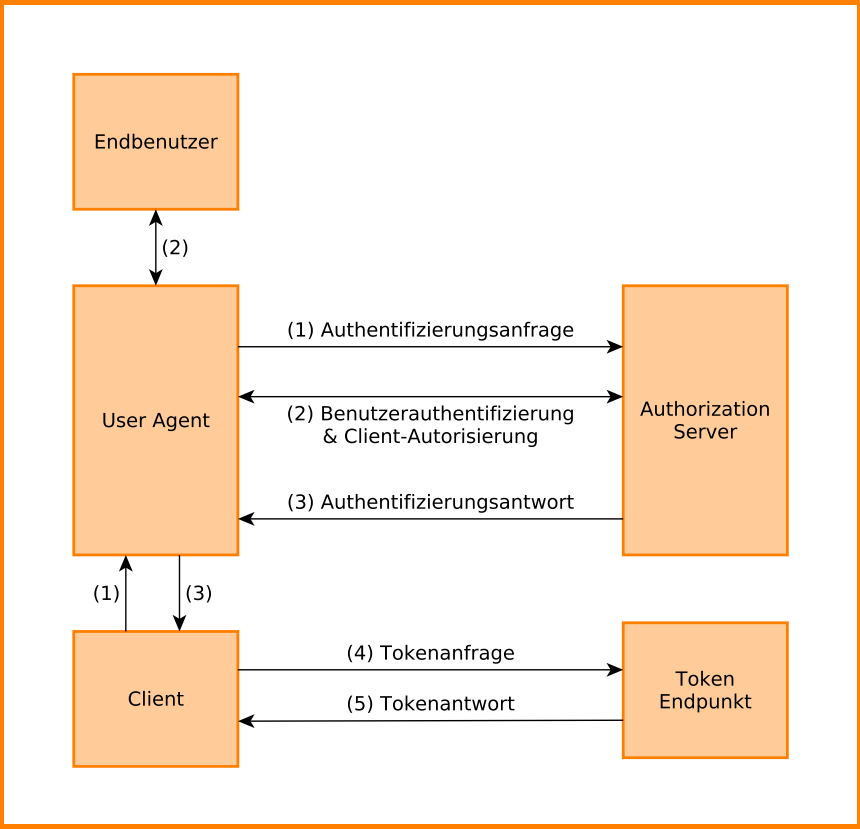
Open C3S
Open Competence Center for Cyber Security

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik



Mikromodul 8007: Identitätsmanagement und Single Sign-on

Autoren:
Prof. Dr. Hans P. Reiser
Noëlle Rakotondravony
Johannes Köstler

Universität Passau

Mikromodul 8007: Identitäts-management und Single Sign-on

Autoren:

Prof. Dr. Hans P. Reiser

Noëlle Rakotondravony

Johannes Köstler

---

1. Auflage

Universität Passau

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

AUFSTIEG DURCH BILDUNG >>
OFFENE HOCHSCHULEN

Universität Passau
Fakultät für Informatik und Mathematik
Innstraße 43
94034 Passau

1. Auflage (4. Mai 2017)

Um die Lesbarkeit zu vereinfachen, wird auf die zusätzliche Formulierung der weiblichen Form bei Personenbezeichnungen verzichtet. Wir weisen deshalb darauf hin, dass die Verwendung der männlichen Form explizit als geschlechtsunabhängig verstanden werden soll.

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 16OH12025 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

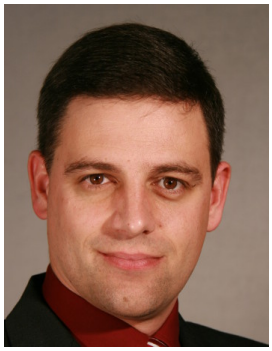
Inhaltsverzeichnis

Einleitung

I. Abkürzungen der Randsymbole und Farbkodierungen

Beispiel	B
Definition	D
Exkurs	E
Übung	Ü

II. Zu den Autoren



Hans P. Reiser ist Juniorprofessor für Sicherheit in Informationssystemen an der Universität Passau. Schwerpunkte seiner Arbeitsgruppe sind die Weiterentwicklung von Konzepten und Systemen aus dem Bereich der fehler- und einbruchstoleranten Replikation, die frühzeitliche und umfassende Erkennung von Sicherheitsproblemen und Sicherheitsvorfällen in Cloud-Umgebungen sowie die Erforschung neuartiger Sicherheitskonzepte auf Hypervisorebene.

Noëlle Rakotondravony ist seit September 2015 wissenschaftliche Mitarbeiterin in der Arbeitsgruppe von Prof. Hans P. Reiser an der Universität Passau.

Johannes Köstler ist seit Mai 2015 wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Prof. Hans P. Reiser an der Universität Passau.

Mikromodul: Identitätsmanagement und Single Sign-on

1 Lernziele

Nach Bearbeitung dieses Mikromoduls sind Sie mit den Grundlagen von Authentisierung, Autorisierung und Zugriffskontrolle vertraut. Sie kennen die grundlegende Funktionsweise von Single-Sign-on-Systemen. Sie sind mit praxisüblichen Systemen vertraut, insbesondere vertieft mit den Protokollen OAuth und OpenID Connect, und können deren sicherheitsrelevanten Eigenschaften bewerten. Darüber hinaus sind Sie in der Lage, OpenID Connect und OAuth selbst zu verwenden.

2 Identitätsmanagement

Um den Begriff des Identitäts- und Zugriffsmanagements[1] etwas genauer zu definieren, greifen wir aus vielen leicht unterschiedlichen Varianten in der Literatur die Definition des IT-Glossars von Gartner auf (siehe Definition 1).

Definition 1: Identitäts- und Zugriffsmanagement

„Identity and access management (IAM) is the security discipline that enables the right individuals to access the right resources at the right times for the right reasons.“ (Gartner Inc.)

D

Grundsätzlich zielt also Identitäts- und Zugriffsmanagement darauf ab, Regeln zu verwalten und umzusetzen, wer wann warum auf was zugreifen darf. Im folgenden Abschnitt 2.1 werden zunächst die beteiligten Akteure etwas näher charakterisiert. Darauf aufbauend beschreibt der Abschnitt 2.2 die Interaktionen zwischen diesen und definiert die Begriffe der Identifizierung, Authentisierung, Authentifizierung und Autorisierung. In Abschnitt 2.3 betrachten wir die wichtigsten Modelle der Zugriffskontrolle.

2.1 Allgemeine Grundbegriffe

Zunächst soll die Terminologie, mit der das *„Wer“* bezeichnet wird, etwas genauer definiert werden. Bei diesem aktiven Element kann es sich zum einen um eine natürliche oder juristische Person handeln, welche wir im Folgenden als *Subjekt* bezeichnen. Es kann sich aber auch um eine technische Komponente wie beispielsweise einen Rechner oder einen Dienst handeln. Als Oberbegriff für diese beiden Arten von aktiven Elementen dient im Weiteren die Bezeichnung *Entität* oder *Akteur*.

Beim Ziel des Zugriffs (*„Was“*) handelt es sich ebenfalls um eine Entität oder um eine *Ressource*. Diese bezeichnen wir im Folgenden auch als *Ziel* bzw. als *Objekt*.

Dieser Entität ist eine *Identität* zugeordnet. Auf eine ausführliche philosophische Diskussion des Begriffs der *Identität* verzichten wir an dieser Stelle und beschränken uns auf eine einfache Definition: Die *Identität* charakterisiert als Attribut bzw. Menge von Attributen eine Entität.

[1] Synonym dazu ist auch der Begriff *Zugangsmanagement* üblich.

2.2 Identifizierung, Authentisierung, Authentifizierung und Autorisierung

Die vier zentralen Begriffe der Identifizierung, Authentisierung, Authentifizierung und Autorisierung werden in der Praxis leider oft ungenau oder untereinander austauschbar verwendet. Eine differenzierte Verwendung dieser Begriffe erlaubt es aber, die einzelnen Aspekte genauer zu benennen.

D

Definition 2: Identifizierung

Identifizierung heißt, dass eine bestimmte Identität von einer Entität vorgegeben (behauptet) wird oder dass diese Identität aus anderen Beobachtungen abgeleitet wird.

Im Sinne der Definition 2 verstehen wir also einen Vorgang, bei dem eine Identität einer Entität zugeordnet wird, ohne die Korrektheit dieser Zuordnung zu prüfen. Hierzu ein einfaches Beispiel: Bei einer passwortbasierten Anmeldung eines Clients an einem Server besteht die *Identifizierung* des Clients darin, dass der Client dem Server seinen Benutzernamen mitteilt.

Die Prüfung der Identität ist der nächste Schritt nach der Identifizierung. Während im Englischen hierbei nur ein Verb „to authenticate“ verwendet wird, gibt es im Deutschen differenzierter die beiden Begriffe „authentisieren“ und „authentifizieren”, die beide den gleichen Vorgang aus unterschiedlicher Perspektive beschreiben (siehe Definition 3 und 4).

D

Definition 3: Authentisierung

Authentisierung bedeutet, dass eine Entität gegenüber einer anderen Entität einen Nachweis erbringt, dass sie eine bestimmte Identität besitzt.

D

Definition 4: Authentifizierung

Authentifizierung bedeutet, dass eine Entität die Korrektheit einer von einer anderen Entität behaupteten/abgeleiteten Identität verifiziert.

Auch im Englischen kann man diese Differenzierung nachvollziehen, zum Beispiel bei einer Interaktion zwischen einem Client und einem Server: „the client authenticates to the server“ vs. „the server authenticates the client“. Im zuvor genannten einfachen Beispiel einer passwortbasierten Anmeldung eines Clients an einem Server besteht also die Authentisierung darin, dass der Client als Nachweis seiner Identität sein Passwort an den Server sendet, während die Authentifizierung darin besteht, dass der Server prüft, ob das erhaltene Passwort mit einem lokal gespeicherten Vergleichswert übereinstimmt.

Von Authentisierung und Authentifizierung klar zu unterscheiden ist die *Autorisierung*. Während die ersten beiden Konzepte auf die Verifikation der Identität abzielen, geht es bei der Autorisierung darum, welche Zugriffsrechte dieser Identität eingeräumt werden.

Definition 5: Autorisierung

Die *Autorisierung* einer Entität bestimmt, mit welchen anderen Entitäten diese interagieren bzw. auf welche Ressourcen diese zugreifen darf.

D

Im einfachsten Fall kann die Autorisierung eine binäre Entscheidung sein: Im obigen Beispiel kann z. B. der Server entscheiden, ob der Client nach seiner Authentisierung mit dem Server interagieren darf oder nicht. In vielen Fällen geht es aber um wesentlich komplexere Entscheidungen, auf welche Teilmenge von einer Vielzahl von Ressourcen eine Entität wann und wie zugreifen darf. Daher betrachten wir im nächsten Abschnitt diesen Aspekt der Zugriffskontrolle näher.

2.3 Zugriffskontrolle

Grundsätzlich unterschieden wird bei der Zugriffskontrolle zwischen dem Modell *Discretionary Access Control (DAC)* bzw. *benutzerbestimmmbare Zugriffskontrolle* einerseits und *Mandatory Access Control (MAC)* bzw. *zwingend erforderliche Zugriffskontrolle* andererseits. Bei MAC regelt eine zentrale Verwaltungsinstanz, welche Subjekte wann auf welche Ziele zugreifen dürfen.

Bei MAC-System sind vor allem *Multi-Level-Sicherheitssysteme (MLS)* üblich. Hierbei wird zentral sowohl jedem Subjekt als auch jedem Ziel eine *Schutzstufe* zugeordnet. Die bekanntesten Modelle, nach denen Zugriffskontrolle entsprechend den Schutzstufen erreicht wird, sind das *Bell LaPadula-Modell* (Bell und LaPadula, 1973) und das *Biba-Modell* (Biba, 1977).

Das Bell LaPadula-Modell zielt auf die *Vertraulichkeit* von Daten. Es ist charakterisiert durch die Eigenschaften „no-read-up“ und „no-write-down“. Die erste Eigenschaft besagt, dass ein Subjekt auf keine Ziele mit höherer Schutzstufe lesend zugreifen darf, die zweite besagt, dass ein Subjekt keine Ziele mit niedrigerer Schutzstufe verändern darf. Ein Informationsfluss von höherer zu niedrigerer Schutzstufe ist dadurch nicht möglich.

Das Biba-Modell zielt dagegen auf die *Integrität* von Daten, und lässt sich durch die Eigenschaften „no-write-up“ und „no-read-down“ charakterisieren. Die erste Eigenschaft besagt, dass ein Subjekt keine Ziele mit höherer Schutzstufe verändern darf, während die zweite Eigenschaft besagt, dass ein Subjekt auf keine Ziele mit niedrigerer Schutzstufe lesend zugreifen darf. Ein Informationsfluss von niedrigerer zu höherer Schutzstufe ist dadurch nicht möglich.

Bei DAC-Systemen werden Zugriffsrechte dagegen durch das Ziel (bzw. durch den Besitzer der Ziel-Ressource) festgelegt und können ggf. zwischen Subjekten weitergegeben werden.

Abstrakt beschreiben lassen sich die Zugriffsrechte von einer Menge von Subjekten S auf eine Menge von Objekten O durch eine Zugriffskontrollmatrix (engl. „access control matrix“), wie sie erstmals von Lampson (1971) beschrieben wurde. Bei dieser Matrix werden die Zeilen jeweils einem Subjekt zugeordnet, die Spalten jeweils einem Objekt. Das Matrix-Element, das zu einer bestimmten Zeile und Spalte gehört, enthält *Zugriffsattribute*, welche die Zugriffsberechtigung des entsprechenden Subjekts auf das korrespondierende Objekt beschreiben. Typische Attribute in der Matrix sind „Lesen“ sowie „Schreiben“. Mit Attributen kann aber auch beschrieben werden, ob das Subjekt die Zugriffsattribute verändern darf, also z. B. einem anderen Subjekt Zugriffsrechte einräumen kann.

Während die Zugriffskontrollmatrix als abstraktes Modell gut geeignet ist, die Zugriffs-Beziehungen zwischen Subjekten und Objekten zu beschreiben, ist eine direkte Implementierung durch eine zentrale Instanz, die alle Attribute in Matritzenform verwaltet, insbesondere bei einer großen Zahl an Subjekten und Objekten kaum praktikabel. Eine Implementierung in der Praxis geht daher im Allgemeinen anders vor. Die zwei einfachsten Varianten sind Zugriffskontrolllisten (*Access Control Lists – ACLs*) und Capability-Listen (*C-Lists*).

Bei Zugriffskontrolllisten erfolgt eine *objektbezogene* Speicherung der Zugriffs-Attribute, d. h. zusammen mit jedem Objekt wird eine ACL gespeichert, die Informationen über alle Zugriffsberechtigungen auf dieses Objekt enthält. Ein einfaches Beispiel für ACL-basierte Zugriffskontrolle sind übliche Dateisysteme: Dort werden in den Metainformationen einer Datei Angaben darüber gespeichert, welche Nutzer auf diese Datei wie zugreifen dürfen.

Bei Capability-basierter Zugriffskontrolle dagegen wird die Information über Zugriffsberechtigungen auf Subjektseite gespeichert. Die C-Liste eines Subjekt enthält die Berechtigungen, mit der dieses Subjekt auf Objekte zugreifen kann. Bei der Implementierung von Capabilities ist zu gewährleisten, dass ein Subjekt diese subjektbezogenen Berechtigungen nicht beliebig manipulieren kann. Dieser Manipulationsschutz kann auf verschiedene Weise implementiert sein.

Betrachten wir zunächst ein verteiltes System aus Nutzern und Diensten. Eine Capability kann hier beispielsweise durch ein von einem Dienst digital signiertes Token repräsentiert sein. Durch den Besitz dieses Tokens kann der Client seine Zugriffsberechtigung nachweisen. Durch die Signatur ist sichergestellt, dass ein Client keine beliebigen Token fälschen kann.

Ein weiteres Beispiel stammt aus dem Bereich der Betriebssysteme. Öffnet ein Anwendungsprogramm eine Datei, so wird bei dieser Operation (Datei öffnen) die Berechtigung des Nutzers geprüft. Ist der gewünschte Zugriff erlaubt, so wird der Anwendung ein Dateideskriptor (*file handle*) ausgestellt, der eine Capability darstellt. Wird über den Dateideskriptor auf die Datei zugegriffen, so erfolgt keine erneute Prüfung auf Grundlage der Zugriffskontrollliste der Datei. Stattdessen ergibt sich die Berechtigung aus dem Besitz des Dateideskriptors. In dieser Variante ist der Deskriptor vor Manipulation dadurch geschützt, dass die Daten des Deskriptors vor direktem Zugriff der Anwendung geschützt im Adressraum des Betriebssystems gespeichert werden, und die Anwendung nur einen numerischen Index erhält, der auf diese Daten verweist.

Die bisher beschriebenen Formen der Autorisierung stellen eine wesentliche Grundlage dar, sind aber nicht ausreichend, um Autorisierung in heutigen komplexen Systemen hinreichend zu beschreiben. Eine direkte Zuordnung von Berechtigungen zu Subjekten wird bei einer hohen Zahl von Subjekten unübersichtlich und daher aufwändig zu verwalten und fehlerträchtig. Von Ferraiolo und Kuhn (1992) wurde daher das Modell der rollenbasierten Zugriffskontrolle (RBAC) vorgeschlagen. In diesem Modell werden Subjekten Rollen zugeordnet und das Zugriffsrecht auf Objekte ist an eine Rollenzugehörigkeit gebunden. Das RBAC-Modell stellt heute die Grundlage für die meisten Identitätsmanagement-Systeme (IDM) dar.

2.4 Single Sign-on

Single Sign-on ist ein Authentifizierunskonzept, welches die Nutzung eines einzigen Benutzerprofils bei mehreren Diensten erlaubt. Aktive Sitzungen können so einfach übertragen werden. Früher fand dieses Konzept hauptsächlich in abgeschlossenen Umgebungen – wie beispielsweise Unternehmens- oder Organisationsnetzwerken

– Anwendung. Den bekanntesten Vertreter stellt in diesem Zusammenhang das *Kerberos*-Protokoll dar (Neuman et al., 2005). Es überträgt den Anmeldeprozess von einer Vielzahl unabhängiger Dienste zu einem zentralen Authentifizierungsdienst, der gegenüber den anderen Diensten als vertrauenswürdige dritte Partei auftritt. Nutzer, die sich gegenüber dem zentralen Dienst identifiziert und authentisiert haben, können sich diese Vorgänge durch spezielle Tickets beglaubigen lassen und somit auf andere Dienste übertragen.

Mit der wachsenden Zahl an Internetdiensten nahmen auch die Bestrebungen zu, dieses Konzept auf das Internet zu übertragen. Mittlerweile existiert eine Vielzahl von Lösungen, deren Ansätze sich durchaus ähneln. An dieser Stelle wollen wir die Beispiele *OpenID*, *Shibboleth*, und *Mozilla Persona* erwähnen.

OpenID ist ein Beispiel für solch eine Single-Sign-on-Lösung im Internet. Das Protokoll ist mittlerweile überholt und wurde von seinem Nachfolger *OpenID Connect* abgelöst, welches in Abschnitt 4 behandelt wird. In beiden Varianten können sich Internetnutzer Identitäten bei so genannten *OpenID Providern* registrieren. Teilnehmende Internetdienste erlauben die Verwendung dieser Identitäten im eigenen Anwendungsbereich und verweisen die Nutzer zur Authentifizierung an die Provider. Diese bestätigen den Diensten erfolgreiche Authentifzierungsvorgänge und liefern Informationen über die registrierten Identitäten.

Shibboleth[2] ist ein weiterer Vertreter dieser Gruppe und basiert auf der Security Assertion Markup Language (SAML) (Lockhart und Campbell, 2008). Nicht authentifizierte Benutzer, die auf geschützte Ressourcen von teilnehmenden Dienstanbietern zugreifen möchten, werden zur Authentisierung an Identitätsprovider weitergeleitet. Nach einer erfolgreich verlaufenen Authentifizierung senden diese eine SAML Assertion zur Bestätigung der erfolgten Authentifizierung zurück an den Dienstanbieter.

Mozilla Persona folgt einem ähnlichen Prinzip, setzt für den Nachweis der Authentifizierung aber auf Zertifikate (Mozilla Corporation, 2014). Die Identitätsprovider stellen bei der initialen Authentifizierung ein Zertifikat aus welches für einen bestimmten Zeitraum bei den teilnehmenden Dienstanbietern zur Authentifizierung genutzt werden kann. Dies resultiert in einem höheren Schutz der Privatsphäre der Nutzer, da die Identitätsprovider nicht bei jeder Interaktion mit den Dienstanbietern hinzugezogen werden müssen. Das Projekt wurde aber aufgrund von mangelnder Akzeptanz bereits im Jahre 2016 wieder eingestellt.

Generell lässt sich festhalten, dass Single-Sign-on-Lösungen das Identitäts- und Zugriffsmanagement für beide beteiligten Gruppen – auf der einen Seite die Nutzer internetbasierter Dienste und auf der anderen Seite die Dienstanbieter – deutlich vereinfachen. Während sich für erstere die Zahl der zu verwaltenden Authentifizierungsinformationen drastisch verringert, profitieren letztere von der Auslagerung des Authentifizierungsprozesses auf den vertrauenswürdigen Dritten. Dies kann die Sicherheit erhöhen, da eine sichere Verwahrung der Authentifizierungsinformationen nur an einer Stelle erfolgen muss und dort potenziell mit höherem Aufwand geschützt werden kann, und weil die Reduktion auf wenige Authentifizierungsimplementierungen die Anzahl möglicher Sicherheitslücken potenziell verringert. Gleichzeitig muss aber auch festgehalten werden, dass die wenigen verbleibenden Authentifizierungsinformationen nun den Zugang zu einer Vielzahl von Diensten ermöglichen und somit ein weitaus interessanteres Ziel für Angreifer darstellen. Ferner besteht die Gefahr des Missbrauchs der Informationen durch den Identitätsprovider, weshalb ein ausgeprägtes Vertrauensverhältnis beim Einsatz solcher Lösungen erforderlich ist.

[2] http://shibboleth.net [abgerufen am 2017-03-01]

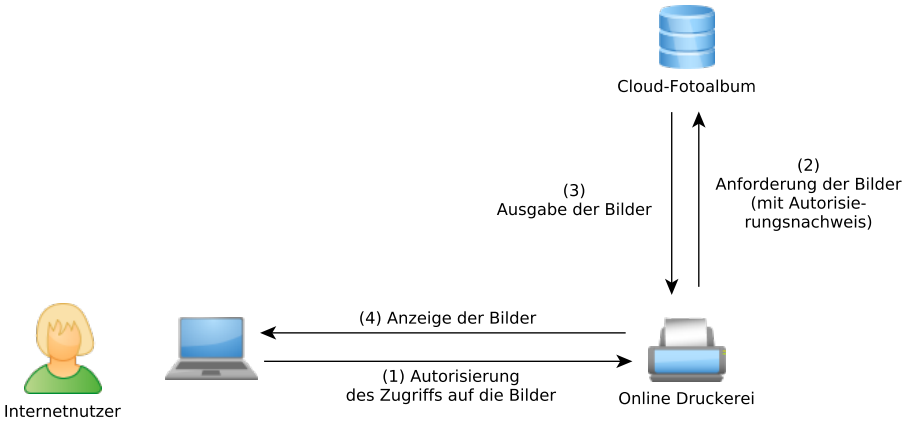


Abb. 1: Die Abläufe bei einer API-Autorisierung am Beispiel einer Online-Druckerei, bei der ein Nutzer die Druckerei autorisiert, direkt auf die Bilder in einem Cloud-Fotoalbum zuzugreifen.

3 OAuth 2.0

Dieser Abschnitt befasst sich mit dem *OAuth*-Protokoll, welches eine standardisierte Autorisierung von Programmierschnittstellen – *Application Programming Interfaces* (APIs) – verschiedener Applikationen und Dienste ermöglicht. Mit dem offenen Protokoll können Anwendungen untereinander geschützte Benutzerdaten abfragen und austauschen.

Ein typischer Anwendungsfall für eine derartige API-Autorisierung wird durch das folgende Beispiel verdeutlicht (vgl. Abbildung 1): Ein Internetnutzer möchte Bilder aus einem cloudbasierten Fotoalbum bei einer Online-Druckerei drucken. Die Druckerei nutzt dabei die API des Bilddienstes (Cloud-Fotoalbum) um – nach erfolgter Zustimmung des Nutzers – die Bilder des Nutzers anzuzeigen und gegebenenfalls zu drucken. Die erforderliche Autorisierung der anfragenden Anwendung kann durch den Benutzer ohne die Weitergabe seines Passworts (oder anderer Authentisierungsinformationen) erfolgen. Nach einer allgemeinen Einführung beschreibt dieses Kapitel detailliert die einzelnen Schritte der am häufigsten eingesetzten Protokollvarianten.

3.1 Hintergrund

Die Anfänge der Entwicklung von OAuth gehen zurück bis in das Jahr 2006 und fallen damit in eine Zeit, die stark von den Ideen des *Web 2.0* geprägt war. Soziale Netzwerke wie *MySpace* und *Facebook* entstanden, Inhalte wurden nicht mehr ausschließlich von Redaktionen sondern zunehmend von den Internetnutzern selbst erstellt, und mit so genannten *Mashups* gingen durch die Kombination von Daten aus verschiedenen bestehenden Internetangeboten völlig neue und innovative Angebote hervor. Für die beteiligten Applikationen an solchen Mashups existierte lange Zeit keine standardisierte Vorgehensweise zur Autorisierung des Datenaustausches.

Diese Lücke versuchte OAuth mit einem neuartigen Protokoll zu schließen. Kern des Protokolls bildet damals wie heute die Ausstellung von *Zugriffstoken*, welche – vom Benutzer autorisiert – eine zeitlich begrenzte Abfrage von genau festgelegten Datensätzen ermöglichen. 2007 erschienen neben ersten öffentlichen Entwürfen einer Spezifikation von OAuth auch erste Implementierungen dieser Spezifikation. Am Ende der Bemühungen stand die Fertigstellung des *OAuth Core 1.0 final draft*. Bis zur Veröffentlichung des finalen OAuth-1.0-Protokolls als *RFC 5849* (Hammer-Lahav, 2010) im April 2010 entwickelte sich OAuth zum De-Facto-Standard im Bereich der *API-Autorisierung*. Eine Revision des Protokolls erfolgte im Oktober 2012 mit der Spezifizierung des *OAuth-2.0-Frameworks* als *RFC 6749* (Hardt, 2012).

3.2 Konzept

Bei der Authentifizierung im klassischen Client-Server-Modell erfolgt die Freigabe des Client-Zugriffs auf geschützte Ressourcen nachdem sich der Client gegenüber dem Server mit Hilfe seiner Authentifizierungsinformation als Besitzer der Ressourcen ausgewiesen hat. Eine einfache Übertragung dieses Modells auf den Datenaustausch mit Drittanbieteranwendungen hätte zur Folge, dass die Anmeldedaten auch der Drittanbieteranwendung kenntlich gemacht werden müssten, damit diese die gewünschten Daten stellvertretend für den Benutzer abfragen kann. Diese Vorgehensweise lässt sich auch heute noch gelegentlich finden: Manche Anwendungen wie beispielsweise Social Networks oder Messaging-Dienste ermöglichen z. B. den Import von Kontakten von einem E-Mail-Provider, indem man die Anmeldedaten für den E-Mail-Dienst an diese Anwendungen direkt weitergibt.

Solch eine Vorgehensweise führt aber gleich auf mehreren Ebenen zu Problemen. Zum einen ist es bei der Weitergabe der Anmeldeinformationen nicht möglich die Menge und Art der Daten zu beschränken sowie einzelne Datenzugriffsberechtigungen zeitlich einzugrenzen oder zurückzunehmen. Zum anderen vergrößert sich damit die potenzielle Angriffsfläche, da auch der Drittanbieter die Vertraulichkeit der Anmeldeinformationen sicherstellen muss und – im Falle einer Kompromittierung dieser Daten beim Drittanbieter durch einen Angreifer – auch die Daten der ursprünglichen Applikation gefährdet sind.

Das OAuth-Protokoll umgeht vorgenannte Probleme, indem es die Rollen des Clients und des Besitzers der Ressourcen unterscheidet und Möglichkeiten zur feingranularen Regelung des Zugriffs auf diese Ressourcen bereitstellt. Auf diese Weise ist es möglich, dass der Ressourcenbesitzer seine Zugriffsrechte für bestimmte Ressourcen und exakt definierte Zeiträume auf beliebige Clients überträgt. Die Zugriffsberechtigungen werden dabei als Token repräsentiert, welche eine einfache Übertragung ermöglichen.

3.3 Rollen

Der OAuth-Standard 2.0 nach *RFC 6749* (Hardt, 2012) definiert wie folgt vier Rollen (zu beachten ist, dass sich die Terminologie gegenüber früheren Versionen des Standards teilweise geändert hat):

Ressourcenbesitzer (Resource Owner): Beim *Ressourcenbesitzer* handelt es sich in der Regel um den Endbenutzer, welcher eine oder mehrere Ressourcen besitzt und Zugriffe auf diese Ressourcen autorisiert.

Ressourcenserver (Resource Server): Der *Ressourcenserver* stellt die Ressourcen bereit. Die Abfrage von geschützten Ressourcen, welche bestimmten Endbenutzern zugeordnet sind, erfolgt dabei durch Anfragen mit dem in Abschnitt 3.1 bereits kurz vorgestellten Zugrifftoken.

Client: Als *Client* wird jede Applikation bezeichnet, die im Auftrag des Ressourcenbesitzers und mit Hilfe dessen Autorisierung Ressourcen beim Ressourcenserver anfragt. Mögliche Applikationen sind dabei nicht auf bestimmte Implementierungen oder Geräte beschränkt. Allerdings können bestimmte Client-Implementierungen besondere Protokollvarianten erfordern, wie im nachfolgenden Abschnitt genauer erläutert wird.

Autorisierungsserver (Authorization Server): Der *Autorisierungsserver* stellt dem Client Zugriffstoken aus, nachdem der Ressourcenbesitzer sich erfolgreich authentisiert und den entsprechenden Zugriff autorisiert hat.

Zu beachten ist insbesondere, dass die Bezeichnung *Client* sich nicht wie man vielleicht erwarten könnte auf den Endbenutzer bezieht. Vielmehr bezeichnet sie eine Drittanwendung, die mit Erlaubnis des Endbenutzers auf externe Ressourcen des Endbenutzers zugreifen können soll.

Im vorherigen Anwendungsfall der Online-Druckerei (siehe Abbildung 1) entspricht der Endbenutzer, der Fotos online gespeichert hat und diese mit Hilfe der Druckerei drucken möchte, dem Ressourcenbesitzer. Die Fotos sind bei einem Cloud-Hoster gespeichert, welcher als Ressourcenserver auftritt. Um die Bilder bei der Online-Druckerei drucken zu lassen, muss die Druckerei auf die Bilder des Nutzers beim Ressourcenserver zugreifen. Die Online-Druckerei wird also zum Client des cloudbasierten Bilderdiensts. Den Zugriff regelt der Autorisierungsserver, welcher im Beispiel auch in die Zuständigkeit des Bilderdiensts fällt. Die Rollen des Ressourcenserver und Autorisierungsserver können physisch sowie logisch durch den gleichen Server implementiert sein – konzeptionell sind beide Rollen aber getrennt, um eine maximale Flexibilität zu gewährleisten. Damit ist es auch möglich, dass einem Autorisierungsserver mehrere Ressourcenserver zugewiesen sind.

3.4 Clients

Der OAuth-Standard unterscheidet zwischen zwei Arten von Clients: *öffentliche* und *vertrauliche Clients*. Das Hauptunterscheidungsmerkmal stellt hierbei die Fähigkeit eines Clients zur Authentisierung gegenüber dem Autorisierungsserver dar, welche in aller Regel über ein gemeinsames Geheimnis (Passwörter, kryptographische Schlüssel, etc.) realisiert wird:

- Clients in Form von Applikationen, die auf einem separaten Server ausgeführt werden, sind in der Lage ein solches Geheimnis sicher zu speichern und vor dem Ressourcenbesitzer sowie dem User Agent verborgen zu übertragen.
- Für mittels Skriptsprachen implementierte anwenderseitige Anwendungen (beispielsweise browserbasierte Anwendungen, die auf dem Rechner des Endbenutzers ausgeführt werden) sowie für native mobile Anwendungen auf einem Endgerät des Nutzers kann dies im Allgemeinen nicht garantiert werden.

Daher werden erstere Anwendungen in der OAuth-Terminologie als *vertrauliche Clients* bezeichnet, wohingegen letztere als *öffentliche Clients* benannt werden. Würden dennoch Geheimnisse in öffentlichen Clients gespeichert und verarbeitet werden, könnten diese von bösartigen Angreifern extrahiert werden, welche sich infolgedessen fälschlicherweise als legitime Clients ausgeben könnten. Da diese Tatsache die Client-Authentifizierung ad absurdum führen würde, ist dieser Schritt für öffentliche Clients nicht vorgesehen. Vielmehr existiert für öffentliche Clients eine eigene Protokollvariante mit besonderen Sicherheitseinschränkungen.

Bevor eine Client-Applikation mit Hilfe des OAuth-Protokolls geschützte Ressourcen nutzen kann, muss sich diese bei demjenigen Autorisierungsserver registrieren, der für diese Ressourcen verantwortlich zeichnet. Der genaue Ablauf dazu ist vom Standard nicht vorgegeben. In aller Regel werden dazu die Client-Eigenschaften beim Autorisierungsserver – beispielsweise über ein Webformular – registriert. Zu den Eigenschaften zählen: die Art des Clients, die URI, an die erfolgreich authentifizierte Clients zurückgeleitet werden sollen sowie weitere generelle Informationen wie der Name und eine Beschreibung der Applikation oder gesetzliche Bestimmungen, die bei der Nutzung der Dienste Anwendung finden. Der Autorisierungsserver

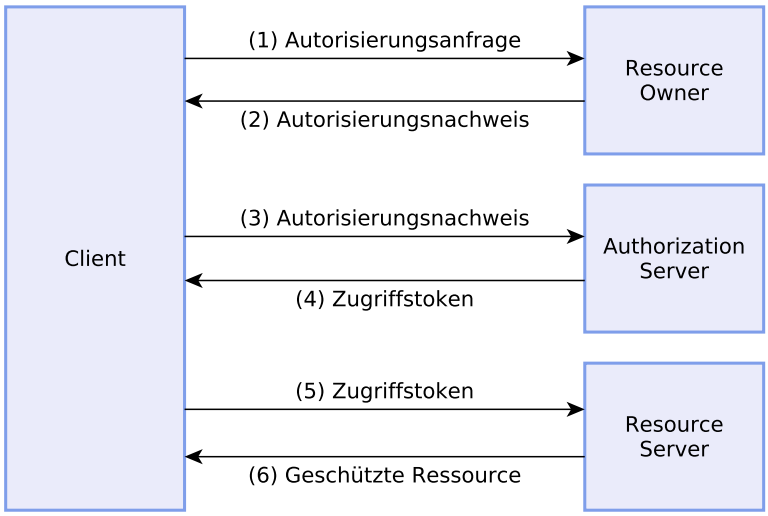


Abb. 2: Abstrakter Protokollfluss beim OAuth-2.0-Protokoll aus Perspektive der Client-Anwendung

vergibt daraufhin einen eindeutigen Client-Bezeichner und handelt im Falle eines vertrauenswürdigen Clients mit diesem ein gemeinsames Geheimnis aus. Die Registrierung des Clients beim Autorisierungsserver kann auch dynamisch und ohne direkte Interaktion beispielsweise über Dritte erfolgen. Weiterhin ist auch die Verwendung von nicht registrierten Clients denkbar. Dies würde aber eine weiterführende Sicherheitsanalyse erfordern, welche wiederum stark abhängig vom jeweiligen Anwendungsfall und der zugrundeliegenden Implementierung wäre und deshalb nicht Teil des eigentlichen Standards ist.

Auch zum eigentlichen Ablauf der Client-Authentifizierung gibt der OAuth-Standard lediglich Empfehlungen. So ist es letztendlich dem jeweiligen Autorisierungsserver überlassen, ob die Authentifizierungsinformationen des Clients (das gemeinsame Geheimnis) aus einem Passwort, einem asymmetrischen Schlüsselpaar oder einem anderweitigen Merkmal bestehen und wie diese sicher zwischen den Endpunkten übertragen werden. In der Praxis kommt jedoch zumeist ein Passwort zum Einsatz, welches entweder über HTTP Basic Authentication (siehe Exkurs 1) oder innerhalb des Request-Body ausgetauscht wird. Hierbei obliegt es dem Client-Entwickler die Vorgaben des Autorisierungsservers einzuhalten.

E

Exkurs 1: HTTP Basic Authentication

HTTP Basic Authentication ist ein einfaches Verfahren, mit dem sich ein Nutzer (bzw. stellvertretend dessen Web Browser) gegenüber einem Webserver authentisieren kann. Dazu schickt der Browser im Header der Anfrage an den Webserver eine Headerzeile der Form

```
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

Die Zeichenkette „`dXNlcm5hbWU6cGFzc3dvcmQ=`“ ist dabei die Base64-Kodierug der Zeichenkette `Benutzername:Passwort`. Wie sich einfach mit einem Base64-Dekoder prüfen lässt (beispielsweise online unter `https://www.base64encode.org/`) entspricht dieses Beispiel dem Benutzernamen „username“ und Password „password“. Da das Passwort also (abgesehen von der Base64-Kodierung, die keinen Sicherheitsgewinn bietet) im Klartext übertragen wird, sollte HTTP Basic Authentication ausschließlich über sichere Kommunikationskanäle (also mittels TLS verschlüsselt) verwendet werden.

3.5 Interaktionen

Die generelle Interaktion der gerade eingeführten Akteure wird konzeptionell in Abbildung 2 dargestellt. Sie gibt einen Überblick über die einzelnen Schritte, welche nötig sind, um den Zugriff auf eine geschützte Ressource zu autorisieren.

1. Zu Beginn bittet der Client den Ressourcenbesitzer um die Einräumung einer Nutzungsberechtigung. Diese Autorisierungsanfrage (*Authorization Request*) kann (wie abgebildet) direkt beim Ressourcenbesitzer gestellt werden. In der Praxis erfolgt diese Autorisierung zumeist über oder mit Beteiligung des Autorisierungsservers.
2. Der Client erhält daraufhin vom Ressourcenbesitzer einen digitalen Autorisierungsnachweis. Der Aufbau des Berechtigungsnachweises richtet sich nach der Art des jeweiligen Autorisierungsablaufs und wird in Abschnitt 3.6 näher erläutert. Der offizielle OAuth-Standard kennt vier verschiedene Typen, unterstützt aber auch die Verwendung selbsterstellter Erweiterungstypen.
3. Der Client benutzt im folgenden Schritt den ausgestellten Berechtigungsnachweis, um beim Autorisierungsserver ein Zugriffstoken (*Access Token*) anzufragen.
4. Dazu authentifiziert der Autorisierungsserver zuerst den Client, bevor er den Berechtigungsnachweis validiert. Falls beide Überprüfungen erfolgreich verlaufen stellt der Autorisierungsserver das Zugriffstoken aus.
5. Im letzten Request-Response-Zyklus erfolgt die Anfrage der eigentlichen Ressource. Dazu präsentiert der Client dem Ressourcenserver das Zugriffstoken.
6. Der Ressourcenserver wiederum überprüft die Gültigkeit des Zugriffstokens und gibt bei Korrektheit die geschützte Ressource heraus.

Die Interaktion zwischen Autorisierungsserver und Ressourcenserver, insbesondere die Bekanntgabe und Verifikation des Zugriffstokens, werden im OAuth-Standard nicht weiter spezifiziert. Falls es sich bei Autorisierungsserver und Ressourcenserver nicht um den gleichen Endpunkt handelt, müssen entsprechende

Maßnahmen zum sicheren Austausch der Tokeninformationen sichergestellt werden.

3.6 Autorisierungsnachweise

Die Autorisierung, die der Client vom Ressourcenbesitzer erhält, wird durch einen Autorisierungsnachweis ausgedrückt. Für diesen kennt der OAuth-Standard vier unterschiedliche Varianten, welche nachfolgend aufgelistet sind. Während die ersten beiden Arten sehr häufig Anwendung finden, handelt es sich bei den beiden letzten Varianten um Spezialfälle, für die Dienstanbieter nicht immer Implementierungen bereitstellen.

Autorisierungscode („*Authorization code grant*“): Diese Variante ist insbesondere für vertrauliche Clients vorgesehen und basiert auf einer indirekten Übermittlung eines Autorisierungscodes. Dieser Code wird nach Autorisierung durch den Ressourcenbesitzer vom Autorisierungsserver erzeugt und indirekt über den Ressourcenbesitzer (beispielsweise mittels einer Weiterleitung des Web-Browsers des Ressourcenbesitzers) zum Client geschickt.
In einem weiteren Schritt kann der Client diesen Autorisierungscode in einem zusätzlichen Request-Response-Zyklus beim Autorisierungsserver gegen ein Zugriffstoken (Access Token) austauschen, welches den Zugriff auf die geschützte Ressource zulässt. Ein Vorteil dieser Variante ist, dass die vorliegende Abfolge eine Authentifizierung des Clients und einen sicheren Austausch des Zugriffstokens erlaubt. Alternativen (wie eine unmittelbare Übermittlung des Zugriffstokens über den User Agent zum Client) würden das vertrauliche Zugriffstoken weiteren Akteuren (in diesem Fall dem User Agent des Ressourcenbesitzers) offenbaren.

Implizit („*Implicit grant*“): Der implizite Autorisierungsablauf ist speziell für öffentliche Clients konzipiert. Aufgrund des zuvor bereits angesprochenen Unvermögens solcher Clients zur sicheren Verwahrung von Client-Geheimnissen fällt in dieser Protokollvariante die Client-Authentifizierung weg. Vielmehr wird nach der Autorisierung des Ressourcenbesitzers anstelle eines Autorisierungscodes direkt ein Zugriffstoken ausgestellt und vom Autorisierungsserver an den Client übermittelt. Der Wegfall eines vollständigen Request-Response-Zyklus mag die Laufzeit und Effizienz des Verfahrens beschleunigen – die Effekte auf die Sicherheit des Verfahrens verhalten sich dabei allerdings weniger positiv. Zum einen ermöglicht der Wegfall der Authentifizierung des Clients, dass bösartige Angreifer versuchen sich als legitime Clients auszugeben. Und zum anderen kann das Zugriffstoken während der Übergabe eventuell vom User Agent oder anderen Anwendungen eingesehen werden.

Ressourcenbesitzer-Anmeldung: Ein weiterer Autorisierungsablauf verwendet die Anmeldeinformationen des Ressourcenbesitzers direkt als Autorisierungsnachweis. Folglich werden hierbei die Anmeldeinformationen mit dem Client geteilt. Dieser sendet jene Informationen direkt an den Autorisierungsserver und erhält im Gegenzug ein Zugriffstoken, welches er für nachfolgende Anfragen verwenden kann. Trotzdem hat der Client kurzfristig Zugang zu den Zugangsdaten, weshalb dieser Autorisierungsablauf nur Anwendung finden sollte wenn zwischen dem Ressourcenbesitzer und dem Client ein besonderes Vertrauensverhältnis besteht – beispielsweise weil es sich beim Client um das Betriebssystem oder eine anderweitig außerordentlich privilegierte Anwendung handelt.

Client-Anmeldung: Falls der Client auf eigene, für ihn und nicht für den Ressourcenbesitzer vorhergesehene, Ressourcen zugreifen möchte, ermöglicht der OAuth-Standard auch eine Client-Anmeldung. Ähnlich zum vorher-

gehenden Autorisierungsablauf werden hierbei die vorher vereinbarten Anmeldeinformationen des Clients als Autorisierungsnachweis verwendet.

3.7 Token

Token stellen bei OAuth 2.0 die Grundlage für die Zugriffskontrolle auf geschützte Ressourcen und Bereiche dar. Nach einmaliger Durchführung eines Authentifizierungsverfahrens ermöglichen Token einen mehrfachen Zugriff auf geschützte Informationen. Außerdem sind Token einfach übertragbar, was sie für die Delegation von Zugriffsrechten prädestiniert.

Bevor im Folgenden auf Details zu den Protokollabläufen mit Autorisierungscode und dem mit impliziter Autorisierung eingegangen wird, betrachten wir zunächst die von OAuth verwendeten Token etwas genauer. Zum einen wird das – bereits in Abschnitt 3.1 genannte, aber bisher nicht weiter erläuterte – Zugriffstoken, welches den Zugriff auf Ressourcen legitimiert, erklärt. Zum anderen wird das – bisher noch nicht genannte – Aktualisierungstoken, welches die Anfrage neuer Zugriffstoken ermöglicht, beschrieben.

Zugriffstoken: Das Zugriffstoken (*Access Token*) ist eine Zeichenkette, welche als Nachweis für eine erfolgte Client-Autorisierung diesem Client einen zeitlich begrenzten Zugang auf bestimmte geschützte Ressourcen ermöglicht. Das Zugriffstoken kann dabei entweder eine einfache Referenz auf eine ausgehandelte Autorisierung zwischen Ressourcenbesitzer und Autorisierungsserver darstellen, oder aber auch die Autorisierungsinformationen in nachvollziehbarer Form (also beispielsweise in Form einer kryptografischen Signatur) selbst beinhalten.

Aktualisierungstoken: Da Zugriffstoken in der Regel nur für einen zeitlich begrenzten Zeitraum gültig sind, können zusätzlich zu den Zugriffstoken auch Aktualisierungstoken (*Refresh Token*) ausgestellt werden. Diese sind normalerweise für einen weitaus längeren Zeitraum gültig und ermöglichen die Ausstellung neuer Zugriffstokens mit gleichem oder verkürztem Gültigkeitszeitraum sowie gleichem oder beschränktem Zugriffsbereich. Die Ausstellung – und gegebenenfalls der Widerruf – solcher Tokens obliegen dem Autorisierungsserver. Eine Ausstellung ist optional. Wenn eine solche aber stattfindet, erfolgt diese mit der Ausstellung und Weitergabe des Zugriffstokens (vgl. Schritt 4 in Abbildung 2). Analog zu den Zugriffstoken handelt es sich auch bei den Aktualisierungstoken um eine Zeichenkette, die auf eine erfolgte Client-Autorisierung verweist. Durch die erweiterte Gültigkeit des Aktualisierungstokens erhöhen sich auch die Anforderungen hinsichtlich der sicheren Weitergabe und Speicherung.

Die Ausstellung, Weitergabe und Verwendung eines Aktualisierungstokens ist in Abbildung 3 dargestellt und wird im Folgenden näher erläutert:

1. Zu Beginn fragt der Client beim Autorisierungsserver ein Zugriffstoken an und präsentiert dabei einen gültigen Autorisierungsnachweis.
2. Nachdem der Autorisierungsserver den Client authentifiziert und dessen Autorisierungsnachweis verifiziert hat, stellt er dem Client ein Zugriffstoken sowie ein Aktualisierungstoken aus.
3. Daraufhin fragt der Client beim Ressourcenserver eine geschützte Ressource an.
4. Der Ressourcenserver wiederum verifiziert die Gültigkeit des Zugriffstokens und liefert die geschützte Ressource aus.

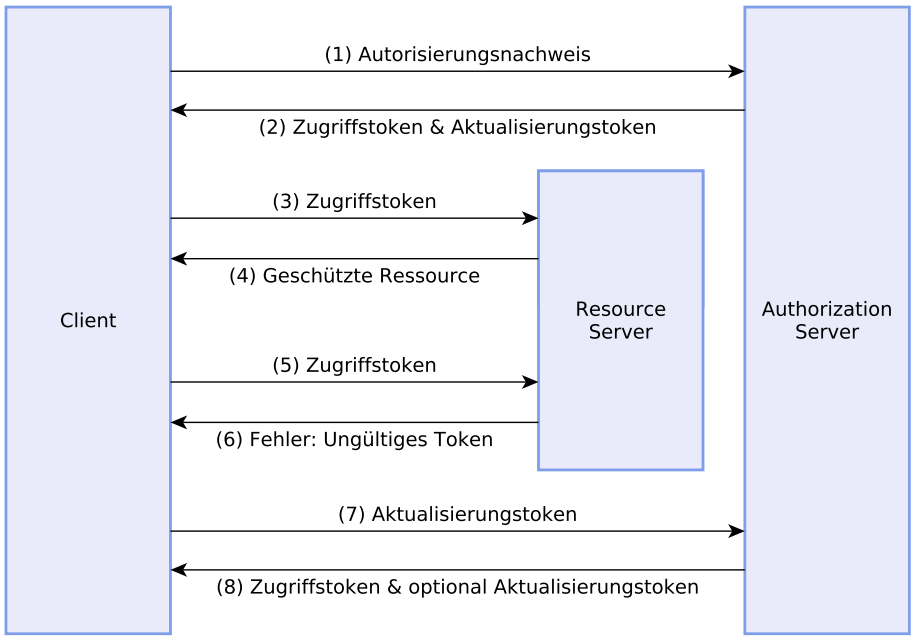


Abb. 3: Anfrage und Verwendung von Aktualisierungstoken bei OAuth 2.0

5. Die Schritte 3. und 4. können so lange wiederholt werden, wie das Zugriffstoken gültig ist.
6. Sobald das Zugriffstoken seine Gültigkeit verloren hat, antwortet der Ressourcenserver auf die Anfrage mit einer Fehlermeldung, welche auf die Ungültigkeit des Tokens hinweist.
7. Der Client erbittet beim Autorisierungsserver ein neues Zugriffstoken. Dazu authentisiert sich der Client gegebenenfalls erneut gegenüber dem Autorisierungsserver und präsentiert sein Aktualisierungstoken.
8. Der Autorisierungsserver authentifiziert den Client und verifiziert dessen Aktualisierungstoken. Wenn diese Prüfung erfolgreich verläuft, stellt er diesem ein neues Zugriffstoken aus (sowie gegebenenfalls auch ein neues Aktualisierungstoken).

3.8 Authorization Code Grant

Um die internen Vorgänge bei der ersten Variante mit Autorisierungscode für vertrauliche Clients besser nachvollziehen zu können, betrachten wir in diesem Abschnitt die einzelnen Schritte und Nachrichten, welche bei dieser Protokollvariante durchgeführt bzw. ausgetauscht werden und von Abbildung 4 aufgezeigt werden.

Als Beispiel für einen vertraulichen Client eignet sich die bereits beschriebene Online-Druckerei. Es wird davon ausgegangen, dass der Endbenutzer bereits begonnen hat, mit der Online-Druckerei z. B. mittels eines Web-Browsers zu interagieren, und nun die Online-Druckerei auf das Cloud-Fotoalbum zugreifen möchte. An dieser Stelle beginnt die dargestellte Interaktion (d. h. der initiale Zugriff des Endbenutzers (Ressourcenbesitzer) auf die Online-Druckerei (Client-Dienst) ist nicht dargestellt).

(1) Autorisierungsanfrage: Sobald der Client auf eine geschützte Ressource zugreifen möchte, startet dieser den Autorisierungsprozess, indem er in der bestehenden Session den Browser (*User Agent*) durch einen HTTP Redirect

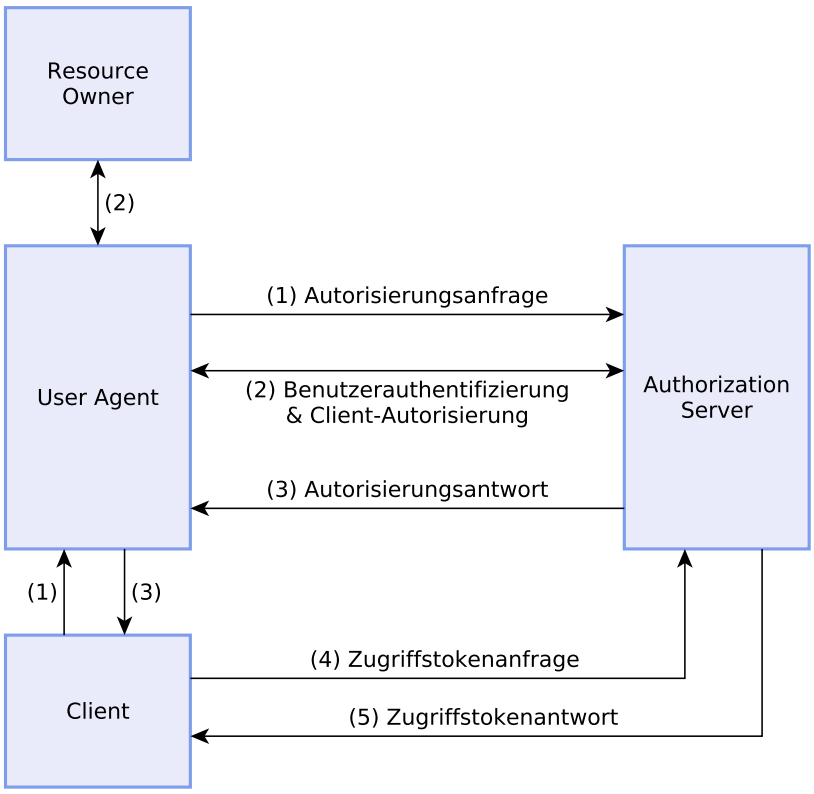


Abb. 4: OAuth 2.0: Protokollablauf in der Variante „Authorization Code Grant“

an den Autorisierungsserver weiterleitet. Der Client übermittelt dem Autorisierungsserver in dieser Anfrage den Namen der Protokollvariante, die zur Autorisierung angewandt werden soll, und seinen Client-Identifier. Darüber hinaus benennt die Anfrage den oder die Bereiche, für die ein später ausgestelltes Zugriffstoken gültig sein soll, sowie eine Weiterleitungs-URI, an die der Autorisierungsserver den User Agent anschließend weiterleiten soll.

B

Beispiel 1: OAuth-2.0-Autorisierungsanfrage des Clients, vom User Agent an den Autorisierungsserver geschickt

```
GET /authorize?response_type=code&client_id=t7AaMt3s&state
    =foo&redirect_uri=https%3A%2F%2Fclient%2Eoauth%2Eopen
    %2Dc3s%2Ede%2Fcb HTTP/1.1
Host: server.oauth.open-c3s.de
```

Die verfügbaren Parameter der Autorisierungsanfrage lauten wie folgt (alle nicht als optional oder empfohlen gekennzeichneten Parameter sind immer erforderlich):

response_type: Gibt die Art des Autorisierungsablaufs an und ist im Falle des Authorization Code Grants immer `code`.

client_id: Identifiziert den Client eindeutig. Dieser Identifier wird während der Client-Registrierung vergeben.

redirect_uri: *[optional]* Enthält eine absolute URI, über welche der User Agent des Ressourcenbesitzers nach Ablauf des Protokollschritts wie-

der an den Client geleitet wird. Die URI wird in der Regel während der Client-Registrierung festgelegt.

scope: *[optional]* Definiert die Aktionen und Ressourcen, welche durch den Einsatz des Zugriffstoken ausgeführt und abgefragt werden können. Diese Berechtigungen werden als einzelne Worte, welche mit Leerzeichen aneinander gelistet werden können, repräsentiert. Die möglichen Werte sind dabei implementierungsabhängig und müssen vom Autorisierungsserver spezifiziert und dokumentiert werden.

state: *[empfohlen]* Eine beliebige nicht einfach zu erratende Zeichenkette, welche später vom Autorisierungsserver reflektiert wird. Somit kann der Client sicher sein, dass die Antwort des Autorisierungsservers auf die eigene Anfrage erfolgt und nicht auf irgendeine andere (möglicherweise gefälschte) Autorisierungsanfrage. Da mit Hilfe dieses Parameters Cross-Site-Request-Forgery-Angriffe erschwert werden, wird die Verwendung eindringlich empfohlen.

(2) Benutzerauthentifizierung & Client-Autorisierung: Der Ressourcenbesitzer authentisiert sich mittels des User Agent gegenüber dem Autorisierungsserver und räumt dem Client ein Nutzungsrecht über die geforderten Ressourcen ein. Der genaue Ablauf der Authentifizierung obliegt dem Autorisierungsserver und ist im OAuth-2.0-Standard nicht näher vorgegeben.

(3) Autorisierungsantwort: Nach erfolgreicher Authentifizierung und Autorisierung stellt der Autorisierungsserver einen Autorisierungscode aus. Dieser wird in einer Redirect-Antwort an den User-Agent in die Weiterleitungs-URI integriert und dadurch mittels des User Agents an den Client übertragen. Im Fehlerfall informiert der Autorisierungsserver den Client entsprechend.

Beispiel 2: OAuth-2.0-Antwort des Autorisierungsservers an den User Agent in der Variante „Authorization Code Grant“

```
HTTP/1.1 302 Found
Location: https://client.oauth.open-c3s.de/cb?code=
    Tet67xl4abqxGTrfNv6ieR&state=foo
```

B

Die verfügbaren Parametern lauten dabei:

code: Der Autorisierungscode, welcher vom Autorisierungsserver ausgestellt wurde. Der Autorisierungsserver sollte diesen Code intern an einen Client-Identifier und eine Weiterleitungs-URI binden, damit abgefangene Autorisierungscodes nicht unrechtmäßig verwendet werden können. Weiterhin darf der Code nur für eine zeitlich stark begrenzte Zeit gültig sein (empfohlen sind maximal 10 Minuten) und vom zugewiesenen Client nur einmal verwendet werden. Sobald der Versuch einer Mehrfachnutzung festgestellt wird, muss die Anfrage abgewiesen werden und – falls möglich – alle auf Basis des verwendeten Autorisierungscodes ausgestellten Token widerrufen werden.

state: Falls die Autorisierungsanfrage den Parameter `state` enthalten hat, wird hier der gleiche Wert übermittelt.

(4) Zugriffstokenanfrage: Nachdem der Client den Autorisierungscode aus der Weiterleitungs-URI extrahiert hat überträgt er diesen zusammen mit einer weiteren Weiterleitungs-URI an den Autorisierungsserver, um ein Zugriffstoken anzufragen. Da dieser Autorisierungsablauf für vertrauenswürdige Clients vorgesehen ist, identifizieren sich jene in diesem Schritt gegenüber dem

Autorisierungsserver durch die Angabe des Client-Identifiers und authentisieren sich mit dem vereinbarten Geheimnis. Die Parameter `client_id` und `client_secret` können dazu vorzugsweise als Benutzername und Passwort mit der HTTP Basic Authentifizierung übertragen werden oder alternativ in URL-Kodierung innerhalb der Anfrageparameter.

B

Beispiel 3: Anfrage des Clients nach dem Zugriffstoken bei OAuth 2.0

```
POST /token HTTP/1.1
Host: server.oauth.open-c3s.de
Authorization: Basic dDdBYU10M3M6cGFzc3dvcmQ=
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=Tet67xl4abqxGTrfNv6ieR&
    redirect_uri=https%3A%2F%2Fclient%2Eoauth%2Eopen%2Dc3s
    %2Ede%2Fcb
```

Die verfügbaren Parameter zur Anfrage eines Zugriffstokens lauten wie folgt:

grant_type: Gibt die Art des Berechtigungsnachweises an und ist im Falle der Zugriffstokenanfrage immer `authorization_code`.

code: Der Autorisierungscode, wie er vom Autorisierungsserver im vorherigen Schritt ausgestellt wurde.

client_id: Falls nicht anderweitig übermittelt: Identifiziert den Client eindeutig.

client_secret: Falls nicht anderweitig übermittelt: das ausgehandelte Client-Geheimnis zur Authentifizierung des Clients.

redirect_uri: Enthält den gleichen Wert wie in der Autorisierungsanfrage.

(5) Zugriffstokenantwort: Der Autorisierungsserver authentifiziert den Client und verifiziert den Autorisierungscode. Dabei sollte auch sichergestellt werden, dass der ausgestellte Autorisierungscode dem anfragenden Client zugewiesen wurde und gültig ist. Falls in der Autorisierungsanfrage der Parameter `redirect_uri` vorlag, wird zusätzlich die Existenz sowie die Übereinstimmung dessen Wertes überprüft. Nach erfolgreicher Überprüfung stellt der Autorisierungsserver ein Zugriffstoken und gegebenenfalls ein Aktualisierungstoken aus und sendet diese zusammen mit dem Gül-

tigkeitsbereich sowie dem Ablaufdatum des Zugriffstokens zurück an den Client. Die Angabe dieser Inhalte erfolgt dabei in JSON-Notation.

Beispiel 4: Antwort des OAuth-2.0-Autorisierungsservers mit Zugriffstoken

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

 {
   "access_token":"kL_3.M7m-4.J1Mf",
   "token_type":"Bearer",
   "expires_in":3600,
   "refresh_token":"2Govn3ZkE0rkOxeic3WpIA",
 }
```

B

Die verfügbaren Parameter lauten dabei:

access_token: Das Zugriffstoken, welches vom Autorisierungsserver ausgestellt wurde.

token_type: Der Typ des Zugriffstokens. RFC 6750 (Jones und Hardt, 2012) definiert dazu das Bearer Token als Standardzugriffstoken.

expires_in: *[empfohlen]* Das Ablaufdatum des Zugriffstokens in Sekunden.

refresh_token: *[optional]* Das Aktualisierungstoken, welches in der Regel eine weitaus längere Gültigkeitsdauer als das Zugriffstoken besitzt und nach dessen Ablauf für die Generierung neuer Zugriffstoken eingesetzt werden kann.

scope: *[optional]* Die Aktionen und Ressourcen, welche mit Hilfe des ausgestellten Zugriffstokens ausgeführt beziehungsweise abgefragt werden können.

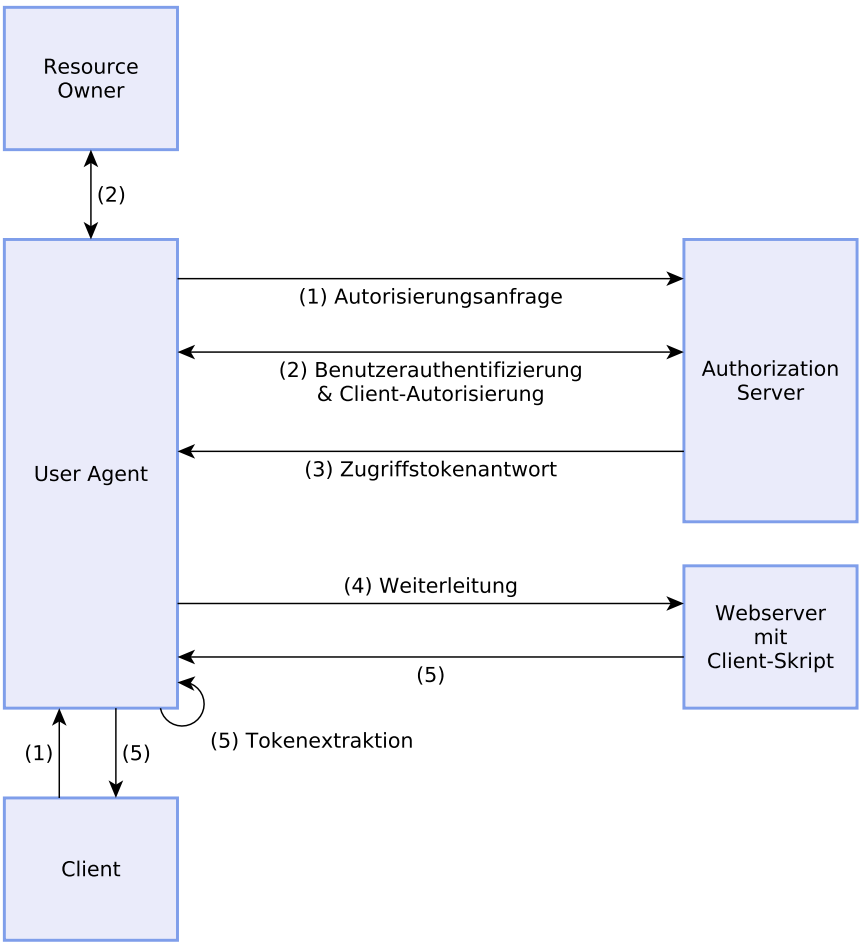
3.9 Implicit Grant

Der implizite Autorisierungsablauf trägt seinen Namen, weil er Zugriffstoken nicht als Antwort auf eine Zugriffstokenanfrage – legitimiert durch einen Autorisierungscode – ausstellt, sondern implizit als Antwort auf die Autorisierungsanfrage. Wie bereits erwähnt, wurde diese Protokollvariante für öffentliche Clients eingeführt und verzichtet im Verlauf auf die Client-Authentifizierung.

Beispielsweise kann es sich hier beim Client um eine JavaScript-Anwendung handeln, die innerhalb des Web-Browsers des Nutzers ausgeführt wird. Abbildung 5 zeigt die nötigen Nachrichten und Schritte, welche nachfolgend erklärt werden. Die ersten Schritte ähneln dabei stark dem gerade vorgestellten Authorization Code Grant.

(1) Autorisierungsanfrage: Auch in dieser Variante startet der Client den Autorisierungsprozess, indem er den Browser (User Agent) an den Autorisierungsserver weiterleitet und dabei dem Autorisierungsserver den Namen der Protokollvariante, die zur Autorisierung angewandt werden soll, und

Abb. 5: OAuth 2.0: Implicit Grant



seinen Client-Identifier übermittelt. Diese Anfrage kann außerdem noch den oder die Gültigkeitsbereiche sowie eine Weiterleitungs-URI enthalten.

B

Beispiel 5: OAuth-2.0-Autorisierungsanfrage des Clients in der Variante „Implicit Grant“

```
GET /authorize?response_type=token&client_id=t7AaMt3s&
    state=foo&redirect_uri=https%3A%2F%2Fclient%2Eoauth%2
    Eopen%2Dc3s%2Ede%2Fcb HTTP/1.1
Host: server.oauth.open-c3s.de
```

Die verfügbaren Parameter der Autorisierungsanfrage entsprechen dabei dem Ablauf beim „Authorization Code Grant“. Der einzige Unterschied ist der übermittelte Typ, der hier nun `token` statt `code` lautet.

(2) Benutzerauthentifizierung & Client-Autorisierung: Der Ressourcenbesitzer authentisiert sich mittels des User Agents gegenüber dem Autorisierungsserver und räumt dem Client ein Nutzungsrecht über die geforderten Ressourcen ein.

(3) Zugriffstokenantwort: Nach erfolgreicher Authentifizierung und Autorisierung stellt der Autorisierungsserver direkt ein Zugriffstoken aus. Dieses wird als Fragment in die Weiterleitungs-URI integriert und an den User Agent übertragen. Ein Aktualisierungstoken darf in diesem Autorisierungs-

ablauf nicht ausgestellt werden. Im Fehlerfall informiert der Autorisierungsserver den Client entsprechend.

Beispiel 6: OAuth-2.0-Antwort des Autorisierungsservers an den User Agent in der Variante „Implicit Grant“

```
HTTP/1.1 302 Found
Location: https://client.oauth.open-c3s.de/cb?#
    access_token=kL_3.M7m-4.J1Mfstate=foo&token_type=
    Bearer&expires_in=3600
```

B

Die verfügbaren Parametern lauten dabei:

access_token: Das Zugriffstoken, welches vom Autorisierungsserver ausgestellt wurde.

token_type: Der Typ des Zugriffstokens. RFC 6750 definiert dazu das Bearer Token als Standardzugriffstoken.

expires_in: *[empfohlen]* Die Gültigkeitsdauer des Zugriffstokens in Sekunden.

scope: *[optional]* Die Aktionen und Ressourcen, welche mit Hilfe des ausgestellten Zugriffstokens ausgeführt beziehungsweise abgefragt werden können.

state: Enthält den gleichen Wert wie in der Autorisierungsanfrage.

(4) Weiterleitung: Der User Agent folgt der Weiterleitung, indem er einen neuen HTTP-Request an die angegebene URI stellt. Die URI in der Anfrage enthält dabei nicht mehr den Fragmentteil, da dieser vor dem Abschicken vom User Agent entfernt wird. Die Anfrage verweist auf einen Webserver, welcher ein Client-Skript ausliefert.

(5) Tokenextraktion: Der User Agent führt das übermittelte Skript aus. Das Skript hat daher Zugriff auf bestimmte Eigenschaften des User Agents, darunter auch der Fragmentteil der URI, welcher im Browser immer noch verfügbar ist. Das Skript extrahiert daraufhin aus dem Fragment das Zugriffstoken und übergibt es dem Client.

3.10 Implementierungen

Die OAuth-2.0-Spezifikation regelt nicht alle Details des Protokolls und ermöglicht in vielen Bereichen eigene Erweiterungen. Für Dienstanbieter, die selbst Ressourcen und APIs anderer Dienste als Clients nutzen wollen, ist es daher unerlässlich sich über die genauen Eigenheiten der Implementierung – beispielsweise hinsichtlich der Client-Registrierung und -Authentifizierung oder der verwendeten Tokenmechanismen – zu informieren.

4 OpenID Connect

Das *OpenID Connect* Protokoll bietet als Erweiterung des OAuth-Standards eine einfache und standardisierte Identitätsverwaltung, von der sowohl die Dienstanbieter als auch die Dienstnutzer profitieren. Nutzer können ihre Identitäten zentral beim Identitätsprovider pflegen und dabei selbst genau bestimmen welche Informationen sie mit welchen Dienstanbietern teilen möchten. Und für Dienstanbieter bedeutet die Auslagerung von Teilen der Benutzerverwaltung sowie der Benutzerauthentifizierung eine Reduktion der Komplexität sowie des Aufwands. Im

Folgenden wird das Protokoll erst grundlegend vorgestellt, bevor die Details des Protokollablaufs erörtert und mögliche Sicherheitsprobleme skizziert werden.

4.1 Hintergrund

OpenID Connect ist der direkte Nachfolger des *OpenID*-Standards. Die Anfänge von OpenID reichen zurück in das Jahr 2005, und damit in eine Zeit in der mit wachsender Popularität des Internets zahlreiche neue Dienste entstanden, deren Nutzung zumeist auch ein eigenes Benutzerkonto erforderte. Für Nutzer solcher Dienste bedeutete die hohe Zahl an Benutzerkonten nicht nur einen erhöhten Verwaltungsaufwand, sondern wirkte sich auch negativ auf die Sicherheit der einzelnen Konten aus.

Letzteres liegt darin begründet, dass zur Authentifizierung solcher Konten im Normalfall Passwörter herangezogen werden, welche wiederum komplex genug und gleichzeitig einzigartig sein sollten, damit diese nicht einfach erraten oder im Falle der Offenlegung für weitere Dienste missbraucht werden können. Mit steigender Zahl der Passwörter neigen Internetnutzer allerdings dazu, die Komplexität der einzelnen Passwörter zu reduzieren oder die gleichen Passwörter für mehrere Konten zu verwenden. Zahlreiche Entwickler, welche dem daraus resultierenden Verwaltungsaufwand und den damit verbundenen negativen Folgen für die Sicherheit entgegen treten wollten, begannen zu dieser Zeit mit der Entwicklung von verteilten Identitätsmanagementsystemen. Nachdem eine Vielzahl von Insellösungen existierten, bündelten einige dieser Entwickler ihre Bemühungen und brachten die erste Version von OpenID hervor (Recordon und Fitzpatrick, 2006). Als immer weitere Partner für das Projekt gewonnen wurden und die Weiterentwicklung des Protokolls mit zahlreichen Erweiterungen vorangetrieben werden konnte, wurde im Juni 2007 die OpenID Foundation gegründet und im Dezember 2007 die zweite Version der OpenID Spezifikation publiziert.

Der OpenID-Standard erlaubte es Internetnutzern Identitäten in Form von Benutzerprofilen zentral bei Identitätsprovidern zu registrieren. Diese Profile wurden mittels einer eindeutigen URL – auch selbst als OpenID bezeichnet – referenziert und konnten applikationsübergreifend für die Identifizierung und Authentifizierung bei unterschiedlichen Webanwendungen genutzt werden. Dienstanbieter konnten die OpenID einfach einem Benutzerkonto zuweisen, um die dienst- und kontenspezifischen Attribute zu speichern. Für einige Standardattribute wie beispielsweise Alter, Geschlecht oder Adresse regelte eine Erweiterung den einfachen Austausch zwischen Identitätsprovider und Dienstanbieter. Nutzer eines OpenID-Profils waren somit in der Lage ein solches Konto direkt für die Anmeldung bei OpenID-kompatiblen Diensten zu verwenden. Dabei wurde der Nutzer vom Dienstanbieter an den zuständigen Identitätsprovider verwiesen, welcher die erforderliche Authentifizierung durchführte. Im Falle einer erfolgreichen Authentifizierung leitete der Identitätsprovider den Nutzer wieder zurück an den Dienst und bestätigte diesem die Identität des Nutzers.

Obwohl einige bedeutsame Dienstanbieter – wie Google und Yahoo – den OpenID-Standard schnell und umfassend umsetzten, verwehrten andere Anbieter dem Standard ihre Akzeptanz. Das mitgliederstärkste soziale Netzwerk Facebook brachte mit Facebook Connect einen eigenen auf OAuth-basierenden Konkurrenten ins Rennen. Aufgrund der einfacheren Handhabung von OAuth und der weiten Verbreitung von Facebook adaptierten viele Dienstanbieter die Konkurrenzlösung als Authentifizierungsprotokoll.

Hierbei muss allerdings beachtet werden, dass OAuth selbst gar keine echte Authentifizierung leisten kann – siehe dazu auch Bradley (2012). Wie anfangs definiert

dient die Authentifizierung der Verifikation einer Identität. OAuth hingegen ist ein Protokoll zur API-Autorisierung, also eine Delegation der Berechtigung zum Zugriff auf Ressourcen. Wenn ein Dienst die Berechtigung zum Zugriff erhält ist es zwar denkbar, daraus zu schließen, dass der zu einer Identität gehörende Benutzer dieser Berechtigung zugestimmt hat um ihn dadurch implizit zu authentifizieren. Ohne geeignete Sicherheitsmaßnahmen können bei einem solchen Vorgehen aber leicht Schwachstellen entstehen. Verwendet beispielsweise ein Benutzer dieses Verfahren, um sich bei einem Dienst X anzumelden (der Dienst erhält dann ein Access Token), dann könnte der Dienst X dieses Token missbrauchen und sich gegenüber einem Dienst Y als der Benutzer ausgehen (dies ist insbesondere in der *impliziten* Variante von OAuth einfach möglich).

Um derartige Schwachstellen zu vermeiden, nutzen auf OAuth basierende Authentifizierungsverfahren daher zusätzliche Schutzmechanismen, die einerseits auf eine erfolgte Authentifizierung verweisen und andererseits die Nutzung der daraus resultierenden Informationen an einen bestimmten Client binden. Nachdem diese Vorgehensweise in auf OAuth aufbauenden, proprietären Verfahren eine wachsende Verbreitung fand und teilweise das ursprüngliche OpenID-Verfahren ablösen konnte, ist nun *OpenID Connect* ein Versuch, mit einem einfachen und leistungsfähigen Protokoll die ursprüngliche Idee eines unabhängigen internetweiten Identitätsmanagement weiter standardisiert voranzutreiben. Ende 2014 konnte durch die Veröffentlichung des OpenID Connect-Standards ein OAuth-basiertes Autorisierungsprotokoll als Nachfolger von OpenID etabliert werden (Sakimura et al., 2014a).

4.2 Konzept

Wie die verschiedenen Vorgänger-Versionen von OpenID ermöglicht OpenID Connect, dass Client-Anwendungen die Identitäten von Internetnutzern sowie bestimmte Attribute bei zentralen Identitätsprovidern abfragen und verifizieren können. OpenID Connect ist dabei – wie gerade festgestellt – als Erweiterung des OAuth 2.0-Protokolls konzipiert. Die beteiligten Akteure und Abläufe ähneln daher stark dem bereits vorgestellten Protokoll.

Ein Internetnutzer kann bei einem beliebigen Identitätsprovider – beispielsweise *Google* oder *PayPal* – eine Identität in Form eines Benutzerprofils registrieren. Zur Nutzung eines teilnehmenden Dienstes – wie zum Beispiel einem Online-Blog – kann sich der Nutzer nun mittels des Identitätsproviders anmelden. Der Nutzer authentisiert sich dabei mit dem ausgemachten Identitätsprovider. Wenn die Authentifizierung erfolgreich verläuft, stellt dieser ein ID-Token aus und sendet dieses zurück an den Dienstanbieter. Dieser wiederum kann den Nutzer mit dieser Information identifizieren und dessen Sitzung autorisieren.

In der Praxis geschieht dieser Ablauf oft über einen vom Dienst fest vorgegebenen Identitätsprovider und statisch konfigurierten Adressen z. B. für den Autorisierungsserver. Es gibt mit *OpenID Connect Discovery* aber auch eine Erweiterung, mit deren Hilfe sich die erforderlichen Angaben zum Autorisierungserver automatisch ermitteln lassen. Details dazu lassen sich in der offiziellen Spezifikation, *OpenID Connect Discovery 1.0*, nachlesen (Sakimura et al., 2014b).

4.3 Rollen

Die OpenID Connect-Spezifikation definiert die nachfolgenden Rollen für die beteiligten Entitäten. Aufgrund seiner Historie gibt es für OpenID Connect zwei

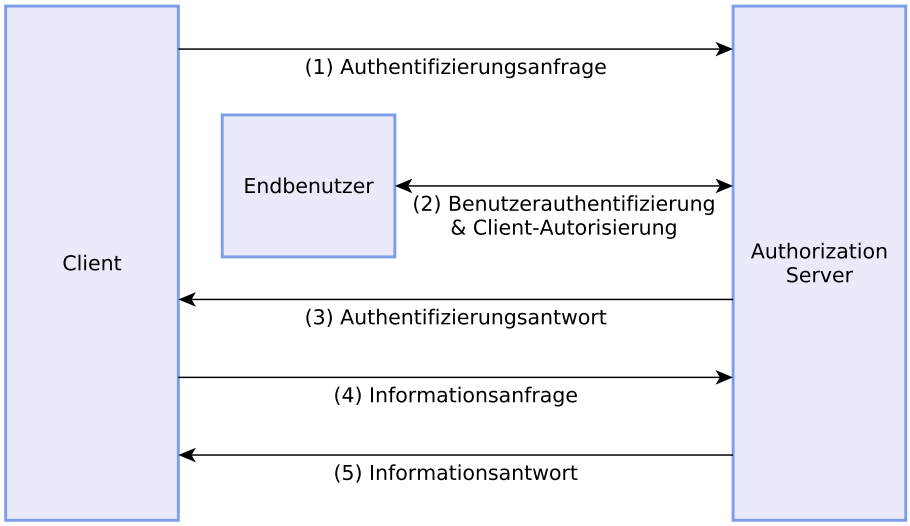


Abb. 6: Abstrakter Protokollfluss des OpenID Connect-Protokolls

unterschiedliche Terminologien. Im Folgenden wird die neuere und sich am OAuth-Standard orientierende Terminologie verwendet. In Klammern ergänzt findet sich auch die ursprüngliche OpenID-Terminologie.

Endbenutzer: Der Endbenutzer besitzt ein Benutzerprofil bei einem Identitätsprovider und möchte dieses zur Identifikation und Authentisierung bei verschiedenen Diensten nutzen.

Client (*Relying Party*): Als Clients werden die Dienste und Anwendungen bezeichnet, welche sich die Identität der Endbenutzer von einem ausgelagerten Identitätsprovider bestätigen lassen und dort gegebenenfalls weitere Informationen über die Benutzer abfragen. Da OpenID Connect auf OAuth aufbaut sind mögliche Applikation ebenfalls nicht auf bestimmte Implementierungen oder Geräte beschränkt.

Autorisierungsserver (*OpenID Provider*): Der eigentliche Identitätsprovider ist ein Autorisierungsserver nach dem OAuth-Standard, welcher die Authentifizierung des Endbenutzers übernimmt und den Clients diese Authentifizierung bestätigt.

Userinfo-Endpunkt: Beim Userinfo-Endpunkt handelt es sich um einen OAuth-Token-Endpunkt, bei dem der Client mit einem Zugriffstoken Informationen abfragen kann. In Abbildung 6 ist der Userinfo-Endpunkt Teil des Authorization Servers.

4.4 Interaktion

Abbildung 6 zeigt die Interaktion der gerade vorgestellten Akteure während einer Benutzerauthentifizierung mit anschließender Abfrage von weiterführenden Profilinformationen. Die einzelnen Schritte sind dabei:

(1) Authentifizierungsanfrage: Zur Authentifizierung des Endbenutzers stellt der Client eine Anfrage an den zuständigen Autorisierungsserver.

(2) Benutzerauthentifizierung & Client-Autorisierung: Der Endbenutzer authentisiert sich gegenüber dem Autorisierungsserver und autorisiert den Client zur Abfrage weiterer Profilinformationen. Der genaue Ablauf der Authentifizierung obliegt dem Autorisierungsserver und ist nicht näher vorgegeben.

(3) Authentifizierungsantwort: Nach erfolgreicher Authentifizierung und Autorisierung stellt der Autorisierungsserver ein ID-Token, welches die erfolgte Authentifizierung bestätigt, sowie gegebenenfalls ein Zugriffstoken, welches den Besitzer zur Abfrage der festgelegten Profilinformationen autorisiert, aus und sendet diese zurück an den Client.

(4) Informationsanfrage: Wenn ein Zugriffstoken ausgestellt wurde, schickt der Client dieses Token im nächsten Schritt an den Token-Endpunkt, um die damit verknüpften Information abzufragen.

(5) Informationsantwort: Der Token-Endpunkt verifiziert das Zugriffstoken und sendet die angeforderten Informationen zurück an den Client. Der Autorisierungsserver kann Informationen („*claims*“) über den Endbenutzer entweder innerhalb des ID-Tokens in Schritt (3) mitteilen oder innerhalb eines Userinfo-Tokens, das in der Antwort in Schritt (5) übermittelt wird.

4.5 ID-Token

Das bereits angesprochene ID-Token stellt die primäre Erweiterung zum OAuth-Protokoll dar. Es bestätigt die erfolgte Authentifizierung und enthält Details über die Benutzeranmeldung. Dazu zählen neben dem Aussteller des Tokens und dem Subjekt, das authentifiziert wurde, sowie dem Adressaten der Authentifizierung weitere Informationen wie Ausstellungs- und Verfallsdaten. Technisch ist das ID-Token als JSON Web Token (JWT) realisiert – ein Standard, welcher selbst im RFC 7519 spezifiziert ist (Jones et al., 2015). Ein Beispieltoken ist unten aufgeführt. Daneben nutzt OpenID Connect auch die beiden bereits bekannten OAuth-Token – das Zugriffstoken zur Abfrage weiterer Informationen sowie das Erneuerungstoken zur Ausstellung neuer Zugriffstoken.

Beispiel 7: JSON-Datenstruktur des ID-Token

```
{
  "iss": "https://openid.open-c3s.de",
  "sub": "42232003",
  "aud": "qt3Rkhds6B",
  "nonce": "75RT-am_r5p6",
  "exp": 1484075381,
  "iat": 1484074381,
  "auth_time": 1484074380
}
```

B

Die verfügbaren Parametern für ein ID-Token umfassen unter anderem:

iss: Der ausstellende Autorisierungsserver des ID-Tokens. Der Aussteller wird mittels einer URL eindeutig identifiziert. Diese URL enthält ein Schema und den Host sowie optional eine Portnummer – allerdings keine Query- oder Fragment-Bestandteile.

sub: Der Endbenutzer für den die Authentifizierung erfolgte – oder auch das Subjekt der Authentifizierung. Das Subjekt wird hier mittels eines Bezeichners referenziert, welcher vom Autorisierungsserver für den Benutzer lokal eindeutig ausgestellt wurde.

aud: Die Adressaten des ID-Tokens. Es muss den Bezeichner des anfragenden Clients enthalten, kann daneben aber auch noch weitere Adressaten beinhalten.

exp: Das Verfallsdatum des ID-Tokens. Nach diesem Zeitpunkt darf das Token nicht mehr weiter verwendet werden, sondern muss verworfen werden. Verfallsdaten sollten kurz gewählt werden und nur wenige Minuten in der Zukunft liegen. Der Wert wird dabei als Unix-Zeitstempel angegeben.

iat: Das Ausstellungsdatum des Tokens. Die Angabe des Werts erfolgt als Unix Zeitstempel.

auth_time: *[optional]* Der Zeitpunkt an dem die Authentifizierung erfolgte. Der Wert wird dabei als Unix-Zeitstempel repräsentiert.

nonce: Der Nonce-Wert aus der Authentifizierungsanfrage. Der Wert ist an eine Client-Sitzung sowie an das ID-Token gebunden und verhindert Replay-Angriffe.

amr: *[optional]* Eine Auflistung der Verfahren, die bei zur Authentifizierung angewendet wurden. Die möglichen Werte sind nicht weiter spezifiziert und müssen von den beteiligten Akteuren vorher ausgehandelt werden.

4.6 Profilinformationen

Mit OpenID Connect können standardisierte Profilinformationen – so genannte *Claims* über den Endbenutzer – ermittelt werden. Enthält die Antwort des Autorisierungsservers ein Zugriffstoken, so lassen sich diese Informationen von einem UserInfo-Endpunkt gegen Vorlage des Zugriffstokens abfragen. Ansonsten, falls durch die vorliegende Protokollvariante neben dem ID-Token kein zusätzliches Zugriffstoken ausgestellt wird, werden die Profilinformationen direkt im ID-Token vom Autorisierungsserver zurückgegeben.

Im `scope`-Parameter der Autorisierungsanfrage kann eine Auswahl an Standardinformationen angefordert werden (`profile`, `email`, `address` und `phone`), zu denen folgende Auskünfte erteilt werden:

profile: Zugriff auf die Standardinformationen eines Benutzerprofils. Diese Angaben umfassen den Namen (`name`), den Nachnamen (`family_name`), den Rufnamen (`given_name`), den oder die Zweitnamen (`middle_name`), den Alias (`nickname`), den bevorzugten Benutzernamen (`preferred_username`), die URL zum Benutzerprofil (`profile`), die URL zum Benutzerbild (`picture`), die URL zur Webseite des Benutzers (`website`), das Geschlecht (`gender`), das Geburtsdatum (`birthdate`), die Zeitzone des Nutzers (`zoneinfo`), das Gebietsschema als BCP47 Sprachcode (`locale`) sowie das letzte Aktualisierungsdatum (`updated_at`).

email: Information über die E-Mail-Adresse des Nutzers (`email`) und die Information, ob diese verifiziert wurde (`email_verified`).

address: Die postalische Adresse des Nutzers (`address`), repräsentiert durch mehrere Attribute in einem JSON-Objekt.

phone: Die Telefonnummer des Nutzers (`phone_number`) und die Information, ob diese verifiziert wurde (`phone_number_verified`).

Diese Profilinformationen können um weitere Informationen mit eigenen Bezeichnern erweitert werden. Erweiterungs-*Claims* werden dann aber nicht im `scope`-Parameter sondern im eigens dafür eingeführten `claims`-Parameter angefordert. Unbekannte *Claims* sollten stets ignoriert werden. Um gleichzeitig mehrere dieser Informationen anzufragen, können die Bezeichner durch Leerzeichen getrennt im Parameter `scope` aneinandergereiht werden (`scope=openid profile email phone`).

4.7 Protokollabläufe

Der OpenID Standard kennt drei Protokollabläufe: Den Ablauf mit Autorisierungscode (*Authorization Code Flow*), den Ablauf mit implizierter Autorisierung (*Implicit Flow*) sowie den hybriden Ablauf (*Hybrid Flow*). Die ersten beiden Varianten korrespondieren dabei mit den in Abschnitt 3.8 (Authorization Code Grant) und Abschnitt 3.9 (Implicit Grant) beschriebenen OAuth-2.0-Varianten.

Die wesentlichen Merkmale und Unterschiede der einzelnen Protokollabläufe sind in Tabelle 1 skizziert. Während im *Implicit Flow* alle Tokens vom Autorisierungsserver direkt ausgestellt werden, ist im *Authorization Code Flow* der Token-Endpunkt exklusiv für die Ausgabe und Verarbeitung der Token verantwortlich. Darüber hinaus ist der *Authorization Code Flow* der einzige Ablauf, bei dem sich die Token nicht innerhalb des Zugriffsbereichs des Browsers des Benutzers befinden, da die Übertragung der Token direkt zwischen Autorisierungsserver und Client stattfindet. Dem gegenüber steht die vereinfachte Kommunikationsstruktur der Variante mit implizierter Autorisierung, welche nur einen Anfrage-Antwort-Zyklus benötigt. Weitere Vorteile, welche sowohl im *Authorization Code Flow* als auch im *hybriden Autorisierungsablauf* verfügbar sind, sind zum einen die Option zur Authentifizierung des Clients sowie zum anderen die Möglichkeit zur Ausstellung von Erneuerungstoken.

Merkmal	**Auth. Code Flow**	**Implicit Flow**	**Hybrid Flow**
Alle Token werden vom Autorisierungsserver ausgestellt	✗	✓	✗
Alle Token werden vom Token-Endpunkt ausgestellt	✓	✗	✗
Die Token sind dem User Agent nicht zugänglich	✓	✗	✗
Der Client kann authentifiziert werden	✓	✗	✓
Erneuerungstoken können ausgestellt werden	✓	✗	✓
Kommunikation erfolgt in nur einem Anfrage-Antwort-Zyklus	✗	✓	✗
Die meiste Kommunikation erfolgt direkt zwischen den Servern (Client, Autorisierungsserver, Token-Endpunkt)	✓	✗	variiert

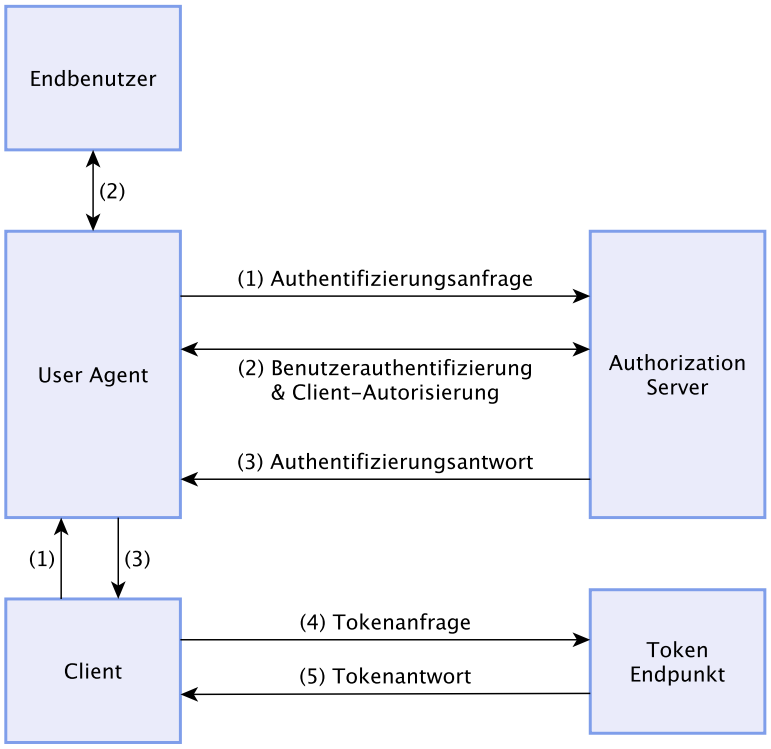
Tabelle 1: Besonderheiten der OpenID Connect-Autorisierungsabläufe

Die Festlegung des Protokollablaufs erfolgt über die Angabe eines Wertes für den Parameter `response_type` in der Autorisierungsanfrage. Tabelle 2 listet die möglichen Wertkombinationen und ihre zugehörigen Autorisierungsabläufe auf.

Wert des Parameters `response_type`	**Autorisierungsablauf**
`code`	Authorization Code Flow
`id_token`	Implicit Flow
`id_token token`	Implicit Flow
`code id_token`	Hybrid Flow
`code token`	Hybrid Flow
`code id_token token`	Hybrid Flow

Tabelle 2: Mögliche OpenID Connect-Autorisierungsabläufe

Abb. 7: OpenID Connect: Authorization Code Flow



4.8 Authorization Code Flow

Der Autorisierungscode-Ablauf (siehe Abbildung 7) basiert auf dem entsprechenden OAuth-Autorisierungsablauf, sodass der generelle Ablauf bereits bekannt ist. Im Folgenden wird daher besonders auf die Eigenheiten der OpenID Connect-Variante eingegangen.

(1) Authentifizierungsanfrage: Sobald der Client die Identität eines Endbenutzers sicherstellen oder weitere Informationen über seine Identität erfahren möchte, initiiert dieser eine Authentifizierungsanfrage, indem er den Browser (User Agent) an den Autorisierungsserver weiterleitet. Neben den Informationen, die bereits bei OAuth beschrieben wurden, muss nun hier der Parameter `scope` den Wert `openid` enthalten und kann zusätzlich die Anforderung der in Abschnitt 4.6 beschriebenen Zusatzinformationen wie `profile` und `phone` spezifizieren.

B

Beispiel 8: Authentifizierungsanfrage bei OpenID Connect an den Autorisierungsserver

```
GET /authorize?response_type=code
   &scope=openid%20profile%20email
   &client_id=r3nHNrlc5Q&state=foo
   &redirect_uri=https%3A%2F%2Fclient.openid.open-c3s.de%2
       Fcb
Host: server.openid.open-c3s.de
```

(2) Benutzerauthentifizierung & Client-Autorisierung: Wie bei OAuth überprüft der Autorisierungsserver den vorliegenden Request. Falls der Endbenutzer nicht bereits authentisiert ist, zeigt der Autorisierungsserver dem Nutzer eine Benutzeroberfläche zur Authentifizierung und Autorisierung

an. Der Endbenutzer authentisiert sich daraufhin mittels des User Agents gegenüber dem Autorisierungsserver und räumt dem Client gegebenenfalls ein Nutzungsrecht für die angeforderten weiteren Profilinformationen ein. Der genaue Ablauf der Authentifizierung obliegt dem Autorisierungsserver und ist nicht näher vorgegeben.

(3) Authentifizierungsantwort: Nach erfolgreicher Authentifizierung und Autorisierung stellt der Autorisierungsserver einen Autorisierungscode aus.

Beispiel 9: Antwort des Autorisierungsservers auf eine OpenID Connect-Authentifizierungsanfrage

B

```
HTTP/1.1 302 Found
Location: https://client.openid.open-c3s.de/cb?code=
    Tet52xl4abqxGTrfNv6ieR&state=foo
```

(4) Tokenanfrage: Die Tokenanfrage entspricht technisch exakt der bei OAuth beschriebenen Interaktion. Es wird aber durch die Anfrage stets ein ID-Token angefordert, neben optional den bereits beschriebenen Zugriffstoken und ggf. Aktualisierungstoken.

Beispiel 10: Anfrage des Clients nach dem Zugriffstoken bei OpenID Connect

B

```
POST /token HTTP/1.1
Host: server.oauth.open-c3s.de
Authorization: Basic dDdBYU10M3M6cGFzc3dvcmQ=
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=Tet52xl4abqxGTrfNv6ieR&
    redirect_uri=https%3A%2F%2Fclient%2Eopenid%2Eopen%2
    Dc3s%2Ede%2Fcb
```

(5) Tokenantwort Der Autorisierungsserver authentifiziert den Client und verifiziert den Autorisierungscode. Nach erfolgreicher Überprüfung stellt der Autorisierungsserver ein ID-Token, gegebenenfalls auch ein Zugriffstoken und ein Aktualisierungstoken aus und sendet diese zusammen mit dem

Gültigkeitsbereich sowie dem Ablaufdatum des Zugriffstokens zurück an den Client. Die Angabe dieser Inhalte erfolgt dabei in JSON-Notation.

B

Beispiel 11: Antwort des Autorisierungsservers mit ID-Token, Zugriffstoken und Aktualisierungstoken bei OpenID Connect

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

 {
   "access_token":"neNa24arke",
   "token_type":"Bearer",
   "expires_in":3600,
   "refresh_token":"jeTr2nr86a",
   "state":"foo",
   "id_token":
       "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e
        yJpc3MiOiJodHRwczovL29wZW5pZC5vcGVuLWM
        zcy5kZSIsInN1YiI6IjQyMjMyMDAzIiwiYXVkI
        joicXQzUmtoZHM2QiIsImV4cCI6MTQ4NDA3NTM
        4MSwiaWF0IjoxNDg0MDc0MzgxLCJhdXRoX3Rpb
        WUiOjE0ODQwNzQzODB9.KXKftZ_xDvd63MJRF5
        JYrjQK-MOEBXmWO4ZV23W4DrM"'
 }
```

Die verwendeten Parameter lauten dabei:

access_token: Das Zugriffstoken, welches vom Autorisierungsserver ausgestellt wurde.

token_type: Der Typ des Zugriffstokens. In diesem Falle stets „`Bearer`“.

expires_in: *[empfohlen]* Die Gültigkeitsdauer des Zugriffstokens in Sekunden.

refresh_token: *[optional]* Das Aktualisierungstoken, welches in der Regel eine weitaus längere Gültigkeitsdauer als das Zugriffstoken besitzt und nach dessen Ablauf für die Generierung neuer Zugriffstoken eingesetzt werden kann.

id_token: Das Base64-kodierte ID-Token, welches mit der gerade stattgefundenen Authentifizierung assoziiert ist.

scope: *[optional]* Die Informationen, welche mit Hilfe des ausgestellten Zugriffstokens abgefragt werden können.

state: Enthält den gleichen Wert wie in der Authentifizierungsanfrage.

Beim ID-Token handelt es sich um ein JSON-Web-Token, welches intern aus Base64-kodierten JSON-Dokumenten aufgebaut ist. Obiges Beispiel enthält nach der Dekodierung folgende Informationen:

```
{
 alg: "HS256",
 typ: "JWT"
}.
{
 iss: "https://openid.open-c3s.de",
```

```
 sub: "42232003",
 aud: "qt3Rkhds6B",
 exp: 1484075381,
 iat: 1484074381,
 auth_time: 1484074380
}.
[signature]
```

Beim Authorization Code Flow kann das ID-Token optional zu den im Abschnitt 4.5 definierten Attributen zusätzlich über ein `at_hash` Attribut verfügen, welches einen kryptographischen Hash-Wert des Zugriffstokens beinhaltet.

4.9 Implicit Flow

Der implizierte Ablauf basiert auf der gleichnamigen OAuth-Protokollvariante. Teilnehmende Clients werden dabei als öffentlich angesehen, da das ID-Token sowie gegebenenfalls das Zugriffstoken direkt vom Autorisierungsserver an den Client übermittelt werden und auf dem Transportweg vom Endbenutzer sowie von Applikationen, die Zugriff auf den Browser des Benutzers haben, eingesehen werden können (vgl. Abschnitt 3.9).

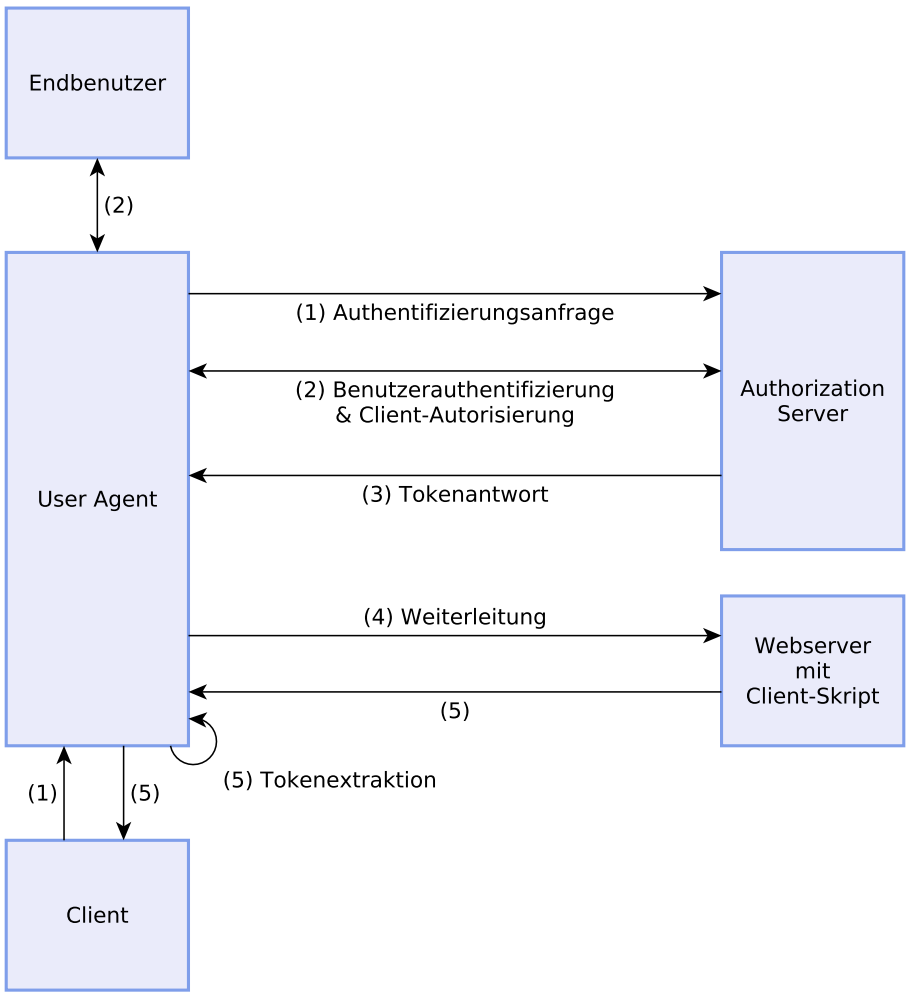


Abb. 8: OpenID Connect: Implicit Flow

Die Erweiterungen gegenüber dem zugrunde liegenden OAuth-2.0-Protokoll entsprechen dabei den in Abschnitt 4.8 beschriebenen Unterschieden.

B

Beispiel 12: Anfrage des Clients mittels des User Agents beim Implicit Flow von OpenID Connect

```
GET /authorize?response_type=id_token%20code
    &client_id=t7AjKt3s
    &scope=openid%20profile
    &state=foo
    &nonce=jk9ie5S6_iG2jk
    &redirect_uri=https%3A%2F%2Fclient%2Eoauth%2Eopen%2Dc3s%2Ede
        %2Fcb HTTP/1.1
Host: server.openid.open-c3s.de
```

Bei der Tokenantwort in Schritt (3) wird dabei – wie bei der expliziten Anforderung von Token vom Token-Endpunkt beim Authorization Code Flow – neben den hier möglichen Zugriffstoken und Erneuerungstoken in jedem Fall das ID-Token als Base64-codierte JSON-Datenstruktur übermittelt.

B

Beispiel 13: Antwort des Autorisierungsservers bei OpenID Connect mit ID-Token und Zugriffstoken

```
HTTP/1.1 302 Found
Location: https://client.oauth.open-c3s.de/cb?#
    access_token=neNa24arke
    &state=foo
    &token_type=Bearer
    &id_token=eyJhbGciOiJI ... S-M_DLTgPPQ8
    &expires_in=3600
```

In Falle des Implicit Flows muss das ID-Token das `nonce` Attribut nutzen. Wenn ein Zugriffstoken ausgestellt wird, muss außerdem das `at_hash`-Attribut verwendet werden um einen kryptographischen Hash-Wert des Zugriffstokens bereitzustellen.

4.10 Hybrid Flow

Neben den beiden vorgestellten Varianten existiert noch eine weitere, die beide Verfahren miteinander kombiniert. Es erlaubt dem Client in einem Ablauf sowohl ID-Token und Zugriffstoken als auch einen Autorisierungscode beim Autorisierungsserver anzufordern. Die Übertragung der Token kann dabei sowohl über den Fragment-Teil der Weiterleitungs-URI als auch über eine eigene Abfrage erfolgen. Dies ermöglicht Clients einerseits die sofortige Verwendung des ID-Tokens, andererseits mit Hilfe des Autorisierungscodes die Bildung von längerfristigen Sitzungen. Der genaue Ablauf des Hybrid Flows wird an dieser Stelle nicht detaillierter beschrieben und kann in der offiziellen Spezifikation nachgelesen werden.

5 Übungsaufgaben

Ü

Übung 1: Authentifizierung und Autorisierung

Erläutern Sie die Begriffe Authentifizerung, Authentisierung und Autorisierung.

Übung 2: Zugriffskontrolle

(a) Was sind die wesentlichen Unterschiede zwischen MAC (Mandatory Access Control) und DAC (Discretionary Access Control)?

(b) Beschreiben Sie, was man unter einer Zugriffskontrollmatrix, einer Zugriffskontrollliste (ACL) und einer Capability-Liste (C-List) versteht.

Ü

Übung 3: Single Sign-on

Diskutieren Sie die Chancen und Risiken von Single-Sign-on-Verfahren.

Ü

Übung 4: Zugriffskontrolle

Auf einem Server werden Dienste unter den TCP-Ports 22 (ssh), 25 (smtp), 443 (https), 143 (imap) und 6969 (bittorrent tracker) betrieben. Der Zugriff auf den Server wird durch eine Firewall eingeschränkt. Dabei gilt das Subnetz 192.168.42.0/24 als vertrauenswürdig und soll Zugriff auf alle Dienste haben, das Subnetz 192.168.43.0/24 dagegen lediglich auf den Bittorrent Tracker. Der SMTP-Server und der HTTPS-Server sollen von überall erreichbar sein. Alle anderen Zugriffe sollen blockiert werden.

Beschreiben Sie die durch die Firewall realisierte Zugriffskontrolle durch eine Zugriffskontrollmatrix. Die Objekte sind dabei die Zielports (22, 25, 443, 143, 6969, sonstige), die Subjekte die Klassen von Quellnetzen (192.168.42.0/24, 192.168.43.0/24, sonstige).

Ü

Übung 5: OAuth 2.0: Clients

Definieren Sie den Unterschied zwischen öffentlichen und vertrauenswürdigen Clients und erklären Sie warum diese Unterscheidung im Protokoll existiert.

Ü

Übung 6: OAuth 2.0: Authentifizierung

Warum eignet sich OAuth 2.0 nicht (direkt) zur Authentifizierung? Welche Gefahren liegen darin begründet?

Ü

Übung 7: OpenID Connect: Protokollablauf

Erläutern Sie anhand des Beispiels eines Online-Forums, bei dem sich ein Benutzer mittels OpenID Connect mit seinem Google-Account anmeldet, den Protokollablauf bei der Authentisierung mit OpenID Connect. Welche der drei beschriebenen Ablaufvarianten erwarten Sie in diesem Anwendungsszenarium? Gehen Sie bei der Ablaufbeschreibung auf den konkreten Nachrichtenaustausch zwischen den Teilnehmern ein.

Ü

Ü

Übung 8: OpenID Connect: Implementierung

Gegeben sei ein webbasierter Dienst, der Benutzer mit Benutzernamen und Passwort authentisiert. Dieser Dienst ist wie folgt aufgebaut:

- Über einen „Anmelden“-Button greift der Nutzer auf eine Login-Seite (`login.jsp`) zu, die eine Anmeldeseite zur Eingabe von Benutzernamen und Passwort darstellt.
- Auf der Anmeldeseite kann der Benutzer die entsprechenden Daten eingeben und über einen „Login“-Button an den Server schicken (`auth.jsp`).
- In `auth.jsp` werden die Anmeldedaten geprüft und bei Korrektheit werden die erfolgreiche Anmeldung und der Benutzername in der HTTP-Session vermerkt. Dann wird der Nutzer zur internen Dienstseite (`messageboard.jsp`) weitergeleitet.

Eine Beispielimplementierung für diesen Dienst erhalten Sie als Ergänzung zu diesem Studienbrief. Ersetzen Sie in dieser Beispielanwendung die passwortbasierte Anmeldung durch ein OpenID-Connect-basiertes Authentifizierungsverfahren und beschreiben Sie, wie Sie dabei vorgegangen sind.

Ü

Übung 9: OpenID Connect: Netzverkehr-Analyse

Als Anlage zu diesem Studientext erhalten Sie einen beim Endbenutzer während einer Authentifizierung mit OpenID Connect aufgezeichneten Mitschnitt des Kommunikationsnetzes sowie (durch geeignete Tools ermittelte) TLS-Session-Keys, mit deren Hilfe verschlüsselte Verbindungen entschlüsselt werden können.

Betrachten Sie die ausgetauschten Daten mit einem geeigneten Werkzeug wie zum Beispiel WireShark[3] und versuchen Sie, folgende Fragen zu beantworten:

- Bei welchem Dienst möchte sich der Nutzer anmelden?
- Welcher OpenID-Connect-Provider wird dafür verwendet?
- Welche Informationen über den Benutzer teilt der OpenID-Connect-Provider dem Dienst mit? Ist der Anmeldevorgang erfolgreich?
- Können Sie dem Mitschnitt entnehmen, auf welche Weise der Benutzer durch den OpenID-Connect-Provider authentifiziert wird?

[3] https://www.wireshark.org – [abgerufen am 2015-03-01]

Verzeichnisse

I. Abbildungen

II. Beispiele

III. Definitionen

IV. Exkurse

V. Tabellen

VI. Literatur

David Elliot Bell und Leonard J. LaPadula. Secure Computer Systems: Mathematical Foundations. Technical Report TR 2547, MITRE Corporation, 1973.

Kenneth J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report ESD-TR 76-372, MITRE Corporation, 1977.

John Bradley. The problem with OAuth for Authentication. Online verfügbar unter http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html [abgerufen am 2017-03-01], 2012.

David Ferraiolo und Richard Kuhn. Role-Based Access Control. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, S. 554–563, 1992.

Gartner Inc. Gartner IT Glossary. Online verfügbar unter http://blogs.gartner.com/it-glossary/identity-and-access-management-iam/ [abgerufen am 2017-01-27].

Eran Hammer-Lahav. The OAuth 1.0 Protocol. RFC 5849 (Informational), April 2010. URL http://www.ietf.org/rfc/rfc5849.txt. Überholt durch RFC 6749.

Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard), Oktober 2012. URL http://www.ietf.org/rfc/rfc6749.txt.

Michael Jones und Dick Hardt. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, Oktober 2012. URL https://rfc-editor.org/rfc/rfc6750.txt.

Mike Jones, John Bradley, und Nat Sakimura. JSON Web Token (JWT). RFC 7519, May 2015. URL http://www.rfc-editor.org/rfc/rfc7519.txt.

Butler W. Lampson. Protection. *Proceedings of the 5th Princeton Symposium on Information Sciences and Systems, Princeton University*, March 1971.

Hal Lockhart und Brian Campbell. SAML V2.0. Oasis, online verfügbar unter https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf [abgerufen am 2017-01-02], 2008.

Mozilla Corporation. Persona. Online verfügbar unter https://developer.mozilla.org/en-US/Persona [abgerufen am 2017-01-02], 2014.

Clifford Neuman, Tom Yu, Sam Hartman, und Kenneth Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), Juli 2005. URL http://www.ietf.org/rfc/rfc4120.txt.

David Recordon und Brad Fitzpatrick. OpenID Authentication 1.1. OpenID Foundation, online verfügbar unter http://openid.net/specs/openid-authentication-1_1.html [abgerufen am 2017-02-14], 2006.

Nat Sakimura, John Bradley, Mike Jones, Breno de Medeiros, und Chuck Mortimore. OpenID Connect Core 1.0 incorporating errata set 1. OpenID Foundation, online verfügbar unter http://openid.net/specs/openid-connect-core-1_0.html [abgerufen am 2017-02-14], 2014a.

Nat Sakimura, John Bradley, Mike Jones, und Edmund Jay. OpenID Connect Discovery 1.0 incorporating errata set 1. OpenID Foundation, online verfügbar unter https://openid.net/specs/openid-connect-discovery-1_0.html [abgerufen am 2017-02-14], 2014b.