Transferleistung Theorie/Praxis* 2

Matrikelnummer:	8323
Freigegebenes Thema:	Weiterleitung der Zwei-Faktor-authentifizierten Anmeldung eines Versicherten aus der TK-App an die Webanwendung Meine TK
Studiengang, Zenturie:	A17b. Angewandte Informatik / 2017.

* *Studierende, die unter den Anwendungsbereich der PVO bis 03.02.2015 fallen, fertigen Transferleistungen weiterhin in der Form von Praxisberichten an und der Begriff hält Einzug in das Abschlusszeugnis. Ab dem Jahrgang 2016 hat der Begriff vollumfängliche Gültigkeit. In der Kommunikation hält der Begriff Transferleistungen ab sofort Einzug.*

NORDAKADEMIE
HOCHSCHULE DER WIRTSCHAFT

THEORIE PRAXIS

Weiterleitung der Zwei-Faktor authentifizierten Anmeldung eines Versicherten aus der TK-App an die Webanwendung Meine TK.

Transferleistung
im Studiengang Angewandte Informatik
angefertigt im Projekt-Team Online and Mobile Projects
der Techniker Krankenkasse

Martrikelnummer: 8323

Hamburg, den 29. Dezember 2018

Sperrvermerk

Die Inhalte dieser Arbeit betreffen Informationen über betriebliche Belange der Techniker Krankenkasse. Dies gilt ebenfalls bezüglich der für die Beurteilung der Arbeit zur Verfügung gestellten Dokumente der Techniker Krankenkasse. Die Arbeit selbst und weitere Dokumente sind für den innerbetrieblichen Gebrauch bestimmt und dürfen somit der Öffentlichkeit – auch in Auszügen – nicht zugänglich gemacht werden.

Kurzzusammenfassung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: Dies ist ein Blindtext oder Huardest gefburn? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie Lorem ipsum dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

Glossar

Grant Type bestimmt, wie Clients Authorization Grants von Resource Ownern erhalten können. Die vier Grant Types Authorization Code, Implicit, Resource Owner Password Credentials sowie Client Credentials wurden in RFC 6749 definiert; auch eigene Grant Types zu definieren ist möglich (vgl. Hardt 2012, S. 23ff.).. 2, 3

Internet Engineering Task Force (IETF) Organisation von Workinggroups, die sich mit je einem spezifischen Thema befassen. Teil der Internet Society (ISOC). 2, 7

Minimal Viable Product (MVP) Umsetzung eines Features oder Produktes mit zwar minimalem Funktionsumfang, aber dennoch konkretem Mehrwert für den Nutzer. 1

OAuth 2.0 Protokoll zur API-basierten Autorisierung. 2, 3, 7, 9

Transport Layer Security (TLS) Protokoll zur hybriden Verschlüsselung von Datenübertragungen, Grundlage für HTTPS. 3

Kapitel 1

Einleitung

Moderne Softwareentwicklung orientiert sich seit spätestens 2013 an agilen Arbeitsmodellen, um . So auch das Projekt-Team, das mit der Konzeption und Entwicklung der TK-App betraut ist. Teil der agilen Planung ist das Minimal Viable Product (MVP). Da MVPs bewusst unvollständig sind, d.i. kleine Teile der Nutzergruppe werden, gerade in den ersten Zyklen, nicht abgebildet.

Die Frage, ob sich OAuth, das als Autentisierungsverfahren gedacht ist, für die authentifizierung eines eignet, wird in dieser Arbeit elegant umschifft.

Belege

recht-
fertigen,
wozu,
weshalb
warum

Hypo-
these
(begrün-
dete Ver
mutung
über die
Relation
von min
destens
zwei
Prä-
missen)
oder For
schungs
fragen
formulie
ren?

Kapitel 2

OAuth 2.0

OAuth 2.0 ist ein von der Internet Engineering Task Force (IETF) konzipiertes Protokoll zur Delegation von Zugriffsrechten für Anwendungssoftware. Zugriffsrechte werden stets durch den Besitzer der Ressource, den Resource Owner, erteilt. Sie werden durch zeitlich limitierte Access-Tokens abgebildet. Access-Tokens lassen sich gut über den Vergleich mit Konzerttickets oder Bargeld erläutert. Wer im Besitz des zeitlich beschränkten Tickets ist, der Zeichenfolge des Tokens, muss sich nicht weiter ausweisen und ist so auch im Besitz der mit dem Token einhergehenden Zugriffsrechte(vgl. Siriwardena 2014, S. 134), beispielsweise auf eine API, die Nutzerdaten zurückgibt. Access-Tokens sind also eine Möglichkeit für den Resource Owner, Drittanbietern limitierten Zugriff auf Daten zu ermöglichen, die für ihn bereits von einer Anwendung verwaltet werden. Das Zugriffsrecht geht dabei allein auf den Besitz des Tokens zurück. Auf Access-Tokens soll in Abschnitt 2.3 näher eingegangen werden. Die Übergabe des Access-Tokens ist durch sogenannte Grant Types organisiert. Sie beschreiben die Interaktionen zwischen den OAuth-Parteien, die notwendig sind, um einen Access-Token zu erlangen. Die in dieser Arbeit genutzten Grant Types, Authorization Grant sowie Implicit Grant, werden in Abschnitt 2.2 erläutert. Die Kommunikation innerhalb der Grant Types erfolgt mittels standardisierter Nachrichten, die als JSON per HTTP-Methode ausgetauscht werden (vgl. Hardt 2012, S. 29ff.). Im Folgenden werden die in OAuth 2.0 beteiligten Parteien, Objekte und Grant Types erläutert.

2.1 OAuth-Parteien

Resource Owner	besitzt geschützte Ressourcen, die auf einem Resource Server gehostet werden.
Resource Server	hostet geschützte Ressourcen, nimmt Requests auf geschützte Ressourcen entgegen und gewährt Zugriff, sofern der übermittelte Access-Token valide ist.
Client Application	ist eine vom Resource Owner genutzte Anwendung, die Zugriff auf geschützte Ressourcen anfragt und entsprechende Requests an den Resource Server stellt. Jeder Client muss sich am Authorization Server registrieren, woraufhin ihm eine Client-ID und ein Client-Secret zugewiesen werden.
Authorization Server	authentifiziert den Resource Owner anhand seiner Zugangsdaten und stellt dem Client je nach Grant Type einen Authorization Code oder einen Access-Token auf die angefragte Ressource zur Verfügung. (vgl. Hardt 2012, S. 6)

2.2 Grant Types

Grant Types, auch Flows genannt, beschreiben die Interaktionen der vier Parteien untereinander, die notwendig sind, um Zugriff auf eine geschützte Ressource zu erhalten. Abbildung 2.1 stellt den grundlegenden Kommunikationsablaufzwischen dem Client auf der einen, sowie Resource Owner, Authorization Server und Resource Server auf der anderen Seite dar. Jegliche Netzwerkkommunikation zwischen den Parteien sollte hierbei unbedingt durch TLS geschützt werden(vgl. Siriwardena 2014, S. 99). Der Protocol Flow lässt sich auf drei grundlegende Schritte reduzieren.

Authorization Grant — der Client stößt einen der vier spezifizierten Grant Types an.

Access-Token Retrieval — vorausgesetzt, der Grant war erfolgreich, erhält der Client einen Access-Token. In einigen Grant Types spiele Refresh-Tokens eine Rolle, in dieser Arbeit spielen sie keine…

Resource Access — durch Vorlage des Access-Tokens erhält der Client Zugriff auf die entsprechende Ressource.

Im Folgenden werden insbesondere der Authorization Code Grant sowie der Implicit Grant erläutert werden. Resource Owner Password Credentials Grant und Client Credentials Grant sind Reduktionen der beiden vorgestellten Grant Types und werden nicht weiter behandelt.

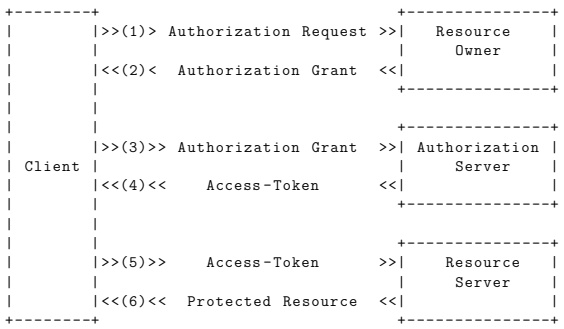

Abb. 2.1: Protocol Flow

2.2.1 Authorization Code Grant

Der Authorization Code Grant ist immer dann möglich, wenn der Client auf einen Web-Browser zugreifen kann und über ein eigenes Backend verfügt oder anderweitig dazu in der Lage ist, sein Client Secret geheimzuhalten. Zwar ist der Authorization Code Grant, aufgrund der vielen Authentifizierungszwischenschritte der umfangreichste, hierdurch aber auch der sicherste der in OAuth 2.0 genutzten Grant Types. Vorausgesetzt, die Übermittlung der Daten erfolgt per Transport Layer Security (TLS), ist die Zuverlässigkeit des Authorization Code Grant formal gesichert (Chari, Jutla & Roy 2011). Wie schon der Protocol Flow, lässt sich der Authorization Code Grant leicht herunterbrechen. Er verfolgt zwei zentrale Ziele: zum einen, den Erwerb eines Authorization Codes, zum anderen das Einlösen dieses Authorization Codes gegen einen Access-Token.

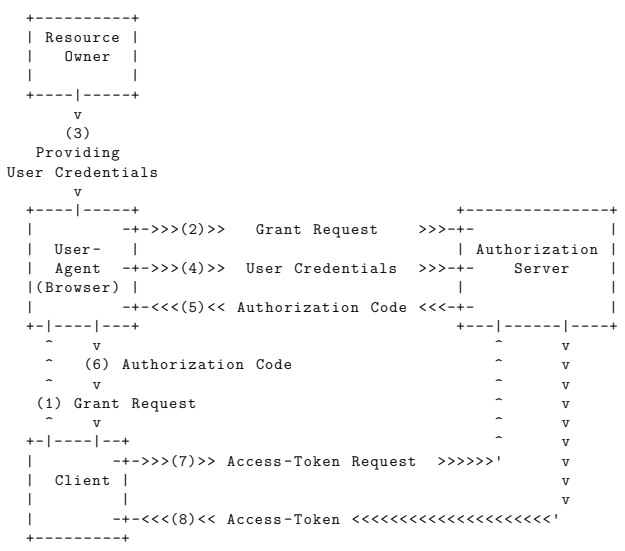

Abb. 2.2: Authorization Code Grant

Authorization Code Retrieval (1–6)

Um Zugriff auf einen Authorization Code zu erlangen, redirectet der Client den Resource Owner im ersten Schritt an den Authorization Server. Aufgabe des Authorization Servers ist es, die Identität des Clients sicherzustellen. In der Regel geschieht dies über einen Login am Authorization Server oder eine bereits bestehende Session. Ist die Identität des Resource Owners verifiziert, führt wiederum der Authorization Server einen Redirect auf die vom Client angegebene Redirect URI. Der Request des Redirects besteht aus zwei erforderlichen Query Parameter:

response_type um einen Authorization Code zu erhalten, muss der Wert ‘code’ gewählt werden.

client_id die Client ID wie in Kapitel 2.1 beschrieben. Der erhaltene Authorization Code kann nur mit dem zugehörigen Client Secret eingelöst werden.

Weitere Parameter wie redirect_uri, scope und state sind optional. Vorausgesetzt, der Grant war

```
https://localhost/authorize?
                response_type=code&
                client_id=my4Client2giz4IDb5zn
```

Abb. 2.3: Authorization Request

erfolgreich, so wird der User Agent an die bei der Client-Registrierung angegebene Redirect URI weitergeleitet, wobei der Authorization Code im URI Fragment gesetzt ist. Wie der Client auf den Parameter zugreift, ist nicht Teil der Spezifikation. Zu beachten ist, dass der Authorization Code auf diesem Weg auch für den Nutzer und im Falle eines manipulierten Browsers auch für etwaige Angreifer sichtbar ist. Um Replay-Angriffen vorzubeugen, sollte jeder Authorization Code nur ein einziges Mal genutzt werden können. Falls der Authorization Server eine erneute Nutzung desselben Authorization Codes festellt, sollten alle bereits ausgestellten Access-Tokens invalidiert werden. (vgl. Siriwardena 2014, S. 97)

```
https://myuri/#code=45273487ckdjfhjgasduz123ioujuj
```

Abb. 2.4: Authorization Response

Access-Token Retrieval (7–8)

Abschließend muss der Client den erhaltenen Authorization Code gegen einen Access-Token eintauschen. Hierzu ist ein POST auf den Access-Token- Endpoint des Authorization Servers auszuführen, der zwei erforderliche Parameter entgegennimmt.

grant_type um zwischen dem Authorization Code Grant und weiteren Grant Types wie dem Resource Owner Password Credentials Grant zu unterscheiden. Als Wert muss ‘authorization_code’ gewählt werden.

code der im vorangegangenen Schritt erhaltene Authorization Code.

Der Access-Token Endpoint sollte mindestens per HTTP Basic Authentication abgesichert werden. Client ID und Client Secret dienen als Zugangsdaten. Da die Zugangsdaten in der Basic Authentication nicht verschlüsselt übertragen werden, ist eine Übermittlung ausschließlich per HTTPS anzuraten. Nur falls der Client nicht in der Lage sein sollte, HTTP Basic Authentication auszuführen, dürfen Client ID und Client Secret im Body des Requests übertragen werden (vgl. Hardt 2012, S. 15f.).

```
curl -v -X POST --basic
     -u  my4Client2giz4IDb5zn:my32Client9btzSecret186baszdg1
     -H "Content-Type:application/x-www-form-urlencoded;charset=utf-8" -k
     -d "grant_type=authorization_code&
         code=45273487ckdjfhjgasduz123ioujuj" https://localhost/token
```

Abb. 2.5: Access-Token cURL

Auf das vorangegangene cURL-Kommando sollte der Authorization Server mit einem HTTP 200 OK und einem JSON-Body antworten. Die möglichen Token Types werden in Kapitel 2.3 näher erläutert. Neben dem Access-Token enthält das JSON Gültigkeitsdauer des Access-Tokens in Sekunden sowie den weiter oben bereits kurz erwähnten Refresh-Token, der in dieser Arbeit jedoch keine weitere Rolle spielt.

```
{
    "token_type":"bearer",
    "expires_in":"600",
    "refresh_token":"31zse4ua14fkjsd5bg39gsjnzg42zg",
    "access_token":"7c2hg718hn21sd2sz693jas6h24g5i"
}
```

Abb. 2.6: Access-Token Response

2.2.2 Implicit Grant

Anders als der Authorization Code Grant richtet sich der Implicit Grant an unsichere Clients, die über kein eigenes Backend verfügen und auch anderweitig nicht dazu in der Lage sind, ihr Client Secret zu schützen. Da Authorization Codes nur mittels Client Secret in Access Codes eingetauscht werden können, entfällt dieser Mechanismus für den Implicit Grant.

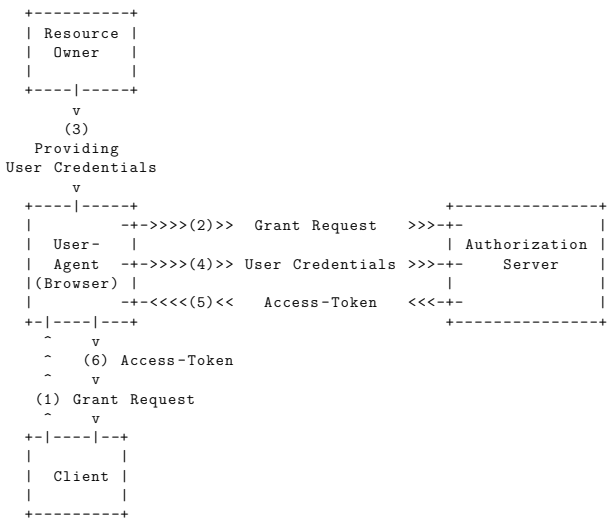

Abb. 2.7: Implicit Grant

Der Request an den Authorization Server ist in seinem Aufbau ident zu Listing 2.3. Der Wert des response_type ist auf ‘token’ anzupassen.

```
https://localhost/authorize?
                    response_type=token&
                    client_id=my4Client2giz4IDb5zn
```

Abb. 2.8: Implicit Authorization Request

Nach erfolgreicher Authentifizierung des Resource Owners in Schritt 1–4 antwortet der Authentication Server direkt mit einem Access-Token. Die Schritte 7–8 des Authorization Code Grants entfallen also. Refresh-Token, die in dieser Arbeit ohnehin keine weitere Rolle spielen, werden in Reaktion auf die geminderte Vertraulichkeit des Implicit Grants nicht mehr bereitgestellt.

```
https://myuri/#access_token=45273487ckdjfhjgasduz123ioujuj&
              expires_in=600
```

Abb. 2.9: Implicit Authorization Response

2.3 Access-Token

Bei Access-Tokens handelt es sich um serverseitig generierte, kryptographische Werte (vgl. Rohr 2018, S. 273), also zufällige Zeichenfolgen, die nur schwer erraten werden können (vgl. Petrlic & Sorge 2017, S. 72)

einer der zentralen Kritikpunkte an OAuth 2.0 ist die Ungebundenheit der in RFC 6750 spezifizierten Bearer-Tokens. Eine Alternative hierzu stellen MAC-Tokens dar.

2.3.1 Bearer-Token

Bearer-Access-Tokens funktionieren, wie zu Beginn dieses Kapitels bereits erläutert. Sie sind vollkommen unabhängig von dem Client, der sie angefordert hat. Wer sich im Besitz eines Bearer-Tokens befindet, der hat auch Zugriff auf die entsprechend geschützten Ressourcen.

2.3.2 MAC-Token

MAC Tokens, spezifiziert in dem IETF HTTP MAC Draft, verhalten sich, um die bereits genutzte Geld-Metapher wieder aufzugreifen, wie Kreditkarten. (vgl. Siriwardena 2014, S. 134) Hierzu wird dem Client ein symmetrischer Schlüssel übermittelt, mit dem er den MAC berechnet. Weitere Informationen bezüglich der Berechnung des MACs finden sich in (Siriwardena 2014, S. 136ff.) sowie in RFC 2104.

Erklären weshalb ein MAC mein Problem nicht löst. De symmetrische Key ist genau s anfällig wie alle andere. Authorization Code, Access-Token. Wird halt unverschlüsselt versendet.

Berechnung kurz

2.4 JWT und JWE

Definiert in RFC 7159 ist JavaScript Object Notation (JSON) ein Format zur Übertragung oder Speicherung von Daten, das sich in den vergangenen Jahren als Alternative zu XML etabliert hat. (vgl. Siriwardena 2014, S. 201) JSON Web Token (JWT) ist ein JSON-basiertes Container-Format für Datenübermittlung, das insbesondere als Dateiformat in OpenID Connect genutzt wird. JWTs werden in der Praxis entweder signiert (JWS) oder verschlüsselt (JWE). Im Folgenden soll ein Beispielhafter JWE erstellt werden.

2.4.1 JSON Web Encryption

JWEs bestehen, je nach Anwendungsfall aus insgesamt fünf bis sechs Attributen. JWEs mit sechs Attributen können sich an mehr als nur einen Empfänger wenden.

```
{
  "header":{"enc":"A128GCM","alg":"RSA-OAEP"},
  "encrypted_key":"G41Y2aizjBW5bFqEYWhIL84bVBbcuaHXB1szgFPEKfqJIMfcUZ
                   faeNqG2B9Cxo7Q",
  "initialization_vector":"3THSMF4n786zAUim204dhgknnJz85FkP",
  "ciphertext":"YXBKFvHo6nxQICZ3jTC1jIYhyHoDGwTDPuFzx2YgWCF39dP2Y6SFzCW24BzpiD0A",
  "tag":"gVYxhN5OVTAWcYub"
}
```

Abb. 2.10: JWE

Listing 2.10 zeigt ein fünfteiliges JWE mit den Attributen

header — enthält unverschlüsselte Metainformationen über die Verschlüsselung des Content Encryption Keys. Wichtige Elemente sind

- alg der Algorithmus zur Verschlüsselung des Content Encryption Keys
- enc der Algorithmus zur Verschlüsselung des Payloads, also der zu übermittelnden Daten.

encrypted_key — base64url-encodierter Content Encryption Key, verschlüsselt durch das in ‘alg’ spezifizierten Algorithmus.

initialization_vector — base64url-encodierter, zufällig generierter Wert, der im Rahmen mancher Ver- und Entschlüsselungsverfahren benötigt wird

ciphertext — base64url-encodiert, enthält die zu übermittelnden Daten der Nachricht. Verschlüsselt durch den Content Encryption Key nach dem in ‘enc’ spezifizierten Algorithmus.

tag

Zur Serialisierung werden die Attribute, falls sie es noch nicht sein sollten, base64url-Encodiert und mit Punkten getrennt konkateniert.

```
eyJlbmMiOiJBMTI4R0NNIiwiYWxnIjoiUlNBLU9BRVAifQ
.G41Y2aizjBW5bFqEYWhIL84bVBbcuaHXB1szgFPEKfqJIMfcUZfaeNqG2B9Cxo7Q
.3THSMF4n786zAUim204dhgknnJz85FkP
.YXBKFvHo6nxQICZ3jTC1jIYhyHoDGwTDPuFzx2YgWCF39dP2Y6SFzCW24BzpiD0A
.gVYxhN5OVTAWcYub
```

Abb. 2.11: Serialisiertes JWE

Codebeispiel in den Anhang einfügen.

Kapitel 3

Implementierung

Der in unserem Team aufgetretene Anwendungsfall involviert zwei Clients, die TK-App und den Web-Client, mit jeweils unabhängigen Backendsessions. Die TK-App ist durch Einbindung KOBIls als sicherer Client zu betrachten. Diesen Umstand zu elaborieren überschreitet jedoch den Umfang dieser Arbeit, er ist als gegeben hinzunehmen. Der Web-Client kommuniziert per HTTPS und nutzt secured-Cookies, durch seine Backendanbindung ist auch er als grundsätzlich sicher einzustufen. Bei erstmaligem Aufruf des Web-Clients wird im User-Agent des Nutzers ein neuer, noch nicht eingeloggter TKSESSION-Cookie gesetzt. Es wird angenommen, dass der Resource Owner bereits in der App eingeloggt ist. Zielsetzung ist nun, die im User-Agent des Web-Clients gesetzte Session einzuloggen. Zur Verfügung stehen die bereits aufgezählten Parteien: eingeloggte TK-App + Backend, User-Agent, Web-Client + Backend.
In ersten Überlegungen wurde die Erstellung einer neuen, eingeloggten Session in Betracht gezogen, was sich jedoch schnell als Vergehen am Loadbalancer herausstellte und daher verworfen wurde. Auch der Ansatz, die Session ID an die TK-App weiterzuleiten und in ihrem Backend einzuloggen erwies sich als undurchführbar. Um eine Session einzuloggen, muss ein Session-eigener User-Context befüllt werden, was durch eine außerhalb liegende Session unmöglich ist.
Die App selbst ist also nicht dazu in der Lage, die Web-Session einzuloggen. Nur der Web-Client selbst kann dies bewerkstelligen. Durch diese Erkenntnis kann auf eine klassische OAuth 2.0-Problemstellung reduziert werden: der Web-Client greift auf eine Login-API zu. Um diese vor unbefugten Nutzern zu schützen, setzt der Zugriff auf sie einen Access-Token voraus. Auch lässt sich in der TK- Implementierung von dem Access-Token auf den Versicherten schließen, für den er ausgestellt wurde, ein weiterer Parameter ist also nicht erforderlich.

3.1 Entkoppelte Flows

Abbildung 3.1 zeigt eine mögliche Realisierung als von der App ausgehender Implicit Flow. Sie ist die einfachste Realisierung des Anwendungsfalls. Auf den zweiten Blick, weist dieses Vorgehen aber zwei wichtige Fehler auf: zunächst dies, die einzige Verbindung zwischen App- und Web-Client, ist der in Schritt 3 übertragene Access-Token. Web-APIs können jedoch von jedem User-Agent aufgerufen werden. Es ist demnach anzunehmen, dass entsprechende APIs von Dritten angegriffen — und diese Angriffe erfolgreich sein werden, falls es einem Dritten gelingen sollte, in den Besitz eines Access-Tokens zu gelangen. Hier also, ebenfalls in Schritt 3, findet sich der zweite Fehler. Die Übertragung des Access-Tokens an den Web-Client erfolgt unverschlüsselt als Query Parameter. Durch einen korrumpierten User Agent, respektive Smartphone, könnte es einem Dritten nun möglich sein, den

Passiv-
Formuli
sind nie
erste
Wahl.
‘Es
wurde
gezeigt,
dass’ —
damit
nehmen
Sie dem
Fazit da
Gewicht
‘Aus xx
ist er-
sichtlich
dass…’

redirec-
tAPI
erklären

Aus-
führen,
weshalb
einfach.
Bloß
ein Web
Api-Call

Access-Token zu entwenden. Da der Token unverschlüsselt übermittelt wurde, kann der Dritte sich durch einen Aufruf der login/redirect-API Zugang zu einer eingeloggten Session verschaffen.

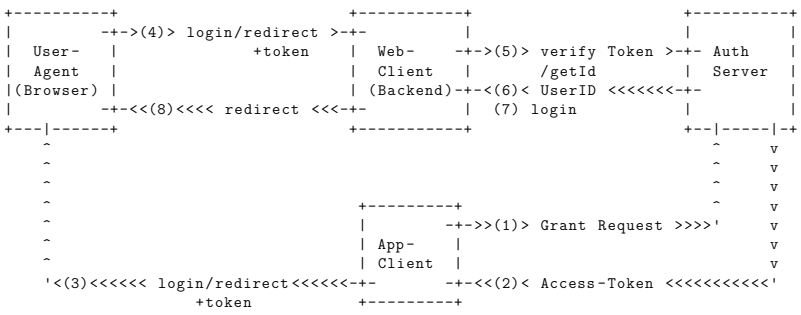

Abb. 3.1: Anwendungsfall, Implicit Login Flow

Eine erste Verbesserung wäre es, in Schritt 3 statt des Access-Tokens einen Authorization Code zu versenden. Im Backend des Web-Clients ist die Umwandlung des Authorization Codes in einen Access-Token von nur geringem zusätzlichem Aufwand. Die ausgestellten Authorization Codes sollten in jedem Fall nur einmalige Gültigkeit besitzen, bei erneuter Nutzung sollten alle eingeloggten Sessions des Resource Owners invalidiert werden, um Replayangriffe auszuschließen. Abbildung 3.1 zeigt den entsprechenden Flow.

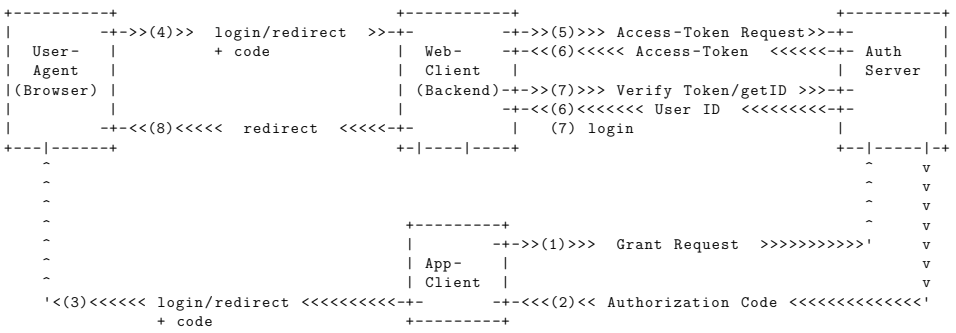

Abb. 3.2: Anwendungsfall, Authorization Code

Die Entwendung des Parameters in Schritt 3 wird auf diese Weise zwar nicht verhindert, dadurch dass es sich jedoch nur um einen einmalig gültigen Authorization handelt, ist der Wert eines Diebstahls erheblich vermindert. Um den Login eines Resource Owners in diesem Flow zu stehlen, wäre es notwendig, die Verbindung des User-Agents zu der Web-API zu unterbinden. In einem korrumpierten User-Agent, respektive Smartphone, wäre dies zwar wieder denkbar, dass eine Verbindung vollkommen blockiert wird, wirkt jedoch noch unwahrscheinlicher als der bloße Diebstahl übermittelter Daten. Davon abgesehen, einen geschickten Angriff dieser Art würde der Resource Owner nicht wahrnehmen, der korrumpierte User-Agent könnte ihn beispielsweise auf die Startseite der TK redirecten. Dass Resource Owner-Identitäten auch in diesem Flow gestohlen werden können, liegt daran, dass er keine Lösung für das erstgenannte Problem liefert: zwischen App- und Web-Client existiert keine Kopplung. Jede Websession ist dazu in der Lage, sich einzuloggen. Dabei ist der Parameter vollkommen egal, auch ein MAC-Token, wie in Abschnitt 2.3 erläutert, kann dieses Problem nicht lösen, denn jede beliebige

Anmerken, gegen welche Regel es verstößt Access-Tokens zu übertragen. Erklären weshalb 2FA so nicht möglich ist

Literatu anführen

Session wäre zu seiner, stets identischen, Berechnung imstande.

3.2 Gekoppelte Flows

Eine Möglichkeit, das Problem der Unabhängigkeit von Web-Session und App-Client aufzulösen, ist es, JWEs zur Übermittlung eines Authorization Codes oder eines Access-Tokens zu nutzen. Abbildung 3.3 zeigt den Flow.

Spell-check erkennt doppelte Leer-zeichen nicht mehr?!

(1) propose/redirect — Aufruf einer API, die feststellt, ob die genutzte Session bereits eingeloggt ist. Als Parameter erhält sie ausschließlich den entsprechend redirect-Key. Falls die Session eingeloggt ist, wird der angefragte Redirect ausgeführt.

(2–4) Key proposal — Ist die Session jedoch ausgeloggt, so erstellt das Web-Backend ein neues asymmetrisches Schlüsselpaar. Der Private Key wird entweder direkt in der Session gespeichert oder in einer Datenbank mit der Session assoziiert und persistiert. Der Public Key muss in einer Datenbanktabelle gespeichert werden. Das ist notwendig, um in Schritt 5 zu verifizieren, dass der übermittelte Public Key auch tatsächlich im Backend erstellt wurde. Ohne diese Überprüfung könnten Drittanwendungen der App in Schritt 4 eigene, selbst generierte Public Keys übergeben. Abschließend wird der Public Key an den App-Client zurückgeleitet.

(5–6) Access-Token und JWE — Der App-Client nimmt den Public Key entgegen und leitet ihn an sein Backend weiter, indem er eine Protected API aufruft. Diese API kontaktiert den überprüft im ersten Schritt, ob der Public Key in der Datenbank eingetragen ist. Anschließend kontaktiert sie den Authorization Server und lässt sich einen Access-Token ausstellen. Anschließend schreibt sie dieses Token in ein JWE, verschlüsselt und serialisiert es. Das so abgesicherte JWE wird an den App-Client redirectet.

(7–8) JWE Weiterleitung — Der App-Client leitet das JWE weiter an den User-Agent und dieser an das Web-Client-Backend weiter. Dazu wird eine API aufgerufen und das serialisierte JWE als Query Parameter angefügt.

(9) JWE Entschlüsselung — Der in Schritt 2–4 in der Session bzw. einer Datenbank gespeicherte Private Key wird geladen. Anschließend wird das JWE entschlüsselt und der mitgelieferte Access-Token ausgelesen.

(10–11) User Identifizierung — Der in Schritt 9 ausgelesene Access-Token wird an den Authorization Server geleitet und von diesem verifiziert. Falls der Token valide ist,

wird aus mithilfe des Access-Token die User-ID des Resource Owners ermittelt und dem Web-Client übergeben

(12–13) Login & Redirect — Anhand der User-ID wird die Session eingeloggt und an die ursprünglich angefragte Seite redirectet.

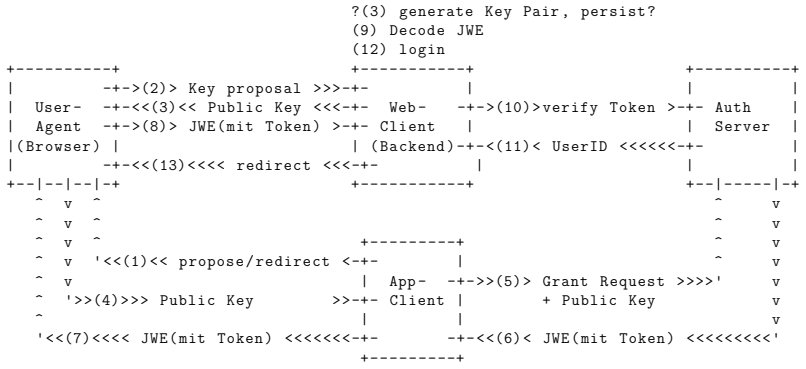

Abb. 3.3: Anwendungsfall, Implicit Login Flow

Um das Problem zu lösen muss eine Kopplung zw. App und Web entstehen, die aber vom Web ausgeht. Momentan geht diese Kopplung ja irgendwie so halb von der App aus, die App ruft das Web auf, eigentlich müsste das Web die App aufrufen.
Es muss sichergestellt sein, dass die API, nur aus der App aufgerufen werden kann. Das ist aber möglich, da die App eigene eigene accessTokens nutzt

JWE Erklärungen XReferenziere

Entscheiden, ob nach items groß oder klein geschrieben werden soll

Entscheiden, ob User ID User-ID oder User-ID

Entscheiden, ob Resource Owner, User oder Versicherter

Entscheiden, ob und oder &

Flows überarbeiten, gerade den letzten Flow!

Kapitel 4

Fazit

Anhang A

Anhang

Literaturverzeichnis

Chari, S., Jutla, C. S. & Roy, A. (2011), 'Universally composable security analysis of oauth v2.0', IACR Cryptology ePrint Archive 2011, 526.
URL: http://eprint.iacr.org/2011/526

Hardt, D. (2012), The oauth 2.0 authorization framework, RFC 6749, RFC Editor.
URL: https://tools.ietf.org/html/rfc6749

Petrlic, R. & Sorge, C. (2017), Datenschutz - Einführung in technischen Datenschutz, Datenschutzrecht und angewandte Kryptographie, Springer Vieweg, Wiesbaden.

Rohr, M. (2018), Sicherheit von Webanwendungen in der Praxis, 2 edn, Springer Vieweg, Wiesbaden.

Siriwardena, P. (2014), Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE, 1st edn, Apress, Berkely, CA, USA.

Selbständigkeitserklärung

Ich versichere hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig angefertigt und ohne fremde Hilfe verfasst habe, keine außer den von mir angegebenen Hilfsmitteln und Quellen dazu verwendet habe und die den benutzten Werken inhaltlich und wörtlich entnommenen Stellen als solche kenntlich gemacht habe.

Hamburg, den 29. Dezember 2018