OpenID Connect: Login mit OAuth, Teil 1 – Grundlagen

Standards 10.06.2014 14:10 Uhr – Torsten Lodderstedt – 0 Kommentare

OAuth 2.0 hat sich als Standard für die Autorisierung von API-Zugriffen im Web etabliert. Wer es auch zum Login einsetzen wollte, war bisher auf anbieterspezifische Erweiterungen angewiesen. Mit OpenID Connect gibt es jetzt einen neuen Standard, der OAuth 2.0 um alle notwendigen Funktionen für Login und Single Sign-On erweitert.

Da OpenID Connect als Erweiterung von OAuth 2.0 konzipiert ist, profitiert es von dessen grundlegenden Eigenschaften, zum Beispiel Einfachheit, Sicherheit und Unterstützung diverser Klassen von Anwendungen – von Web-Portalen bis hin zu Apps auf diversen Endgeräten. Durch die Integration des Logins in OAuth wird es darüber hinaus möglich, mit demselben Protokoll kombinierte Szenarien, also das Login im Namen des Benutzers und den abgesicherten Zugriff auf dessen Ressourcen, zu implementieren. Dafür waren bisher Hybridprotokolle wie Googles Hybrid

aus OpenID 2.0 und OAuth erforderlich.

OpenID Connect wurde am 26. Februar 2014 als Standard der OpenID Foundation verabschiedet und ist bereits von Anbietern wie Google, Microsoft und der Deutschen Telekom implementiert worden. Zwei Artikel erläutern das Protokoll und zeigen anhand von Beispielen, wie sich unterschiedliche Anwendungsfälle verschiedener Komplexitätsstufen implementieren lassen. Dabei greift der Autor auf Erfahrungen aus der Implementierung des Protokolls bei der Deutschen Telekom zurück. Im ersten Artikel stehen die Grundlagen im Fokus, der zweite behandelt weiterführende Themen wie die Integration von unterschiedlichen Identitätsprovidern in eine Anwendung und fortgeschrittene Steuerungsmöglichkeiten für den Login-Prozess. Es werden Kenntnisse des unterliegenden Protokolls OAuth 2.0 vorausgesetzt. Hierfür sei auf den Vorgängerartikel zu OAuth 2.0 verwiesen.

Der Blick zurück

Wer bisher Single Sign-On (SSO) zwischen den Portalen eines Anbieters oder eine ID-Föderation zwischen Webportalen und Apps unterschiedlicher Anbieter auf standardisierte Art und Weise implementieren wollte, hatte die Wahl zwischen OpenID 2.0 und SAML 2.0 (Security Assertion Markup Language). Obgleich OpenID 2.0 ein relativ einfaches Protokoll ist, hat es einige Schwächen im Bereich der Sicherheit. Außerdem ist der limitiert, sodass der Einsatz in etwas anspruchsvolleren Szenarien Erweiterungen erfordert. SAML 2.0 hingegen ist ein sicheres und mächtiges Protokoll, das einige Applikationsserver direkt unterstützen. Es krankt allerdings an der inhärenten Komplexität des unterliegenden Standards für XML-Signaturen und hat sich im Betrieb aufgrund der Verwendung von X.509-Zertifikaten als schwer zu beherrschen herausgestellt. Dass beide Protokolle nicht auf die Erfordernisse moderner Apps ausgelegt sind, überrascht nicht, schließlich entstanden sie vor deren Erfindung (2007 bzw. 2005).

Gleichzeitig hat sich OAuth 2.0 im Verlaufe der letzten Jahre als Standard für die Absicherung des Zugriffs auf alle Arten von APIs im Internet durchgesetzt. Da der OAuth-Prozess in der Regel auch mit einem Login des Benutzers einhergeht, erscheint es für immer mehr Dienstanbieter naheliegend,

OAuth auch für das Login einzusetzen. OAuth ist eine einfache und sichere Basis, die zudem zeitgemäße Anforderungen wie die Unterstützung von Apps auf Smartphones oder Smart-TVs erfüllt.

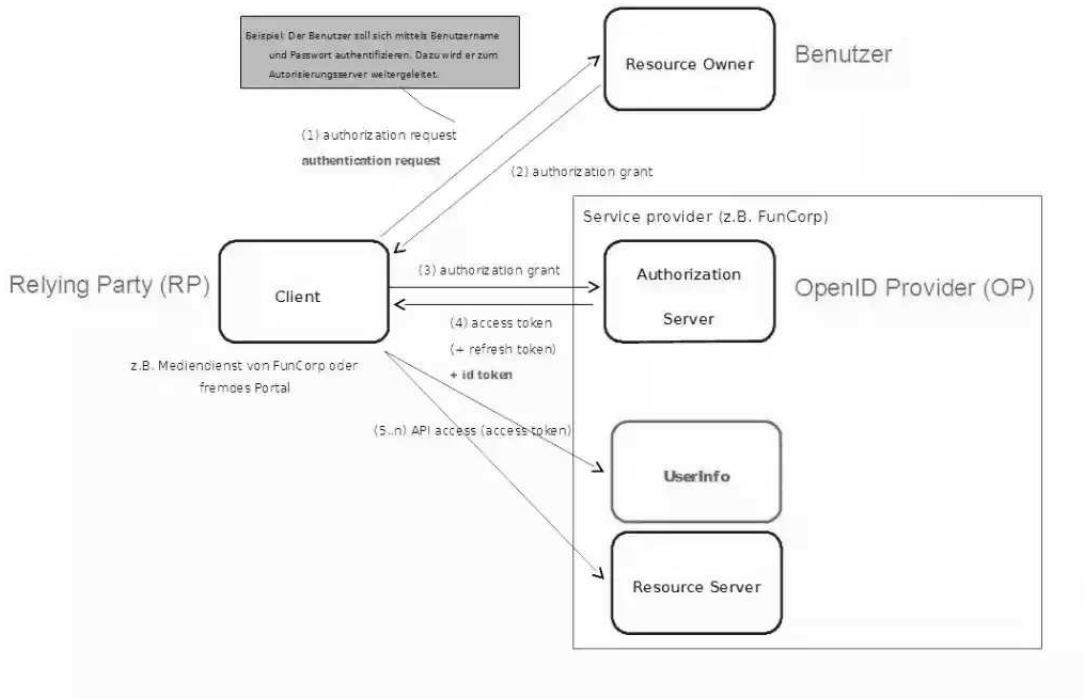
Allerdings fehlen OAuth als Autorisierungsprotokoll einige für das Login wichtige Fähigkeiten: Zum einen stellt der OAuth-Autorisierungsserver dem Client (bewusst) keine Benutzer-ID oder andere Benutzerdaten zu Verfügung. Des Weiteren fehlen Funktionen zur Beeinflussung des Authentifizierungsprozesses (z. B. zum Erzwingen eines Logins) und zur gesicherten Übertragung von Informationen zum Authentifizierungsprozess (bspw. Zeitpunkt und verwendete Authentifizierungsmethoden).

Einige Dienstanbieter wie Facebook und Amazon haben daher für den Zweck des Logins proprietäre OAuth-Erweiterungen implementiert. Dabei wird in der Regel ein OAuth Access Token verwendet, um die eigentlichen Benutzerdaten von einer User Profile API abzufragen. Diese APIs sind aber leider nicht interoperabel. Darüber hinaus birgt die Vorgehensweise auch ein Sicherheitsrisiko. Letztlich kann jeder, der in den Besitz eines passenden Access Token gelangt, sich mit diesem bei allen angeschlossenen Anwendungen im Namen des betreffenden Benutzers einloggen. Der Angriff kann hierbei zum Beispiel von einer Anwendung ausgehen, in die sich der Benutzer eingeloggt hat. Der Betreiber dieser App kann das Token seinerseits verwenden, um sich an einem anderen Gerät in eine oder mehrere andere Apps im Namen des legitimen Benutzers einzuloggen. Das ist möglich, da die Gültigkeit dieses Access Token nicht auf eine bestimmte (Client-)Anwendung, sondern nur auf die User Profile API des Dienstanbieters eingeschränkt ist. Diese Art von Angriffen bezeichnet man
gemeinhin als Substitutionsangriff.

OpenID Connect

OpenID Connect als Antwort

Die Grundidee von OpenID Connect ist, die Authentifizierung eines Benutzers als eine Erweiterung des OAuth-2.0-Autorisierungsprozesses zu implementieren. Ein OAuth-Client kann die OpenID-Erweiterung anfordern, indem er einfach den Scope-Wert "openid" zu einem Autorisierungs-Request hinzufügt. Informationen über die Identität des Benutzers und die durchgeführte Authentifizierung werden dann in einem zusätzlichen (OpenID-spezifischen) Token, dem sogenannten ID Token, an den Client zurückgegeben. Und dieses Token wiederum ist auf das Login bei diesem Client eingeschränkt, ein anderer Client darf das betreffende ID Token nicht akzeptieren. Zusätzlich wird eine weitere OAuth-geschützte API für den Zugriff auf Benutzerdaten, der UserInfo Endpunkt, eingeführt.



Die Abbildung illustriert die Erweiterungen an der Standard-OAuth-Architektur, wobei die Erweiterungen in grüner Farbe hervorgehoben sind.

Wie zu sehen, wird jeder OAuth-2.0-Autorisierungsserver, der OpenID Connect implementiert, auch als OpenID Provider (OP) bezeichnet. Entsprechend heißt jeder OAuth-2.0-Client, der OpenID Connect benutzt, Relying Party (RP).

Einfach einloggen

Den Ablauf eines einfachen Logins sollen nun ein Beispiel und die Übersichtsdarstellung illustrieren: Max verwendet verschiedene Dienste (wie E-Mail und Medienspeicher) des Anbieters Fun Corp. Dieser bietet ein zentrales Identity Management, um seinen Kunden nicht nur einen einheitlichen User Account, sondern auch eine SSO-Erfahrung über die verschiedenen Portale anbieten zu können. Technisch gesehen handelt es sich um einen OAuth-Autorisierungsserver, der auch als OpenID Provider fungiert.

Max möchte sich nun im Mediendienst einloggen, um seine Bilder anzuschauen. Der Mediendienst implementiert das Login unter Nutzung eines Authentifizierungs-Requests auf der Basis des OAuth-Grant-Type-Autorisierungscode wie folgt:

```
HTTP/1.1 302 Found
Location: https://accounts.funcorp.com/oauth/auth?
  response_type=code
  &scope=openid
  &client_id=SDFGHJKLUZTREDFGHJ
  &state=30096545678909876567
  &redirect_uri=https%3A%2F%2Fmediaservice.funcorp.com%2Flogin_cb
```

Dieser Request leitet den Browser des Benutzers mittels HTTP Redirect um und sendet einen Authentifizierungs-Request an den Autorisierungsserver. Der einzige Unterschied zu einem "gewöhnlichen" OAuth-Autorisierungs-Request ist der Scope-Wert "openid". Dieser signalisiert dem Server, dass der Client Identitätsdaten des Benutzers abfragen möchte, und aktiviert damit dessen OpenID-Connect-"Maschinerie". Wie der Server den Benutzer authentifiziert, ist nicht spezifiziert und liegt in der Entscheidung des OpenID Provider. Für gewöhnlich werden hierfür Benutzername und Passwort verwendet, aber es lassen sich auch andere Mechanismen einsetzen. Wenn bereits eine Login-Sitzung beim OP existiert, ist typischerweise für eine bestimmte Zeit kein erneutes manuelles Login erforderlich (SSO).

Es sei an dieser Stelle darauf hingewiesen, dass diese Entkopplung der Anforderung nach Identitätsdaten sowie der aktuellen Art und Weise der Durchführung der Authentifizierung eine wesentliche Stärke von OpenID ist. Hierdurch kann der OP starke und wechselnde

Authentifizierungsmethoden einsetzen und ausgefeilte Sicherheitsmaßnahmen gegen Angriffe implementieren, ohne dass die verschiedenen Anwendungen darauf Rücksicht nehmen müssen oder anzupassen wären.

Nach Abschluss der Authentifizierung stellt der Autorisierungsserver der Anwendung (wie bei OAuth üblich) einen Autorisierungs-Code aus. Dieser wird über den User Agent via HTTP Redirect an den Mediendienst gesendet.

```
HTTP/1.1 302 Found
Location: https://mediaservice.funcorp.com/login_cb?
  code=287654789765678976556789
  &state=30096545678909876567
```

Diese Nachricht entspricht dem OAuth-2.0-Standard und enthält keine OpenID-Connect-Spezifika. Neben dem Autorisierungs-Code enthält sie den Wert des *state*-Parameters, so wie ihn der Mediendienst an den Authentifizierungs-Request übergeben hatte. Der Dienst tauscht dann den Autorisierungs-Code am Token-Endpunkt des Autorisierungsservers gegen Tokens um.

```
POST /oauth/token HTTP/1.1
Host: accounts.funcorp.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic U0RGR0hKS0xVWlRSRURGR0hKOnRlc3QxMjM0
grant_type=authorization_code&
code=287654789765678976556789&
redirect_uri=https%3A%2F%2Fmediaservice.funcorp.com%2Flogin_cb
```

Auch dieser Request entspricht unverändert dem OAuth-Standard, die Response vom Autorisierungsserver hingegen enthält den neuen Ergebnis-Parameter *"id_token"*:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
 "access_token": "IiwKICJleHAiOiAxMzExMjgxOTcwLAo",
 "token_type": "Bearer",
 "expires_in": 3600,
 "id_token": "eyJhbGciOiJub25lIn0.
eyJpc3MiOiAiaHR0cHM6Ly9hY2NvdW50cy5mdW5jb3JwLm
NvbSIsCiAic3ViIjogIjc4Njc2NzY3NyIsCiAiYXVkIjogIlNERkdISk
tMVVpUUkVERkdISiIsCiAiZXhwIjogMTM4ODcwMDM2MDAw
MCwKICJpYXQiOjEzODg2OTY3NjAwMDAsCiAiYXV0aF90aW1
lIjogMTM4ODY5Njc2MDAwMAogImFjciI6ICJ1cm46ZnVuY29
ycDpjb206bGV2ZWwtMSJ9."
}
```

Wie der Name vermuten lässt, enthält dieser Parameter das erwähnte ID Token. Es enthält Identitätsdaten und Informationen über den Authentifizierungsprozess in Form eines sogenannten JSON Web Token. Um dieses gegen Modifikation und andere Angriffe abzusichern, lässt es sich digital signieren und verschlüsseln. Welche Verfahren im konkreten Fall zum Einsatz kommen, erfährt der Client, wenn er den ersten Teil des ID Token bis zum ersten Trennzeichen '.' Base64URL-dekodiert, um den sogenannten Header zu extrahieren. Im Beispiel ergibt das folgendes JSON-Objekt:

```
{"alg":"none"}
```

Dieser Header bedeutet, dass im Beispiel keine Signatur (und auch keine Verschlüsselung) verwendet wird. Das ist auch nicht unbedingt erforderlich, da der Client die Daten vom OP über einen HTTPS-geschützten Kanal erlangt.

Die eigentlichen Nutzdaten des JSON Web Token erlangt der Client, indem er den zweiten Teil des ID Token bis zum nächsten Trennzeichen '.' Base64URL-dekodiert:

```
{
  "iss": "https://accounts.funcorp.com",
  "sub": "786767677",
  "aud": "SDFGHJKLUZTREDFGHJ",
  "exp": 1388700360000,
  "iat":1388696760000,
  "auth_time": 1388696760000
  "acr": "urn:funcorp:com:level-1"
}
```

Wie zu sehen ist, handelt es sich bei einem JSON Web Token um ein einfach zu verarbeitendes JSON-Dokument mit diversen Feldern, jedes enthält einen sogenannten "Claim" zum Benutzer oder zum Authentifizierungsprozess.

Der Claim *"iss"* (Issuer) identifiziert den OpenID Provider, in diesem Fall den Autorisierungsserver von Fun Corp. Der Claim *"sub"* (Subject) enthält die Benutzer-ID des in diesem Prozess authentifizierten User Account. Der Wert von *sub* ist in Bezug auf einen bestimmten Client stabil und lässt sich zur Anlage und Wiedererkennung eines Kontos beim Client verwenden. Wichtig ist anzumerken, dass *"sub"* immer nur innerhalb des Issuers eindeutig ist. Clients, die verschiedene OPs für das Login verwenden, sollten also das Tupel (*iss*, *sub*) zur Identifikation des betreffenden Accounts verwenden.

Der Claim *"aud"* (Audience) identifiziert die legitimen Empfänger des Tokens, im einfachsten Fall enthält er nur die *client_id* der Relying Party. Er kann aber auch weitere Empfänger besitzen. Wie erwähnt, darf ein ID Token ausschließlich zum Login beim betreffenden Client und den zusätzlichen Empfängern verwendet werden (das wird auch als "Audience Restriction" bezeichnet). Durch diese Maßnahme soll das Hijacking von Identitäten durch böswillige Anwendungen (Substitutionsangriff) verhindert werden. Für weitere Informationen zu Substitutionsangriffen sei auf dieses und dieses Dokument verwiesen.

Es ist also essenziell, dass der Client prüft, dass der Claim *"aud"* seine *client_id* enthält. Diese Anforderung ist im Beispiel (wenig überraschend) erfüllt. Der Wert von *"aud"* ist derselbe Wert, wie ihn der Client im initialen Authentifizierungs-Request in den Parameter *client_id* übergeben hat. Damit darf der Client das Login-Ergebnis verarbeiten.

Die Claims *"iat"* (Issued At) und *"exp"* (Expires At) geben Auskunft, wann das Token erzeugt wurde und bis wann es gültig ist. Die Empfänger dürfen das Token nach Ablauf der Gültigkeit nicht mehr
verwenden. Typischerweise besteht ein Zusammenhang zwischen der Gültigkeitsdauer und dem kryptographischen Schutz des Tokens.

Im Claim *"auth_time"* gibt der OP Auskunft darüber, wann der Benutzer authentifiziert wurde. Wenn der OP SSO und gegebenenfalls sogar langlebige Sitzungen (solche Sitzungen überdauern auch das Schließen des Browsers) unterstützt, kann dieser Zeitpunkt weit in der Vergangenheit liegen. Im Claim *"acr"* (Authentication Context Class Reference) kann der Client erfahren, welche Methode(n) für die Authentifizierung des Benutzers zum Einsatz kam. Der Wert von *"acr"* ist eine Abstraktion der Authentifizierungsmethoden und der Gesamtheit der angewendeten Policies (auch Account Management etc.). Wenn der Client spezielle Anforderungen an den Zeitpunkt und die Qualität der Authentifizierung hat, dann kann er diese durch weitere Parameter des Authentifizierungs-Requests ausdrücken. Diese Möglichkeiten werden im Folgeartikel beschrieben. Das ID Token lässt sich darüber hinaus auch als Container für beliebige weitere Benutzerdaten verwenden, auch dieses Thema beleuchtet der zweite Artikel.

Auf der Basis der Daten im ID Token kann der Mediendienst den Benutzer auf sichere Art und Weise identifizieren und einloggen. Im Beispiel sucht und findet der Mediendienst den Account mit der Benutzer-ID "786767677" und lädt dessen Daten.

Die gezeigte Vorgehensweise lässt sich prinzipiell mit jedem OAuth Grant Type kombinieren. Der Standard definiert die Abbildung für Autorisierungs-Code, Implicit Grant und Refresh Tokens. Beim Implicit Grant, der insbesondere für die Integration von JavaScript-Anwendungen gedacht ist, sendet der OAuth-Autorisierungsserver die Ergebnisse des Autorisierungsprozesses in einem sogenannten URL-Fragment der Redirect-URL direkt zum Client. Bei Verwendung dieses Grant Type im Kontext von OpenID wird das ID Token

folgerichtig als weiteres Element des Redirects vom Autorisierungsserver zum Client wie folgt gesendet:

```
HTTP/1.1 302 Found
Location: https://client.example.org/cb#
  access_token=SlAV32hkKG
  &token_type=bearer
  &id_token=eyJ0 ... NiJ9.eyJ1c ... I6IjIifX0.DeWt4Qu ... ZXso
  &expires_in=3600
  &state=af0ifjsldkj
```

Daraus ergeben sich wichtige Unterschiede zum Autorisierungs-Code in Bezug auf die Sicherheit des Ablaufs. Da das ID Token nicht auf einem direkten und geschützten Kanal zwischen OP und RP, sondern über den Umweg des Browsers des Benutzers ausgetauscht wird, ist das ID Token unbedingt durch eine digitale Signatur gegen Modifikation zu schützen. Ansonsten könnte sich der Benutzer mit einer beliebigen Benutzer-ID beim Client einloggen. Des weiteren muß das ID Token gegen Replay-Angriffe abgesichert werden. Hierzu wird es durch ein sogenanntes "Nonce" an einen bestimmt Authentifizierungs-Vorgang gebunden.

Erweitertes Login

Login mit weiteren Benutzerdaten

In vielen Fällen möchten Clients über die (technische) Benutzer-ID hinaus weitere Attribute der Benutzer abfragen, zum Beispiel den Namen oder die E-Mail-Adresse. Hierfür führt OpenID Connect mit dem UserInfo-Endpunkt eine neue Schnittstelle am OP ein. Unter Nutzung dieses Endpunkts kann ein Client die beim OP verfügbaren Attribute zu einem Benutzer, ebenfalls "Claims" genannt, abfragen.

Die Zugriffe auf den UserInfo-Endpunkt werden durch gewöhnliche OAuth Access Token geschützt. Der Client kann daher alle verfügbaren OAuth-Abläufe verwenden, um ein entsprechend autorisiertes Access Token zu erlangen. Im einfachsten Fall erweitert der Client den Scope-Wert des OpenID-Connect-Authentifizierungs-Requests um vordefinierte Werte,

die den Zugriff auf bestimmte Claims anfordern. OpenID Connect definiert die in der folgenden Tabelle dargestellten Standardwerte:

Scope-Wert	Claims
profile	name, family_name, given_name, middle_name, nickname, preferred_username, profile, picture, website, gender, birthdate, zoneinfo, locale, and updated_at
email	email, email_verified
address	formatted, street_address, locality, region, postal_code, country
phone	phone_number, phone_number_verified

Diese Vorgehensweise soll nun das Beispiel illustrieren. Die Annahme ist, dass der Mediendienst sowohl den Namen als auch die E-Mail-Adresse jedes Benutzers abfragen möchte. Hierzu erweitert dieser den Authentifizierungs-Request um die Scope-Werte "profile" und "email" wie folgt:

```
HTTP/1.1 302 Found
Location: https://accounts.funcorp.com/oauth/auth?
  response_type=code
  &scope=openid%20profile%20email
  &client_id=SDFGHJKLUZTREDFGHJ
  &state=34790876543456789765
  &redirect_uri=https%3A%2F%2Fmediaservice.funcorp.com%2Flogin_cb
```

Der Rest des Ablaufs bleibt unverändert. Wenn der Authentifizierungsablauf erfolgreich abgeschlossen ist,

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

```
 Cache-Control: no-store
Pragma: no-cache
{
 "access_token": "4vfKjkM8FcGvnzZUN4_KSP0aAp",
 "token_type": "Bearer",
 "expires_in": 3600,
 "id_token": "eyJhb...cifQ.ew...fQ.gg...zqg"
}
```

dann ist der Mediendienst im Besitz eines Access Token, das diesen zum Zugriff auf die gewünschten Daten berechtigt. Die Abfrage am UserInfo-Endpunkt sieht dann wie folgt aus:

```
GET /userinfo HTTP/1.1
  Host: accounts.funcorp.com
  Authorization: Bearer 4vfKjkM8FcGvnzZUN4_KSP0aAp
```

Hierbei wird das Access Token unter Nutzung des mit OAuth 2.0 eingeführten HTTP-Authentifizierungsschemas Bearer an den Server geschickt. Wenn dieses Token gültig ist, werden alle hierdurch autorisierten Benutzerdaten an den Aufrufer geliefert.

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

{
 "sub": "786767677",
 "name": "Max Mustermann",
 "given_name": "Max",
 "family_name": "Mustermann",
 "email": "m.mustermann@funcorp.com",
 "email_verified": true
}
```

Wie im Beispiel zu sehen ist, liefert der UserInfo-Endpunkt auch die aus dem ID Token bekannte Benutzer-ID an den Client aus. Der Client muss immer prüfen, dass beide Werte übereinstimmen. Sollte das nicht der Fall sein, dann könnte ein Substitutionsangriff vorliegen. Daher dürfen die Benutzerdaten in einem solchen Fall nicht verwendet werden.

Mit den bisher vorgestellten Funktionen sind die grundsätzlichen Anforderungen für das Login auf einfache Art und Weise abgedeckt. Aber auch wenn die Anwendungsfälle komplizierter werden, hat OpenID Connect eine Menge zu bieten. Einige dieser Anwendungsfälle bespricht der Folgeartikel.

Fazit

OpenID Connect ist eine zeitgemäße Weiterentwicklung klassischer Protokolle wie SAML und OpenID, die die Erfordernisse der API- und Apps-Ökonomie berücksichtigt. Entsprechend der Devise "einfache Dinge sollten einfach und komplizierte Dinge sollten möglich sein" erlaubt OpenID Connect dabei einen einfachen Einstieg in das Thema. In den hier gezeigten Anwendungsfällen kommt der Client zum Beispiel mit ein paar einfachen HTTP Requests ohne irgendwelche Kryptographie aus und kann damit ein komplettes Login inklusive des Zugriffs auf Benutzerdaten implementieren. Und Einfachheit ist erfahrungsgemäß gerade in sicherheitskritischen Bereichen wie dem Login ein erfolgskritischer Faktor, insbesondere wenn man berücksichtigt, dass in der Regel leider weder Identity Management noch Sicherheit zu den Kernkompetenzen von Entwicklern zählen.

Aber da ist noch mehr: Um eine wirklich zeitgemäße Nutzungserfahrung implementieren zu können und dabei seine Sicherheitsziele zu erreichen, muss der Client den Authentifizierungsprozess enger steuern können. Darüber hinaus stellt sich die Frage, wie man die Interoperabilität eines sich am Markt etablierenden Standards nutzen kann, um die Benutzerbasen anderer Anbietern "anzapfen" zu können. Schließlich muss sich der Anbieter einer App auch die Frage stellen, ob er in einer Zeit, in der der Diebstahl von Passwörtern ein zunehmendes und bedrohliches Phänomen ist, nicht die Authentifizierung als sicherheitskritische Funktion an Experten auslagern möchte. Diese und andere Themen wird der zweite Artikel beleuchten. (ane)

Dr. Torsten Lodderstedt
verantwortet bei der Deutschen Telekom als Abteilungsleiter die Entwicklung von Basisdiensten für das Identitäts- und Vertrags-Management sowie Bezahlfunktionen, die in diversen Endkundenprodukten zum Einsatz kommen. Er ist Corporate Director der OpenID Foundation und an der Spezifikation von OpenID Connect und OAuth beteiligt.

Kommentieren

Themenseiten:

OPENID

Web-Betrug: Wenn Bots Login-Daten missbrauchen
eBook DRaaS: Katastrophenschutz aus der Cloud
DSGVO: Personenbez. Daten finden und schützen

https://heise.de/-2218446

Drucken