OAuth 2 stellt. Mit dem Parameter “scope” kann ein Client angeben, auf welchen Teil der geschützten Ressourcen er im Namen des Resource Owners zugreifen möchte. Viele Dienstanbieter nutzen Scopes, um bei der Autorisierung der Clients dem Resource Owner eine Übersicht über Ressourcen zu geben, auf die mit einer Zustimmung zugegriffen werden kann. Der Scope ist meist ein optionaler Parameter und sein Wert ist abghängig von der gewünschten Funktionalität beim Zugriff auf geschütze Ressourcen.[68]

3.3 Abläufe des Protokolls

3.3.1 Authorization Code Flow

Der Authorization Code Flow ist für serverseitige Webanwendungen gedacht und kann genutzt werden, um ein Access Token oder ein Refresh Token zu beziehen. Dieser Flow nutzt einen Mechanismus, bei dem Weiterleitungen mit dem HTTP-Protokoll durchgeführt werden. Daher muss die als Client fungierende Webanwendung fähig sein, mit dem Webbrowser des Resource Owners zu interagieren und eingehende Anfragen des Authorization Servers zu verwalten. Bei diesem Ablauf wird der URI zur Weiterleitung des Clients vom Authorization Server zum Redirection Endpoint mit dem Parameter “redirect_uri” angegeben. Der Authorization Code Flow ist in Abbildung 7 dargestellt.[69]

[67] Vgl. Spasovski, M. (2013), S. 20.
[68] Vgl. Spasovski, M. (2013), S. 20.
[69] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-4.1

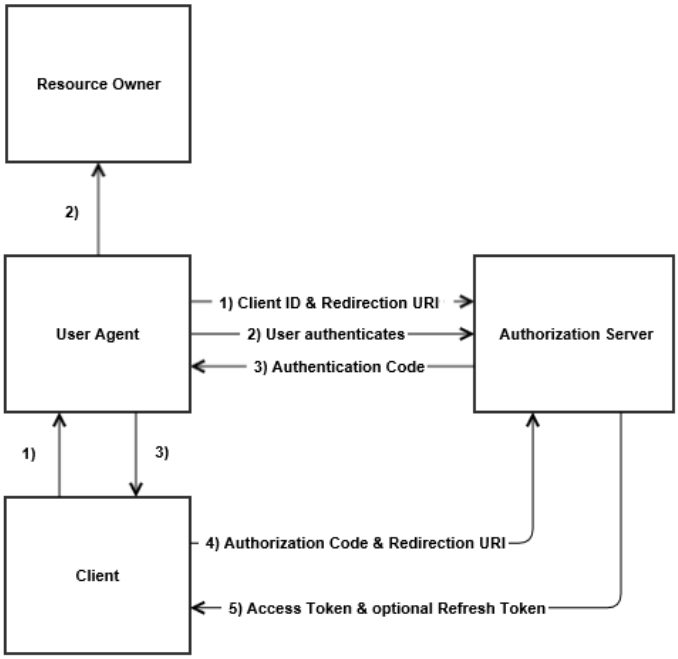

Abbildung 7: Authorization Code Flow
Quelle: Hardt, D. (2012), http://tools.ietf.org/html/rfc6749

Der Authorization Code Flow in Abbildung 7 beinhaltet folgende Schritte:

(1) Der Client initialisiert den Flow, indem er den Webbrowser des Resource Owners zum Authorization Endpunkt des Authorization Servers weiterleitet. Dabei werden die Parameter “response_type” “client_id”, “scope”, “state” und “redirect_uri” mit zum Authorization Server übertragen.

(2) Der Authorization Server authentifiziert den Resource Owner über den Webbrowser und fragt ab, ob der Resource Owner den Zugriff auf die geschützten Ressourcen autorisieren möchte.

(3) Der Authorization Server leitet danach den Webbrowser mittels der “redirect_uri” zurück zum Redirection Endpoint vom Client. Der URI beinhaltet den Authorzation Code und den Parameter “state”, den der Client im Vorfeld übermittelt hat.

(4) Der Client prüft den Wert des Parameters “state” und fragt anschließend mittels des Authorization Codes ein Access Token beim Token Endpoint des Authorization Servers an. Bei der Anfrage übergibt der Client dem Authorization Server den Wert des Parameters “redirect_uri”.

(5) Der Authorization Server authentifiziert den Client, überprüft den Authorization Code und stellt sicher, dass die “redirect_uri” mit dem URI aus Schritt 3 übereinstimmt. Im Anschluss übermittelt der Authorization Server dem Client die Tokens.[70]

[70] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-4.1

Für die Anfrage zur Autorisierung des Zugriffs auf eine geschützte Ressource konstruiert die Webanwendung zu Beginn einen URI. Der Client leitet den Webbrowser des Resource Owners mit dem erzeugten URI zum Authorization Server des Dienstes, der mit OAuth 2 abgesichert ist. Dies geschieht oft mittels des HTTP Statuscodes 303 “See Other”, der den Webbrowser dazu auffordert, eine Ressource an anderer Stelle zu suchen. Das folgende Listing 7 zeigt die Anfrage an den Authorization Endpoint des Authorization Servers unter Verwendung von HTTPS. Der Parameter “response_type” hat beim Authorization Code Flow den Wert “code”.

Listing 7: Anfrage zur Autorisierung an den Authorization Endpoint

```
GET /authorize?response_type=code
    &client_id="Client ID"
    &state="State"
    &redirect_uri="https://www.client.com/cb" HTTP/1.1
Host: www.authserver.com
```

Nachdem der Resource Owner den Zugriff autorisiert hat, fügt der Authorization Server den Authorization Code als Parameter “code” zuammen mit dem Parameter “state” dem URI aus dem Parameter “redirect_uri” hinzu und übergibt diesen angepassten URI dem Client über den Redirection Endpoint. Listing 8 zeigt eine mögliche Antwortmeldung des Authorization Servers mittels HTTPS. Der Redirection Endpoint wird dabei durch den Pfad “/cb” dargestellt.

Listing 8: Antwort des Authorization Servers mit dem Authorization Code

```
HTTP/1.1 302 Found
Location: https://www.client.com/cb?code="Authorization Code"
    &state="State"
```

Für die Anfrage eines Access Tokens, muss der Client den Authorization Code an den Token Endpoint des Authorization Servers mit der POST-Methode von HTTP übermitteln. Dieser Schritt wird bei der Verwendung einer Client Library für OAuth 2 oftmals verborgen. Der Client muss sich außerdem bei dem Authorization Server authentifizieren. Dies geschieht entweder mittels des Authorization-Headerfeldes im Zusammenhang mit dem Schema Basic Authentication, bei dem die Parameter “client_id” den Benutzernamen und “client_secret” das Passwort darstellen oder die Parameter werden im Datenteil der HTTP-Nachricht versendet. Dann wird der Media Type “x-www-form-urlencoded” verwendet, mit dem die Werte der Parameter kodiert werden. Der Client muss dem Authorization Server zusätzlich mitteilen, welcher Typ eines Authorization Grants verwendet werden soll. Dies geschieht mit dem Parameter “grant_type”, der beim Authorization Code Flow den Wert “authorization_code” haben muss. Listing 9 zeigt die entsprechende Anfrage mit dem Protokoll HTTP.

Listing 9: Anfrage an den Token Endpoint mit dem Authorization Code

```
POST /token HTTP/1.1
Host: www.authserver.com
Authorization: Basic Y2xpZW50X2lkOmNsaWVudF9zZWNyZXQ=
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code="Authorization Code"
      &redirect_uri=https://www.client.com/cb
```

3.3.2 Implicit Flow

Der Implicit Flow ist für Clientanwendungen gedacht, bei denen einige Teile der Anwendung in einem Webbrowser mit Skriptsprachen ausgeführt werden. Mit diesem Flow wird direkt ein Access Token vom Authorization Server zur Clientanwendung im Webbrowser geliefert, nachdem ein Resource Owner dem Zugriff auf seine Ressourcen zugestimmt hat. Die Nutzung von Refresh Tokens wird bei diesem Ablauf nicht unterstützt und es wird auch kein Authorization Code benötigt. Daher muss ein Client den gesamten Flow erneut ausführen, wenn ein Access Token veraltet ist. Dieser Flow nutzt ebenfalls einen Mechanismus, bei dem Weiterleitungen mit dem HTTP-Protokoll durchgeführt werden.[71] Der große Unterschied zum Authorization Code Flow ist der Umstand, das die Clientanwendung, nicht im Stande ist, das Access Token sicher zu verwalten, da es dem Benutzer im Webbrowser ersichtlich wird. Die Parameter für die Anfrage werden bei der Übergabe vom Authoriation Server an den Webbrowser als Fragment an den URI angefügt. Dadurch sind die Parameter zur weiteren Verarbeitung mit der Skriptsprache im Webbrowser geeignet.[72] Ein Fragment wird mit dem Zeichen "#" einem URI hinzugefügt.

Der Implicit Flow, der in Abbildung 8 ersichtlich ist, definiert folgende Schritte:

(1) Der Client initialisiert den Flow durch die Weiterleitung des Webbrowsers eines Resource Owners zum Authorization Endpoint des Authorization Servers. Dabei werden die Parameter "response_type" "client_id", "scope", "state" und "redirect_uri" mit zum Authorization Server übertragen.

(2) Der Authorization Server authentifiziert den Resource Owner über den Webbrowser und fragt ab, ob der Resource Owner den Zugriff auf die geschützten Ressourcen autorisieren möchte.

(3) Der Authorization Server leitet danach den Webbrowser mittels der "redirect_uri" zurück zum Redirection Endpoint des Clients. Das Fragment des URI beinhaltet dabei unter anderem das Access Token.

(4) Der Webbrowser leitet das Access Token weiter zur Clientanwendung.[73]

[71] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-4.2
[72] Vgl. Spasovski, M. (2013), S. 48.
[73] Vgl. Spasovski, M. (2013), S. 46-47.

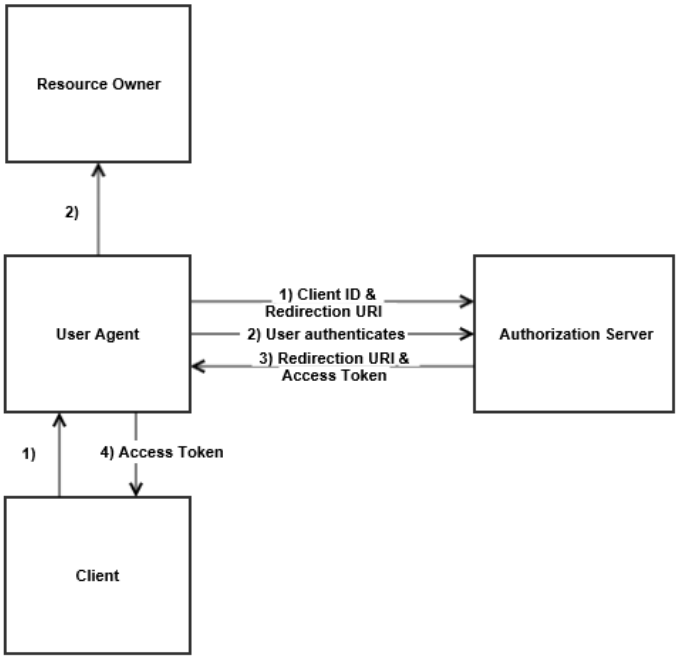

Abbildung 8: Implicit Flow
Quelle: Spasovski, M. (2013), S. 46.

Die Anfrage zur Autorisierung des Zugriffs auf eine geschützte Ressource wird ähnlich wie beim Authorization Code Flow mit einem URI durchgeführt, die der Client dem Webbrowser zu Anfang des Flows übergibt. Der Client leitet den Webbrowser des Resource Owners mit dem erzeugten URI zum Authorization Endpoint des Authorization Servers weiter. Der URI beinhaltet dabei die gleichen Parameter, wie beim Authorization Code Flow. Der Parameter “response_type” hat beim Implicit Flow allerdings den Wert “token”.

Nachdem der Resource Owner den Zugriff autorisiert hat, fügt der Authorization Server das Access Token als Parameter “access_token” dem Fragment des URI hinzu. Anschließend wird der URI an den Redirection Endpoint übergeben. Listing 10 zeigt den Aufbau des URI in einer mögliche Antwortmeldung des Authorization Servers mittels HTTPS. Der Redirection Endpoint wird dabei durch den Pfad “/cb” dargestellt.

Listing 10: URI mit Fragment bei der Antwort des Authorization Servers

```
https://www.client.com/cb#access_token="Access Token"
    &state="State"&expires_in=3600
```

3.3.3 Resource Owner Passwort Credentials Flow

Beim Resource Owner Passwort Credentials Flow werden die Credentials eines Benutzers in ein Access Token eingetauscht. Optional kann auch ein Refresh Token bezogen werden. Dieser Flow führt dazu, dass ein Client in den Besitz von Benutzername und Passwort kommt, weshalb er nur bei Anwendungen genutzt werden sollte, die einem abgesicherten Dienst selbst angehören. Eine mobile Anwendung von Facebook könnte beispielsweise mit diesem Flow den Zugang zur Facebook-Plattform gewähren. Abbildung 9 zeigt den Resource Owner Passwort Cedentials Flow. [74]

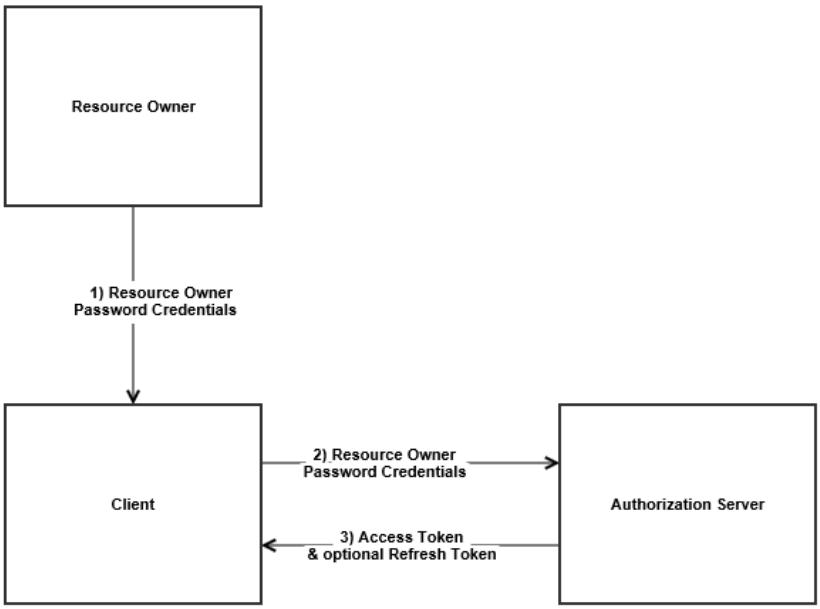

Abbildung 9: Resource Owner Password Credentials Flow
Quelle: Hardt, D. (2012), http://tools.ietf.org/html/rfc6749

Der Resource Owner Passwort Credentials Flow beinhaltet folgende Schritte:

(1) Der Resource Owner übergibt dem Client den Benutzernamen und das Passwort.

(2) Der Client fragt mit den Credentials ein Access Token beim Token Endpoint des Authorization Servers an und authentifiziert sich mit dem Authorization Server.

(3) Der Authorization Server prüft die Credentials und teilt das Access Token aus.[75]

Die Methode, mit der der Client an die Credentials des Resource Owners gelangt, wird nicht genau seitens der Spezifikation von OAuth 2 beschrieben. Der Client sollte jedoch die Credentials verwerfen, sobald er an ein Access Token gelangt.

[74] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-4.3
[75] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-4.3

Zur Erlangung des Access Tokens stellt der Client eine Anfrage an den Token Endpoint des Authorization Servers. Dabei wird der Parameter “grant_type” mit dem Wert “password” versehen. Die Credentials werden innerhalb des Datenteils einer Nachricht eingefügt. Listing 11 zeigt die entsprechende Anfrage an den Token Endpoint.

Listing 11: Anfrage an den Token Endpoint mit den Credentials des Resource Owners

```
POST /token HTTP/1.1
Host: www.authserver.com
Authorization: Basic Y2xpZW50X2lkOmNsaWVudF9zZWNyZXQ=
Content-Type: application/x-www-form-urlencoded

grant_type=password&username="Benutzername:Passwort"
```

3.3.4 Client Credentials Flow

Die bisher beschriebenen Abläufe sind dazu gedacht, Zugriffe von Clientanwendungen auf geschützte Ressourcen im Namen eines Resource Owners zu autorisieren. Es gibt jedoch auch Fälle, bei denen die Ressourcen dem Client selbst angehören und der Client nur die Funktionalitäten eines anderen Dienstes benutzen möchte. Für diese Zwecke gibt es bei OAuth 2 den Client Credentials Flow. Damit kann ein Client direkt ein Token mit seinen Credentials anfordern und nicht mit den Credentials des Resource Owners. Der Client Crentials Flow ist in Abbildung 10 ersichtlich.[76]

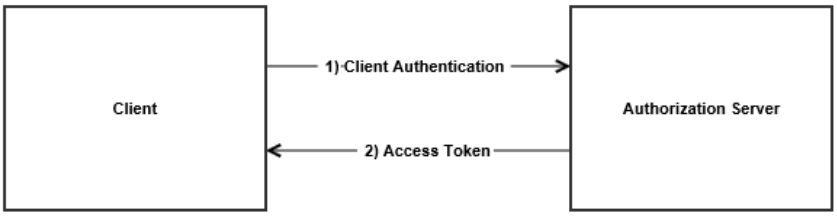

Abbildung 10: Client Credentials Flow
Quelle: Hardt, D. (2012), http://tools.ietf.org/html/rfc6749

[76] Vgl. Boyd, R. (2012), S. 41.

Der Client Credentials Flow beinhaltet die folgenden beiden Schritte:

(1) Der Client authentifiziert sich am Authorization Server und fragt ein Access Token vom Token Endpoint an.

(2) Der Authorization Server übergibt dem Client das Access Token.[77]

Zur Erlangung des Access Tokens stellt der Client eine Anfrage an den Token Endpoint des Authorization Servers. Der Parameter “grant_type” wird beim Client Credentials Flow mit dem Wert “client_credentials” versehen. Die Anfrage an den Token Endpoint wird in Listing 12 gezeigt:

Listing 12: Anfrage an den Token Endpoint mit den Credentials des Clients

```
POST /token HTTP/1.1
Host: www.authserver.com
Authorization: Basic Y2xpZW50X2lkOmNsaWVudF9zZWNyZXQ=
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

3.4 Zugriff auf Ressourcen

Das Erlangen eines Access Tokens und der Zugriff auf eine Ressource werden in der Spezifikation von OAuth 2 getrennt behandelt. Die Kernspezifikation wird in der RFC 6749 definiert. Darin ist die Kommunikation zwischen der Clientanwendung und dem Authorization Server beschrieben. Sie behandelt die Interaktion mit dem Resource Server nur am Rande. Innerhalb der RFC 6750 wird der Zugriff auf die geschützten Ressourcen eines Resource Servers mit Access Tokens und unter Verwendung des Schemas Bearer Authentication beschrieben.[78] Zusätzlich ist die Verwendung von OAuth 2 mit dem Schema MAC Authentication möglich. Damit kann eine Prüfsumme anstelle der kodierten Credentials übertragen werden.[79] Die meisten APIs, bei denen eine Autorisierung mit OAuth 2 umgesetzt wurde, nutzen jedoch das Schema Bearer Authentication.[80]

Alle Flows des Protokolls haben gemeinsam, dass sie von einem Client genutzt werden, um an ein Access Token zu gelangen und damit Anfragen auf geschützte Ressourcen im Namen eines Benutzers stellen zu können. Ein Client erhält den Zugriff auf solche Ressourcen, indem er das Access Token dem Resource Server präsentiert. Der Resource Server muss das Token validieren und damit sicherstellen, dass dieses gültig ist und der Scope zur angeforderten Ressource passt. Die Methoden mit denen ein Resource Server das Access Token prüft, sind seitens der Spezifikation nicht vorge-

[77] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-4.4
[78] Vgl. Mills, W. / Showalter, T. / Tschofenig, H. (2013), http://tools.ietf.org/html/draft-ietf-kitten-sasl-oauth-12#section-1
[79] Vgl. Hammer-Lahav, E. / Barth, A. / Adida B. (2011), http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-00#section-1
[80] Vgl. Boyd, R. (2012), S. 10.

schrieben, allerdings wird dabei eine Kommunikation zwischen Resource Server und Authorization Server benötigt.[81]

Nachdem ein Client ein Access Token bezogen hat, kann dieses auf mehreren Wegen zum Resource Server übertragen werden. Die bevorzugte Methode stellt bei OAuth 2 jedoch die Verwendung des Authorization-Headerfeldes dar. Listing 13 zeigt die Anfrage auf eine geschützte Ressource mit einem Access Token und dem Schema Bearer Authentication.

Listing 13: Anfrage auf eine geschützte Ressource mit einem Access Token

```
GET /resource/user/info HTTP/1.1
Host: www.resourceserver.com
Authorization: Bearer "Access Token"
...
```

Das Authorization-Headerfeld bietet unter anderem den Vorteil, dass es bei Anfragen des Clients nicht im Zwischenspeicher eines Browsers abgelegt wird. In der Spezifikation des Protokolls sind aber noch zwei weitere Varianten enthalten, um das Access Token übertragen zu können. Einerseits kann das Token als Parameter im Datenteil einer Nachricht mit dem Media Type “x-www-form-urlencoded” übertragen werden. Es kann aber auch als angefügter Parameter eines URI übertragen werden. Beide Varianten stellen eine Ausweichmöglichkeit für den Fall dar, dass ein Client das Authorization-Headerfeld nicht verändern kann.[82]

3.5 Verwendungszwecke

Das Protkoll OAuth 2 definiert mit seinen unterschiedlichen Abläufen einige Wege zum Erlangen eines Access Tokens. Die Wahl welcher Client sich für welchen Flow eignet, hängt von einigen Faktoren und dem Aufbau des Clients ab.

Der vielseitigste Ablauf ist der Authorization Code Flow. Der Flow sollte vor allem dann genutzt werden, wenn es sich bei einem Client um eine serverseitige Webanwendung handelt, die im Namen des Benutzers auf geschützte Ressourcen einer API zugreifen möchte. Diese Anwendungen sind in der Lage die Credentials des Ressource Owners so zu verwalten, dass sie vor Fremdzugriffen geschützt werden und nicht eingesehen werden können. Eine Webanwendung könnte beispielsweise auf die Benutzerdaten eines anderen Dienstes zugreifen wollen, um dadurch an den Namen der Benutzer zu gelangen. Damit müsste die Webanwendung keine Passwörter verwalten, sondern könnte stattdessen die Benutzer den Access Tokens zuordnen. Zum Zugriff auf die Ressourcen muss der Client eine HTTP-Weiterleitung zum Authorization Server initiieren und der Authorization Server authentifiziert den Benutzer. Wie der Authorization Server die Authentifizierung vornehmen muss, ist dabei nicht

[81] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-7
[82] Vgl. Boyd, R. (2012), S. 10.

vorgeschrieben und deshalb flexibel gehalten. Der Authorization Code Flow nutzt Access Tokens und Refresh Tokens, daher kann der API-Zugriff für Anwendungen für einen längeren Zeitraum ausgestellt werden.

Anwendungen auf der Clientseite, die in einem Webbrowser mit einer Sktiptrsprache wie JavaScript oder Flash ausgeführt werden, können keine Daten vertraulich behandeln. Sie besitzen zudem kein Client Secret, weshalb sie nicht am Authorization Server authentifiziert werden können. Diese Clients haben zwar eine Client ID, mit der sie identifiziert werden können, aber ohne die Abfrage eines Passwortes kann nicht sichergestellt werden, ob die Client ID auch dem richtigen Client angehört.[83] Der Implicit Flow eignet sich für diese Anwendungen, da auch hierbei HTTP-Weiterletungen genutzt werden können und der Ablauf eine vereinfachte Variante des Authorization Code Flows darstellt. Nach der Übergabe der Credentials des Resource Owners an den Authorization Server, wird direkt ein Access Token seitens des Clients bezogen. Die Anzahl der Anfragen an den Authorization Server reduziert sich und der Client benötigt kein Client Secret, da er vom Benutzer authentifiziert wird. Da der Implicit Flow keine Refresh Tokens behandelt, dient der Ablauf für Anwendungen, die kurzfristigen Zugriff auf eine API benötigen.

Die Abläufe Authorization Code Flow und Implicit Flow können auch bei mobilen- und Desktop-Anwendungen genutzt werden, um ein Access Token für API-Zugriffe zu erhalten. Derartige Clients werden für gewöhnlich nicht innerhalb eines Webbrowsers ausgeführt, weshalb entweder der auf dem Gerät installierte Webbrowser verwendet oder ein Webbrowser in die Clientanwendung integriert werden muss. Die Weiterleitung zum Client gestaltet sich bei beiden Varianten in veränderter Form. Für den ersten Fall müsste ein eigenes URI-Schema verwendet werden, bei dem die Adresse nicht mit der Bezeichnung des Kommunikationsprotokolls beginnt, sondern in der Form “myapp://oauth/cb”. Im zweiten Fall müsste als Ziel ein lokaler URI angegeben werden.[84] Die Wahl des geeigneten Flows für diese Anwendungen hängt davon ab, ob die Credentials des Resource Owners sicher verwaltet werden können.[85]

Bei Anwendungen, die als vertrauenswürdig eingestuft werden können, kann der Resource Owner Password Credentials Flow benutzt werden. Hierbei werden dem Client direkt die Credentials des Resource Owners übertragen und im Anschluss kann dieser damit ein Access Token beim Authorization Server beantragen. Die Anzahl der Anfragen an den Authorization Server wird mit diesem Ablauf reduziert, allerdings bekommt der Client die Einsicht in die Credentials. Der Ablauf eignet sich vor allem dann, wenn die Anwendung und der Resource Server dem selben Dienstanbieter angehören.

Der Client Credentials Flow kann genutzt werden, wenn eine Anwendung selbst den Zugriff auf eine API durchführen möchte und keine Autorisierung des Resource Owners benötigt wird.

[83] Vgl. Hiller, J. / Lodderstedt, T. (2012), S.127-130.
[84] Vgl. Hiller, J. / Lodderstedt, T. (2012), S.127-130.
[85] Vgl. Spasovski, M. (2013), S. 55.

3.6 Vorteile gegenüber der HTTP-Authentifizierung

Bei den Schemata Basic Authentication und Digest Authentication der HTTP-Autentifizierung müssen Benutzer ihre Credentials gegenüber Drittanwendungen preisgeben, wenn Sie damit auf geschützte Ressourcen einer API zugreifen möchten. Es entsteht dadurch ein Sicherheitsproblem, weil meist unklar ist, wie die Credentials von diesen Anwendungen weiterverarbeitet werden.[86] Ändert ein Benutzer seine Credentials für den Zugriff auf die jeweilige API, haben die Anwendungen zudem keinen Zugriff mehr aur die dort hinterlegten Ressourcen.

Bei der Verwendung von OAuth 2 können für jede Anwendung eigene Credentials vergeben werden, weshalb es eine flexiblere Handhabung bei der Autorisierung von API-Zugriffen ermöglicht. Bei der Registrierung am Authorization Server können die Rechte für jede Clientanwendung getrennt verwaltet werden und Änderungen von Credentials führen nicht mehr dazu, dass die Anwendungen den API-Zugriff verlieren. Zusätzlich erhält ein Resource Owner vor der Autorisierung der API-Zugriffe für jede Anwendung eine Übersicht über die angeforderte Ressourcen.

3.7 Sicherheitsbetrachtung

Die Trennung der Erlangung eines Access Tokens von dem Zugriff auf Ressourcen und die verschiedenen Abläufe des Protokolls OAuth 2 führen zu einer Vielzahl von unterschiedlichen Implementationen des Authorization Servers und der Clients. Dadurch steigen auch die Anforderungen an die Sicherheit beim Austausch von Credentials und Tokens über das Netz. Weiterführende Sicherheitsüberlegung bei der Verwendung von OAuth 2 wurden getrennt von der Kernspezifikation in einem weiteren Dokument festgehalten. Die RFC 6819 dient dazu, die Anwender über mögliche Sicherheitsmaßnahmen zu informieren.

Während der Ausführung eines Ablaufs des Protokolls werden einige Parameter zwischen den beteiligten Rollen mittels Anfrage- und Antwortnachrichten ausgetauscht. Wenn diese Informationen unzureichend geschützt werden, entstehen einige Sicherheitsrisiken:

- Mittels Client ID und Client Secret kann die Identität einer Anwendung angenommen werden.
- Ein Access Token kann genutzt werden, um auf geschützte Ressourcen im Namen eines Benutzers zugreifen und Operationen mit Methoden darauf ausführen zu können.
- Das Refresh Token kann zusammen mit den Client Credentials verwendet werden, um ein neues Access Token beziehen zu können.
- Mit den Credentials des Resource Owners kann ein Angreifer auf mehrere Ressourcen zugreifen und dadurch vielfältigen Schaden verursachen.[87]

[86] Vgl. Makice, K. (2009), S. 142.
[87] Vgl. Spasovski, M. (2013), S. 78.

Das Protokoll OAuth 2 stellt jedoch einige Funktionen zur Verfügung, um die Sicherheit zu erhöhen und Angriffen vorzubeugen:

- Scopes können definiert werden.
- Access Tokens können mit einer Lebensdauer behaftet sein.
- Refresh Tokens können in eingen Abläufen genutzt werden.
- Authorization Codes können benutzt werden.
- Der Parameter “state” kann genutzt werden.

Die Nutzung eines Scopes kann bei jedem der Abläufe dazu führen, dass nur auf bestimmte Ressourcen mit ausgewählten Methoden seitens eines Clients zugegriffen werden kann. Der Client bestimmt mit dem Scope welche Art von Zugriff er auf bestimmten Ressourcen ausführen möchte. Der Authorization Server stellt dem Client nach der Zustimmung des Resource Owners ein Access Token aus, welches die Angaben im Scope berücksichtigt. Dadurch lässt sich der Geltungsbereich eines Access Tokens bestimmen und es können die Methode zum Zugriff auf die Ressourcen festgelegt werden. Zusätzlich können die Scopes der Anwendungen auch im Nachhinein weiter eingeschränkt werden.

Der Parameter “expires_in” wird vom Authorization Server genutzt, um einem Access Token eine Lebensdauer zuordnen zu können. Wenn das Access Token an einen Client übergeben wird, fügt der Authorization Server den Wert des Parameters in seiner Antwortmeldung hinzu. Die Lebensdauer sollte dabei möglichst gering ausfallen, damit das Acess Token bei einem Missbrauch nicht benutzt werden kann. Je unsicherer die Behandlung der Tokens in einem Client gewährleistet ist, desto kleiner sollte deren Lebensdauer ausfallen.

Mittels Refresh Tokens wird die Häufigkeit des Versendens von Credentials zwischen einem Client und dem Authorization Server reduziert. Dadurch ergeben sich weniger Angriffsmöglichkeiten, um an diese Informationen gelangen zu können und das Nachrichtenaufkommen wird verringert.

Authorization Codes werden beim Authorization Code Flow verwendet. Anstatt ein Access Token nach der Authentifizierung eines Resource Owners zu versenden, erhält ein Client zunächst einen Authorization Code, mit dem er im zweiten Schritt ein Access Token anfordern kann. Der Authroization Code ist nur für eine kurze Zeit gültig und führt dazu, dass auch ein Client beim Authorization Server authentifiziert wird, wenn er ein Access Token anfordert. Dadurch erhält der Authroization Server zusätzliche Informationen über einen Client.[88]

Der Parameter “state” wird beim Authorization Code Flow und beim Implicit Flow vom Client erzeugt und bei der Antwort eines Authorization Servers zur Überprüfung an den Client zurückgegeben. Damit können vorrangig Cross-Site-Request-Forgery (CSRF)-Attacken abgewehrt werden. In Verbindung mit dem Protokoll OAuth 2 kann ein Angreifer bei einer CSRF-Attacke den Redirection Endpoint eines Clients so manipulieren, dass dieser andere Authorization Codes oder Tokens entgegennimmt. Der Angreifer kann dadurch Informationen aus der Clientanwendungen erhalten, wenn ein Zugriff auf Ressourcen stattfinden soll, da in diesem Fall Ressourcen angesprochen werden, die ihm

[88] Vgl. Spasovski, M. (2013), S. 78-80.

selbst gehören. Zum Beispiel könnten damit Informationen zu Bankdaten versehentlich übertragen werden.[89] Die URIs zur Weiterleitung können allerdings bei der Registrierung der Clients am Authorization Server angegeben werden. Dadurch ist es möglich, dass ein Authorization Server vor der HTTP-Weiterleitung den entsprechenden URI überprüft.[90]

Da es sich bei OAuth 2 um eine Spezifikation eines Protokolls handelt, hängt die Sicherheit bedeutend von der Implementierung ab. Die Spezifikation schreibt die Benutzung von Bearer Tokens vor, die den Zugriff auf geschützte Ressourcen autorisieren. Sie müssen bei jeder Anfrage auf die Ressourcen übertragen werden und liegen nicht in verschlüsselter Form vor, weshalb eine gesicherte Verbindung mit TLS erforderlich ist. Obwohl es als ausreichend sicher gilt, steht TLS unter Kritik. Viele Browser unterstützen bislang nicht die aktuelle Version und es traten in der Vergangenheit öfter Sicherheitslücken auf.[91] Kurz vor der Veröffentlichung der Spezifikation von OAuth 2, gab der Autor Eran Hammer-Lahav seinen Rücktritt aus dem OAuth-Team bekannt. Als eine der Ursachen nennt er die geringere Sicherheit gegenüber der Vorgängerversion des Protokolls auf Grund des Wegfalls von verschlüsselten Anfragen mit Signaturen. Neben der Kritik äußerte er jedoch auch, dass das Protokoll OAuth 2 mit ausreichenden Sicherheitsüberlegungen auch zu einer sicheren Implementierung führen kann.[92]

3.8 Schwachstellen und Fehler

Die erste Sicherheitslücke von OAuth 2 wurde im Frühjahr 2014 von einem Studenten der Universität Nanyang in Singapur entdeckt. Die Sicherheitslücke hört auf den Namen Covert Redirect und soll Angreifern das Ausspähen von Ressourcen ermöglichen. Ein Angreifer kann demnach mit einem veränderten URI für die Weiterleiung an den Client einen Auhtorization Server dazu bringen, die Antwortmeldungen an eine beliebige Internetseite zu versenden.[93]

Da OAuth 2 bei vielen großen Plattformen eingesetzt wird, kann ein Angreifer so an eine Vielzahl von personenbezogenen Daten gelangen. Die URIs sollten daher bereits während der Registrierung von Clientanwendungen vom Anbieter abgefragt werden, damit sie vor einer möglichen Weiterleitung validiert werden können.[94]

[89] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-10
[90] Vgl. Spasovski, M. (2013), S. 80-81.
[91] Vgl. Hammer-Lahav, E. (2010), http://hueniverse.com/2010/09/15/oauth-2-0-without-signatures-is-bad-for-the-web/
[92] Vgl. Hammer-Lahav, E. (2012), http://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/
[93] Vgl. Hammer-Lahav, E. (2014), http://oauth.net/advisories/2014-1-covert-redirect/
[94] Vgl. Hardt, D. (2012), http://tools.ietf.org/html/rfc6749#section-10.6

4 Zugriff auf Ressourcen der Dropbox API

Die Plattform Dropbox bietet den Zugriff auf die hinterlegten Ressourcen der Benutzer mit dem Protokoll OAuth 2 an. Dabei können die Abläufe Authorization Code Flow oder Implicit Flow verwendet werden. Im Folgenden soll der Zugriff auf die Benutzerdaten von Dropbox mit dem Authorization Code Flow erstellt werden. Dropbox schreibt, bei der Benutzung der API mit diesem Ablauf folgende Schritte vor:

(1) Autorisierung des gewünschten Zugriffs auf eine Ressource und der Erhalt des Authorization Codes
(2) Umtausch des Authorization Codes in ein Access Token
(3) Zugriff auf die Ressourcen mit dem Access Token

In Schritt 1 muss der Ressource Owner mit der HTTP-Weiterleitung zum Authorization Endpoint von Dropbox weitergeleitet werden. Der Aufbau eines möglichen URI für die Anfrage an den Authorization Server von Dropbox ist in Listing 14 ersichtlich.

Listing 14: URI für die Anfrage an den Authorization Endpoint von Dropbox

```
https://www.dropbox.com/1/oauth2/authorize?client_id="Client ID"
    &response_type=code
    &redirect_uri="Redirection URI"
    &state="Base64-Zeichenkette"
```

Die Verwendung des Parameters “state” wird seitens Dropbox empfohlen, um CSRF-Attacken zu verhindern. Bei der Verwendung des offiziellen Software Developement Kit (SDK) von Dropbox ist er eine Zeichenkette, die mit dem Verfahren Base64 erzeugt wurde. Nachdem der Ressource Owner sich an dem Authorization Server authentifiziert hat, wird er mittels eines URI und weiteren Parametern zum Client weitergeleitet. Im Anschluss muss der Client den Parameter “state” überprüfen und den Authorization Code aus dem URI extrahieren. Listing 15 zeigt einen URI zur Weiterleitung an den Client bei Dropbox:

Listing 15: URI zur Weiterleitung an den Client bei Dropbox

```
https://www.client.com/cb?code="Authorization Code"
    &state="Base64-Zeichenkette"
```

Im Schritt 2 soll der Authorization Code in ein Access Token umgewandelt werden. Dazu muss der Client dem Token Endpoint des Authorization Servers den Authorization Code liefern. Als Antwort erhält der Client im Erfolgsfall ein Access Token, mit dem der Zugriff auf Ressourcen von Dropbox durchgeführt werden kann. Listing 16 zeigt die Anfrage an den Token Endpoint von Dropbox mit dem Kommandozeilentool cURL.

Listing 16: Anfrage an den Token Endpoint von Dropbox

```
curl https://api.dropbox.com/1/oauth2/token
    -d code="Authorization Code"
    -d grant_type=authorization_code
    -d redirect_uri="Redirection URI"
    -u "Client ID":"Client Secret"
```

Der Zugriff auf die Ressourcen der API von Dropbox erfolgt im Schritt 3 unter Verwendung des Schemas Bearer Authentication. Dabei muss das Authorization-Headerfeld mit dem Access Token bei der Anfrage hinzugefügt werden. Der Header kann auch bei einer Anfrage mit dem Kommandozeilentool cURL verwendet werden und ist in Listing 17 dargestellt.

Listing 17: Anfrage an den Resource Server von Dropbox

```
curl https://api.dropbox.com/1/account/info
    -H "Authorization: Bearer "Access Token"
```

Eine Webanwendung muss sich im Vorfeld bei dem Authorization Server des Anbieters von OAuth 2 registrieren. Bei der Registrierung der Clients werden einer Anwendung die Client Credentials zugeteilt. Außerdem muss ein URI für die Weiterleitung angegeben werden. Der Dialog zur Registrierung der Clientanwendungen von Dropbox ist in Abbildung 11 ersichtlich.

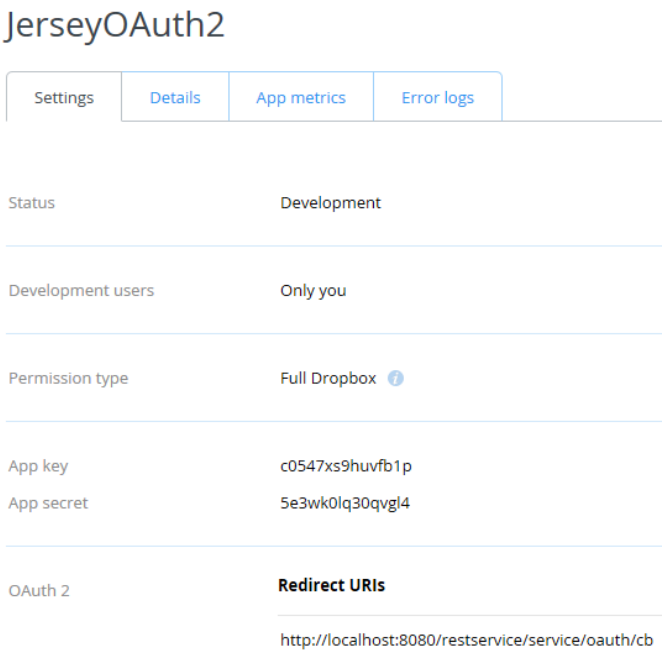

Abbildung 11: Client-Registrierung von Dropbox

Die Webanwendung, welche im Authorization Code Flow von OAuth 2 als Client agiert, wird in der Programmiersprache Java mit der Spezifikation der Softwarearchitektur Java Plattform Enterprise Edition (Java EE) 7 innerhalb der Entwicklungsumgebung Netbeans erstellt. Die Referenzimplementierung für diesen Standard ist der Glassfish-Applikationsserver in der Version 4. Das Open-Source-Projekt Jersey dient zusätzlich als Referenzimplementierung für Internetdienste mit dem Architekturstil REST, welche in der Spezifikation Java API for RESTful Web Services (JAX-RS) 2.0 definiert sind. Das Projekt Jersey ist in der Java EE 7 enthalten. Mittels Jersey können Annotationen genutzt werden, welche die Enwticklung der Dienste und der dazugehörigen Endpunkte vereinfacht. Mit den Annotationen können beispielsweise URIs und HTTP-Methoden auf Java-Klassen gemappt werden.

Damit eine Webanwendung mit Jersey erstellt werden kann, bedarf es zunächst einer Konfiguration für die Behandlung des Einstiegspunktes in die Anwendung. Eine Webanwendung kann dazu die Java-Klasse Application erweitern, wie es in Lisiting 18 ersichtlich ist.

Listing 18: Konfiguration eines Webservices nach JAX-RS 2.0

```
1 @ApplicationPath("/service")
2 public class RESTConfiguration extends Application {}
```

Damit die Funktionalität von OAuth 2 innerhalb von Jersey genutzt werden kann, bedarf es zusätzlich der Bekanntgabe von Abhängigkeiten zu der API von Jersey. Diese werden mit dem Build-Management-Tool Maven in der Konfigurationsdatei “pom.xml” angegeben. Listing 19 zeigt den benötigten Eintrag.

Listing 19: Bekanntgabe von Abhängigkeiten in der pom.xml

```
<dependency>
    <groupId>org.glassfish.jersey.security</groupId>
    <artifactId>oauth2-client</artifactId>
    <version>2.11</version>
    <type>jar</type>
</dependency>
```

Die Webanwendung benötigt nun noch Ressourcen, welche in eigenen Java-Klassen erstellt werden müssen. Als Beispiel soll eine Ressource dienen, die nach einer Anfrage mit der GET-Methode die Zeichenkette “Willkommen” zusammen mit dem jeweiligen Benutzernamen zurückliefert. Der Benutzername wird zu diesem Zweck aus den Benutzerdaten von Dropbox herangezogen. Für den Zugriff auf die API von Dropbox wird daher ein Access Token benötigt. Der erste Teil der Ressourcen-Klasse der Webanwendung ist in Listing 20 dargestellt.

Listing 20: Erster Teil der WelomeResource.java

```
1  @Path("/")
2  public class WelcomeResource{
3      @GET
4      @Path("hello")
5      @Produces("text/plain")
6      public Response getHello() {
7          if (OAuth2Controller.getAccessToken() == null){
8              OAuth2Controller. setClientIdentifier ();
9              final String REDIRECTURI = "http://localhost:8080/restservice/service/oauth/cb";
10
11             OAuth2CodeGrantFlow flow = OAuth2ClientSupport.authorizationCodeGrantFlowBuilder(
12                 OAuth2Controller. getClientIdentifier (),
13                 "https :// www.dropbox.com/1/oauth2/authorize",
14                 "https :// api.dropbox.com/1/oauth2/token")
15                 . redirectUri (REDIRECTURI)
16                 .build ();
17
18             OAuth2Controller.setAuthFlow(flow);
19
20             String authuri = flow. start ();
21
22             return Response.seeOther(UriBuilder.fromUri(authuri).build()). build ();
23         }
24         ...
25     }
26 }
```

Die Ressource "WelcomeResource" ist über das in Listing 21 angezeigte cURL-Kommando erreichbar.

Listing 21: cURL-Kommando für den Zugriff auf die Ressource WelcomeResource

```
curl -X GET http://localhost:8080/restservice/service/hello
```

Wenn ein Zugriff auf diese Ressource stattgefunden hat, sendet der Webbrowser eines Benutzers eine HTTP-Anfrage an die Webanwendung. Diese wird in Listing 22 dargestellt.

Listing 22: Anfrage an die Webanwendung

```
GET service/hello HTTP/1.1
Host: localhost:8080
```

Da kein Access Token während der Anfrage zur Verfügung steht, wird der Webbrowser von der Webanwendung zum Authorization Server von Dropbox weitergeleitet. Die Weiterleitung mit dem Statuscode 303 ist in Listing 23 ersichtlich.

Listing 23: Antwort der Webanwendung

```
HTTP/1.1 303 See Other
Location: https://www.dropbox.com/1/oauth2/authorize?response_type=code
    &redirect_uri=http://localhost:8080/restservice/service/oauth/cb
    &state=e10e40cb-d764-429c-802a-e3f95dda3b64&client_id=c0547xs9huvfb1p
Host: www.dropbox.com
```

Anschließend folgt der Webbrowser dem URI im Location-Header, welche für die Anfrage benötigte Parameter beinhaltet. Listing 24 zeigt die Anfrage an den Authorization Server von Dropbox.

Listing 24: Anfrage an den Authorization Server von Dropbox

```
GET https://www.dropbox.com/1/oauth2/authorize?response_type=code
    &redirect_uri=http://localhost:8080/restservice/service/oauth/cb
    &state=e10e40cb-d764-429c-802a-e3f95dda3b64&client_id=c0547xs9huvfb1p
    HTTP/1.1
Host: www.dropbox.com
```

Dropbox antwortet darauf mit dem Statuscode 200 und versendet im Datenteil der Antwort eine Internetseite für den Login des Benutzers. Die Antwort des Authorization Servers von Dropbox ist in Listing 25 dargestellt.

Listing 25: Antwort des Authorization Servers von Dropbox

```
HTTP/1.1 200 OK
Content-Type: text/html
Host: www.dropbox.com
```

Der Benutzer wird anschließend zur Eingabe seiner Credentials aufgefordert. In dem angezeigten Authentifizierungsdialog wird zudem der Name der Anwendung, welche auf die Ressourcen von Dropbox zugreifen möchte, angezeigt. Abbildung 12 zeigt den Authentifizierungsdialog von Dropbox.

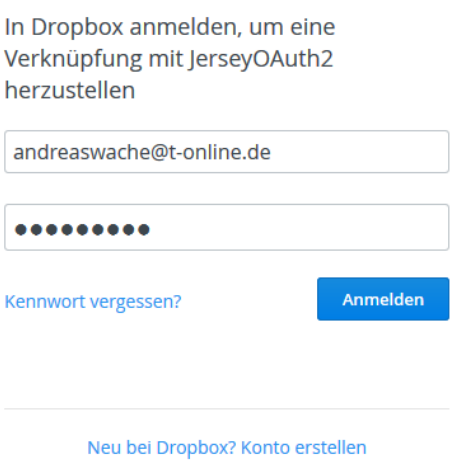

Abbildung 12: Authentifizierungsdialog von Dropbox

Nachdem sich der Benutzer erfolgreich angemeldet hat und somit beim Authorzation Server authentifizert ist, wird ihm ein Dialog für die Autorisierung der Zugriffe auf die Ressourcen von Dropbox angezeigt. Da in diesem Beispiel kein Scope verwendet wurde, ist in dem Dialog nicht ersichtlich auf welche Ressourcen der Zugriff stattfinden soll. Abbildung 13 zeigt den Autorisierungsdialog von Dropbox.

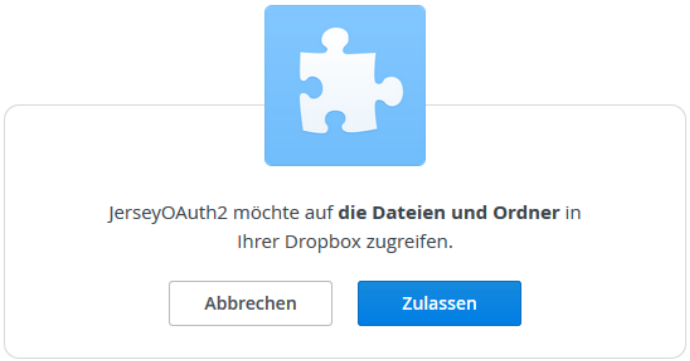

Abbildung 13: Autorisierungsdialog von Dropbox

Wenn der Benutzer seine Zustimmung für die angeforderten Zugriffe auf Ressourcen gegeben hat, wird er vom Authorization Server mittels des URI über den Webbrowser zum Redirection Endpoint der Webanwendung geleitet. Der Webbrowser erhält dazu einen Location-Header in einer Antwort von Dropbox. Die Antwort kommt dadurch zu Stande, dass die Credentials des Benutzers vorab mit der POST-Methode vom Webbrowser an den Authorization Server übergeben werden. Listing 26 zeigt die Antwort des Authorization Servers von Dropbox, in der auch der Authorization Code enthalten ist.

Listing 26: Antwort des Authorization Servers von Dropbox mit dem Authorization Code

```
HTTP/1.1 302 FOUND
Content-Type: text/html
Location: http://localhost:8080/restservice/service/oauth/cb?
    state=e10e40cb-d764-429c-802ae3f95dda3b64
    &code=3nIMkXJBjAUAAAAAAAACkcQNTfOC4aTGdCJQg73aLW0
Host: www.dropbox.com
```

Der Webbrowser lädt anschließend die Seite neu, um dem URI aus dem Location-Header zu folgen. Nun erhält die Webanwendung den Authorization Code und kann diesen im weiteren Verlauf der Kommunikation mit dem Authotization Server als ein Access Token eintauschen. Bevor die Webanwendung das Access Token erhält, wird der Redirection Endpoint in der Webanwendung angesprochen. Der Redirection Endpoint ist in der Java-Klasse “OAuth2Controller.java” implementiert. Listing 27 zeigt den Redirection Endpoint innerhalb dieser Klasse.

Listing 27: Aufbau der Klasse OAuth2Controller.java

```
1  @Path("oauth")
2  public class OAuth2Controller {
3      @GET
4      @Path("cb")
5      public Response cb(@QueryParam("code") String code, @QueryParam("state") String state) {
6          TokenResult tokenResult = authFlow.finish(code, state);
7
8          setAccessToken(tokenResult.getAccessToken());
9
10         return Response.seeOther(UriBuilder.fromUri("http://localhost:8080/restservice/service/hello").build())
11             .build();
12     }
13 }
```

Innerhalb der Klasse wird aus dem URI, den die Webanwendung vom Authorization Server erhalten hat, der Paramer für den Authorization Code ausgelesen und im Anschluss damit ein Access Token angefragt. Dieser und weitere Schritte wird durch die Implementierung von Jersey verborgen.

Sobald ein Access Token in der Webanwendung vorhanden ist, kann damit der Zugrff auf die Benutzerdaten von Dropbox ausgeführt werden. Die Benutzerdaten von Dropbox werden als Objekt im Format JSON zurückgegeben, weshalb der Benutzername zusätzlich noch aus dem Objekt ausgelesen werden muss.

Listing 28: Zweiter Teil der WelomeResource.java

```
1  @Path("/")
2  public class WelcomeResource{
3      @GET
4      @Path("hello")
5      @Produces("text/plain")
6      public Response getHello() {
7          ...
8          else{
9              Client client = OAuth2Controller.getAuthFlow().getAuthorizedClient();
10
11             JsonObject json = client.target("https :// api.dropbox.com")
12                 .path("1/account/info")
13                 .request(MediaType.APPLICATION_JSON)
14                 .get(JsonObject.class);
15
16             String username = json.getString("display_name");
17
18             return Response.ok("Willkommen " + username).type(MediaType.TEXT_PLAIN).build();
19         }
20     }
21 }
```

5 Fazit

Das Protokoll OAuth dient zur Autorisierung von API-Zugriffen für verschiedene Anwendungen. Mittels des Protokolls können diese Anwendungen für den Zugriff auf Ressourcen im Namen des Eigentümers autorisiert werden. Sie erhalten dafür ein Zugriffstoken, mit dem der Zugriff stellvertretend stattfinden kann. Ressourcen, die ein Benutzer schon bei anderen Diensten besitzt, müssen daher kein zweites Mal verwaltet werden und die Angabe von Anmeldedaten in Form von Benutzername und Passwort wird während des Zugriffs vermieden. Typischerweise wird OAuth dazu verwendet, um die Kommunikation zwischen zwei Diensten abzusichern, welche durch einen Benutzer initiiert wird. Ein Beispiel hierfür ist der Zugriff einer Webanwendung auf die Dokumente eines Benutzers, die bei einem Filehoster wie Dropbox gelagert sind. In der Praxis wird das Protokoll allerdings auch oft verwendet, um die Benutzeranmeldung bei Anwendungen mit den Accounts von Plattformen wie Google oder Facebook zu ermöglichen und somit den Authentifzierungsvorgang für die Benutzer zu erleichtern.

Bei OAuth 2 handelt es sich um die Spezifikation der zweiten Version des Protokolls, welche auch als Framework bezeichnet wird. Allerdings ist damit kein Framework im eigentlichen Sinne gemeint, sondern die Tatsache, dass einige Abläufe flexibel gestaltet sind. Die Trennung der Ausstellung eines Tokens von dem Zugriff auf die Ressource ermöglicht es Dienstanbietern, dieselbe API für unterschiedliche Anwendungen zu verwenden und gibt ihnen zusätzlich Auskunft darüber, wer mit ihrem Dienst kommuniziert. Die Spezifikation des Protokolls lässt allerdings auch einige Punkte offen, anstatt alle möglichen Entscheidungen zu beschreiben. Ein Zugriffstoken erhält zwar eine Lebensdauer, es wird aber keine Angabe darüber gemacht, wie lang diese ausfallen soll. Das Token hat daher immer eine Gültigkeit für den ausgestellten Zeitraum, auch wenn der Benutzer der Webanwendung bereits ausgeloggt ist. Außerdem werden keine Angaben dazu gemacht, wie ein Zugriffstoken oder ein Parameter, der seitens des Protokolls genutzt wird, zu überprüfen ist.

Da OAuth 2 für client- und serverseitige Webanwendungen sowie Desktop- und mobile Anwendungen konzipiert wurde, werden innerhalb der Spezifikation des Protokolls unterschiedliche Abläufe für die verschiedenen Anwendungsfälle definiert. Der vielseitigste Ablauf ist der Authorization Code Flow, welcher browserbasiert ist. Der Ablauf eignet sich vor allem für serverseitige Webanwendungen und mobile und Desktopanwendungen, die fähig sind, wichtige Daten geheimhalten zu können. Der Implicit Flow stellt eine vereinfachte Variante für clientseitige Webanwendungen dar, bei denen keine Daten vertraulich behandelt werden können. Für vertrauensvolle Anwendungen ermöglicht das Protokoll einen Ablauf, mit dem ein Benutzername und das dazugehörige Passwort in ein Access Token eingetauscht werden können. Eine direkte Verwendung von Benutzernamen und Passwort führt jedoch dazu, dass die Interaktion zwischen dem Benutzer und dem Authorization Server des Anbieters entfällt, wodurch der Benutzer keine Kontrolle mehr über die Art der Zugriffe erhält. Daher bieten viele Dienstanbieter von OAuth 2 diesen Ablauf gar nicht erst an. Benötigt eine Anwendung den Zugriff auf die Ressourcen nicht im Namen eines Benutzers, bietet das Protokoll die Möglichkeit einen Ablauf zu verwenden, bei dem der anwendungsweite Zugriff mit einem Token eingeräumt werden kann.

Die Integration des Protokolls ist im Vergleich zur Vorgängerversion einfacher geworden, da Entwickler sich nicht mehr mit kryptografischen Verfahren auseinandersetzen müssen. Zusätzlich wird die Verwendung des Protokolls durch die Nutzung von Libraries vereinfacht. Das Protokoll hat daher gute Voraussetzungen, um zum Standard für die Authentifizierung und Autorisierung im Web zu werden und damit die HTTP-Authentifizierung zu erweitern.

Literaturverzeichnis

Allmaraju, S. (2010) RESTful Web Services Cookbook: Solutions for Improving Scalability and Simlicity, Sebastopol.

Berners-Lee, T. / **Fielding R.** / **Masinter, L. (2005):** Uniform Resource Identifier (URI): Generic Syntax, http://tools.ietf.org/html/rfc3986, 03.08.14.

Boyd, R. (2012): Getting Started with OAuth 2.0: Programming Clients for Secure Web API Authorization and Authetication, Sebastopol.

Dierks, T. / **Rescorla, E. (2008):** The Transport Layer Security (TLS) Protocol - Version 1.2, http://tools.ietf.org/html/rfc5246, 02.08.14.

Evans, D.-S. / **Hagiu, A.** / **Schmalensee, R. (2006):** Invisible Engines: How Software Platforms Drive Innovation and Transform Industries, Massachusetts.

Fielding, R. / **Reschke, J. (2014)** Hypertext Transfer Protocol (HTTP/1.1): Authentication, http://tools.ietf.org/html/rfc7235, 11.08.14.

Fielding, R. / **Reschke, J. (2014)** Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, http://tools.ietf.org/html/rfc7231, 04.08.14.

Franks, J. / **Hallam-Baker, P.** / **Hostetler, J. u.a. (1999):** HTTP Authentication: Basic and Digest Access Authentication, http://tools.ietf.org/html/rfc2617, 27.08.14.

Gourley, D. u.a. (2002): HTTP: The Definite Guide, Sebastopol.

Hammer-Lahav, E. (2012): OAuth 2.0 and the Road to Hell, http://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/, 07.09.14.

Hammer-Lahav, E. (2010): OAuth 2.0 (without Signatures) is Bad for the Web, http://hueniverse.com/2010/09/15/oauth-2-0-without-signatures-is-bad-for-the-web/, 07.09.14.

Hammer-Lahav, E. (2007): Explaining OAuth, http://hueniverse.com/2007/09/05/explaining-oauth/, 09.08.14.

Hammer-Lahav, E. / **Barth, A.** / **Adida B. (2011):** HTTP Authentication: MAC Access Authentication, http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-00, 06.09.14.

Hansen, H.-R. / **Neumann, G. (2007):** Arbeitsbuch Wirtschaftsinformatik: IT-Lexikon, Aufgaben und Lösungen, 7. Aufl., Stuttgart.

Hardt, D. (2012): The OAuth 2.0 Authorization Framework, http://tools.ietf.org/html/rfc6749, 05.08.14.

Hiller, J. / **Lodderstedt, T. (2012):** Jederzeit und überall: Flexible und sichere Internet-Dienste mit OAuth 2.0, in iX, H. 10, S. 126.

Josefsson, S. (2006) The Base16, Base32, and Base64 Data Encodings, http://tools.ietf.org/html/rfc4648, 12.08.14.

LeBlanc, J. (2011): Programming Social Applications: Building Viral Experiences with OpenSocial, OAuth, OpenID, and Distributed Web Frameworks, Sebastopol.

Makice, K. (2009): Twitter API: Up and Running, Sebastopol.

Mills, W. / **Showalter, T.** / **Tschofenig, H. (2013)**: A set of SASL Mechanisms for OAuth: draft-ietf-kitten-sasl-oauth-12.txt, http://tools.ietf.org/html/draft-ietf-kitten-sasl-oauth-12, 06.09.14.

OAuth.net (2014): OAuth Security Advisory: 2014.1 “Covert Redirect”, http://oauth.net/advisories/2014-1-covert-redirect/, 09.08.14.

OAuth.net (2009): OAuth Security Advisory: 2009.1, http://oauth.net/advisories/2009-1/, 09.08.14.

Richardson, L. / **Amundsen, M. (2013)** RESTful Web APIs: Services for a Changing World, Sebastopol.

Russel, M.-A. (2014): Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub and more, 2. Aufl., Sebastopol.

Spasovski, M. (2013): OAuth 2.0 Identity and Access Management Patterns: A practical hands-on guide to implementing secure API authorization flow scenarios with OAuth 2.0, Birmingham.

Stahlknecht, P. / **Hasenkamp, U. (2006):** Arbeitsbuch Wirtschaftsinformatik, Berlin u.a..

Tanenbaum, A.-S. / **Wetherall, D.-J. (2011):** Computer Networks, 5. Aufl., Boston.

Tanenbaum, A.-S. / **Van Steen, M. (2008):** Verteilte Systeme: Prinzipien und Oaradigmen, 2. Aufl., München.

Tilkov, S. (2011): REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien, 2. aktualisierte und erweitere Auflage, Heidelberg.

Wind, M. / **Kröger, D. (2006):** Handbuch IT in der Verwaltung, Berlin.

Ehrenwörtliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Bachelorthesis selbstständig und ohne unerlaubte fremde Hilfe angefertigt habe, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Königslutter am Elm, den 12. September 2014