Angreifer Model und Schutzmaßnahmenkatalog für den Schutzbedarf Normal

> 👎 👍 → Feedback und Anmerkungen zu diesem Dokument geben

Zusammenfassung

Das vorliegende Dokument definiert das Angreifer Model und den Schutzmaßnahmenkatalog für den Schutzbedarf Normal auf Basis des "Best Current Practice for OAuth 2.0 Security" [RFC9700] Dokuments.

Einleitung

Das vorliegende Dokument ist ein Begleitdokument für den Schutzbedarf Normal, abgeleitet aus "Best Current Practice for OAuth 2.0 Security" [RFC9700]. Es umfasst Kapitel 3 und 4 aus [RFC9700] und beschreibt somit das Angreifer Model und den Schutzmaßnahmenkatalog für den Schutzbedarf Normal.

Kapitel 3 stellt das aktualisierte OAuth-Angreifer-Modell vor. Kapitel 4 enthält eine detaillierte Analyse der Bedrohungen und Implementierungsprobleme, die (zum Zeitpunkt der Erstellung dieses Dokuments) in der Praxis anzutreffen sind, sowie eine Erörterung möglicher Gegenmaßnahmen.

Es wird empfohlen das vorliegende Dokument zu nutzen, um die Umsetzung von Anforderungen besser zu bewerten und die IT-Sicherheit von API-Absicherungen weiter zu erhöhen.

Schlüsselwörter

Die Schlüsselwörter „MUSS“ (DARF NUR), „DARF NICHT“, „SOLLTE“, „SOLLTE NICHT“, „DARF“ und „KANN“ in diesem Dokument sind gemäß DIN-Norm DIN 820-2 - 2022-12 [DIN 820-2] zu interpretieren. Diese Schlüsselwörter werden nicht als Wörterbuchbegriffe verwendet, sodass jedes Vorkommen als Schlüsselwort zu interpretieren ist und nicht mit ihrer natürlichen Sprachbedeutung.

Normative Verweise

Auf die folgenden Dokumente wird im Text in einer Weise verwiesen, dass ihr Inhalt ganz oder teilweise Anforderungen dieses Dokuments darstellen. Bei datierten Verweisen gilt nur die zitierte Ausgabe. Bei undatierten Verweisen gilt die neueste Ausgabe des referenzierten Dokuments (einschließlich aller Änderungen).

Die normativen Verweise sind in Kapitel 5.1 aufgeführt.

Weiterführende, nicht normative Referenzen sind in Kapitel 5.2 zusammengestellt.

1. Begriffe und Definitionen

Für die Zwecke dieses Dokuments gelten die in [RFC6749], [RFC6750], [RFC7636], [OIDC] und [ISO29100] definierten Begriffe. Zum besseren Verständnis werden im vorliegenden Dokument OAuth und OpenID Connect spezifische technische Begriffe in ihrem englischen Terminus verwendet.

2. Symbole und Abkürzungen

API – Application Programming Interface

BCM – Basin, Cremers, Meier

BCP – Best Current Practice

CAA – Certificate Authority Authorization

CIBA – Client Initiated Backchannel Authentication

CSRF – Cross-Site Request Forgery

DNS – Domain Name System

DNSSEC – Domain Name System Security Extensions

DPoP – Demonstrating Proof-of-Possession

HTTP – Hyper Text Transfer Protocol

IETF – Internet Engineering Task Force

JAR – JWT-Secured Authorization Request

JARM – JWT-Secured Authorization Response Mode

JWK – JSON Web Key

JWKS – JSON Web Key Sets

JWT – JSON Web Token

JOSE – JavaScript Object Signing and Encryption

JSON – JavaScript Object Notation

MTLS – Mutual Transport Layer Security

OIDF – OpenID Foundation

PAR – Pushed Authorization Requests

PKCE – Proof Key for Code Exchange

QR – Quick Response

RSA – Rivest-Shamir-Adleman

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

TLS – Transport Layer Security

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

3. Das aktualisierte OAuth 2.0-Angreifer-Modell

In [RFC6819] wird ein Bedrohungsmodell dargelegt, das die Bedrohungen beschreibt, gegen die OAuth-Implementierungen geschützt werden müssen. Dabei trifft [RFC6819] bestimmte Annahmen über Angreifer und deren Fähigkeiten, d. h. es impliziert ein Angreifer-Modell. Im Folgenden wird dieses Angreifer-Modell explizit gemacht und aktualisiert und erweitert, um die potenziell dynamischen Beziehungen zwischen mehreren Parteien zu berücksichtigen, neue Arten von Angreifern einzubeziehen und das Angreifer-Modell klarer zu definieren.

Das Ziel dieses Dokuments ist es, sicherzustellen, dass die Autorisierung eines Resource Owners (mit einem Benutzeragenten) bei einem Authorization Server und die anschließende Verwendung des Access Tokens bei einem Resource Server so weit wie praktisch möglich zumindest vor den folgenden Angreifern geschützt sind.

(A1)

Web-Angreifer, die eine beliebige Anzahl von Netzwerk-Endpunkten (neben den „ehrlichen") einrichten und betreiben können, darunter Browser und Server. Web-Angreifer können Websites einrichten, die vom Resource Owner besucht werden, ihre eigenen Benutzeragenten betreiben und am Protokoll teilnehmen.

Insbesondere können Web-Angreifer OAuth-Clients betreiben, die beim Authorization Server registriert sind, und sie können ihre eigenen Autorisierungs- und Resource Server betreiben, die (parallel zu den „ehrlichen") vom Resource Owner und anderen Resource Owners genutzt werden können.

Es muss auch davon ausgegangen werden, dass Web-Angreifer den Benutzer jederzeit dazu verleiten können, mit seinem Browser zu beliebigen, vom Angreifer gewählten URIs zu navigieren. In der Praxis kann dies auf vielfältige Weise erreicht werden, beispielsweise durch das Einfügen bösartiger Werbung in Werbenetzwerke oder durch das Versenden von legitim aussehenden E-Mails.

Web-Angreifer können ihre eigenen Benutzeranmeldedaten verwenden, um neue Nachrichten zu erstellen, sowie alle Geheimnisse, die sie zuvor erfahren haben. Wenn ein Web-Angreifer beispielsweise einen Code für die Autorisierung eines Benutzers durch eine falsch konfigurierte Redirection URI erfährt, kann der Web-Angreifer versuchen, diesen Code gegen ein Access Token einzulösen.

Sie können jedoch keine Nachrichten lesen oder manipulieren, die nicht an sie gerichtet sind (z. B. an eine URL eines Authorization Server gesendet, der nicht unter der Kontrolle eines Angreifers steht).

(A2)

Netzwerkangreifer, die zusätzlich die vollständige Kontrolle über das Netzwerk haben, über das die Protokollteilnehmer kommunizieren. Sie können Nachrichten abhören, manipulieren und fälschen, es sei denn, diese sind durch kryptografische Methoden (z. B. TLS) geschützt. Netzwerkangreifer können auch beliebige Nachrichten blockieren.

Während ein Beispiel für einen Webangreifer ein Kunde eines Internetdienstanbieters wäre, könnten Netzwerkangreifer beispielsweise der Internetdienstanbieter selbst, ein Angreifer in einem öffentlichen (Wi-Fi-)Netzwerk, der ARP-Spoofing verwendet, oder ein staatlich geförderter Angreifer mit Zugang zu Internetknotenpunkten sein.

Die oben genannten Angreifer (A1) und (A2) entsprechen dem Angreifer-Modell, das in formalen Analysen für OAuth verwendet wurde [arXiv.1601.01229]. Dies ist ein minimales Angreifer-Modell. Entwickler MÜSSEN alle möglichen Arten von Angreifern in der Umgebung ihrer OAuth-Implementierungen berücksichtigen. In [arXiv.1901.11520] wird beispielsweise ein sehr starkes Angreifer-Modell verwendet, das Angreifer umfasst, die die vollständige Kontrolle über den Token-Endpunkt haben. Dieses Modell simuliert die Auswirkungen einer möglichen Fehlkonfiguration von Endpunkten im Ökosystem, die durch die Verwendung von Authorization Server Metadaten, wie in den Sicherheitsvorgaben für die Absicherung von APIs für den Schutzbedarf Normal beschrieben, vermieden werden kann. Ein solcher Angreifer wird daher hier nicht aufgeführt.

Frühere Angriffe auf OAuth haben jedoch gezeigt, dass insbesondere die folgenden Arten von Angreifern relevant sind:

(A3)

Angreifer, die den Inhalt der Autorisierungsantwort lesen, aber nicht ändern können (d. h. die Autorisierungsantwort kann an einen Angreifer gelangen).

Beispiele für solche Angriffe sind Open-Redirector-Angriffe und Mix-up-Angriffe (siehe Kapitel 4.4), bei denen der Client dazu verleitet wird, Anmeldedaten an einen vom Angreifer kontrollierten Authorization Server zu senden.

Dazu gehören auch Angriffe, die folgende Schwachstellen ausnutzen:

- unzureichende Überprüfung von Redirection URI-Adressen (siehe Kapitel 4.1);
- Probleme bei mobilen Betriebssystemen, bei denen sich verschiedene Apps unter derselben URI registrieren können; und
- URLs, die von Browsern (Verlauf), Proxy-Servern und Betriebssystemen gespeichert/protokolliert werden.

(A4)

Angreifer, die den Inhalt der Autorisierungsanfrage lesen, aber nicht ändern können (d. h. die Autorisierungsanfrage kann auf die gleiche Weise wie oben beschrieben an einen Angreifer weitergegeben werden).

(A5)

Angreifer, die ein von einem Authorization Server ausgestelltes Access Token erwerben können. Beispielsweise kann ein Resource Server von einem Angreifer kompromittiert werden, ein Access Token aufgrund einer Fehlkonfiguration an einen vom Angreifer kontrollierten Resource Server gesendet werden oder Social Engineering eingesetzt werden, um einen Resource Owner dazu zu bringen, einen dann vom Angreifer kontrollierten Resource Server zu verwenden. Siehe auch Kapitel 4.9.2.

(A3), (A4) und (A5) treten in der Regel zusammen mit entweder (A1) oder (A2) auf. Angreifer können zusammenarbeiten, um ein gemeinsames Ziel zu erreichen.

Es ist zu beachten, dass ein Angreifer (A1) oder (A2) ein Resource Owner sein oder als solcher agieren kann. Ein solcher Angreifer kann beispielsweise seinen eigenen Browser verwenden, um Token oder

Autorisierungscodes zu wiederholen, die durch einen der oben beschriebenen Angriffe auf dem Client oder dem Resource Server erlangt wurden.

Dieses Dokument konzentriert sich auf Bedrohungen, die sich aus den Angreifern (A1) bis (A5) ergeben.

4. Angriffe und Abwehrmaßnahmen

Dieser Abschnitt enthält eine detaillierte Beschreibung von Angriffen auf OAuth-Implementierungen sowie mögliche Gegenmaßnahmen. Angriffe und Abwehrmaßnahmen, die bereits in [RFC6819] behandelt wurden, werden hier nicht aufgeführt, es sei denn, es werden neue Empfehlungen ausgesprochen.

In diesem Abschnitt werden zusätzliche Anforderungen (über die in den Sicherheitsvorgaben für die Absicherung von APIs für den Schutzbedarf Normal definierten hinaus) für bestimmte Fälle und Protokolloptionen definiert.

4.1. Unzureichende Validierung der Redirection URI

Einige Authorization Server ermöglichen es Clients, Redirection URI-Patterns anstelle vollständiger Redirection URIs zu registrieren. Die Authorization Server gleichen dann den Wert des Redirection URI-Parameters am Redirection Endpoint mit den registrierten Mustern zur Laufzeit ab. Dieser Ansatz ermöglicht es Clients, den Transaktionsstatus in zusätzliche Redirection-URI-Parameter zu kodieren oder ein einziges Muster für mehrere Redirection URIs zu registrieren.

Dieser Ansatz erwies sich als komplexer in der Implementierung und fehleranfälliger in der Verwaltung als die exakte Übereinstimmung der Redirection URI. In der Praxis wurden mehrere erfolgreiche Angriffe beobachtet, die Schwachstellen in der Implementierung der Mustererkennung oder in konkreten Konfigurationen ausnutzten (siehe z. B. [research.rub2]). Eine unzureichende Validierung der Redirection URI unterbricht effektiv die Client-Identifizierung oder -Authentifizierung (abhängig vom Grant-Type und dem Client-Typ) und ermöglicht es dem Angreifer, einen Autorisierungscode oder ein Access Token zu erhalten, entweder

- durch direktes Senden des Benutzeragenten an eine URI unter der Kontrolle des Angreifers oder
- durch Offenlegung der OAuth-Anmeldedaten gegenüber einem Angreifer unter Verwendung eines offenen Redirects auf dem Client in Verbindung mit der Art und Weise, wie Benutzeragenten URL-Fragmente verarbeiten.

Diese Angriffe werden in den folgenden Unterabschnitten detailliert beschrieben.

4.1.1. Angriffe auf die Validierung der Redirection URI beim Authorization Code Grant

Bei einem Client, der den Grant-Type `code` verwendet, kann ein Angriff wie folgt ablaufen:

Angenommen, das Redirection URI-Muster `https://*.somesite.example/*` ist für den Client mit der Client-ID `s6BhdRkqt3` registriert. Die Absicht besteht darin, jede Subdomain von `somesite.example` als gültige Redirection URI für den Client zuzulassen, beispielsweise `https://app1.somesite.example/redirect`. Eine naive Implementierung auf dem Authorization Server könnte jedoch den Platzhalter `*` als "beliebiges Zeichen" und nicht als "beliebiges Zeichen, das für einen Domainnamen gültig ist" interpretieren. Der Authorization Server könnte daher `https://attacker.example/.somesite.example` als Redirection URI zulassen, obwohl `attacker.example` eine andere Domain ist, die möglicherweise von einer böswilligen Partei kontrolliert wird.

Der Angriff kann dann wie folgt durchgeführt werden:

Zunächst muss der Angreifer den Benutzer dazu verleiten, eine manipulierte URL in seinem Browser zu öffnen, die eine Seite unter der Kontrolle des Angreifers startet, beispielsweise `https://www.evil.example` (siehe Angreifer A1 in Kapitel 3).

Diese URL initiiert die folgende Autorisierungsanfrage mit der Client-ID eines legitimen Clients an den Autorisierungsendpunkt (Zeilenumbrüche nur zur Anzeige):

```
GET /authorize?
response_type=code&client_id=s6BhdRkqt3&state=9ad67f13&redirect_uri=https%3A%2F%2F
attacker.example%2F.somesite.example HTTP/1.1
Host: server.somesite.example
```

Der Authorization Server validiert die Redirection URI und vergleicht sie mit den registrierten Redirect-URI-Patterns für den Client `s6BhdRkqt3`. Die Autorisierungsanfrage wird verarbeitet und dem Benutzer angezeigt.

Wenn der Benutzer die Redirection URI nicht sieht oder den Angriff nicht erkennt, wird der Code ausgegeben und sofort an die Domain des Angreifers gesendet. Wenn eine automatische Genehmigung der Autorisierung aktiviert ist (was laut [RFC6749] für Public Clients nicht empfohlen wird), kann der Angriff auch ohne Benutzerinteraktion durchgeführt werden.

Wenn sich der Angreifer als Public Client ausgibt, kann er den Code am entsprechenden Token-Endpunkt gegen ein Token eintauschen.

Bei Confidential Clients funktioniert dieser Angriff nicht so einfach, da der Codeaustausch eine Authentifizierung mit dem geheimen Schlüssel des legitimen Clients erfordert. Der Angreifer kann jedoch den legitimen Confidential Client nutzen, um den Code einzulösen, indem er einen Autorisierungscode-Injection-Angriff durchführt; siehe Kapitel 4.5.

Es ist wichtig zu beachten, dass Schwachstellen bei der Validierung von Redirection URIs auch dann bestehen können, wenn der Authorization Server Wildcards ordnungsgemäß verarbeitet. Angenommen, der Client registriert das Redirection URI-Pattern `https://*.somesite.example/*` und der Authorization Server interpretiert dies als „Redirection URIs zulassen, die auf einen beliebigen Host in der Domäne `somesite.example` verweisen". Wenn es einem Angreifer gelingt, einen Host oder eine Subdomain in `somesite.example` einzurichten, kann er sich als legitimer Client ausgeben. Dies könnte durch einen Subdomain-Übernahmeangriff [research.udel] verursacht werden, bei dem ein veralteter CNAME-Eintrag (z. B. `external-service.somesite.example`) auf einen externen DNS-Namen verweist, der nicht mehr existiert (z. B. `customer-abc.service.example`) und von einem Angreifer übernommen werden kann (z.B. durch Registrierung als `customer-abc` beim externen Dienst).

4.1.2. Angriffe auf die Validierung von Redirection URI bei Implicit Grant

> **Hinweis:** Der Implicit Grant ist gemäß Kapitel 5.1 des Begleitdokuments „Föderale Sicherheitsvorgaben für den Schutzbedarf Normal" verboten und wird mit OAuth 2.1 entfernt. Die folgende Beschreibung dient ausschließlich der Nachvollziehbarkeit dieses Verbots und der Vollständigkeit des Angreifermodells.

Der oben beschriebene Angriff funktioniert auch bei Implicit Grant. Wenn der Angreifer in der Lage ist, die Autorisierungsantwort an eine von ihm kontrollierte URI zu senden, erhält er direkten Zugriff auf das Fragment, das das Access Token enthält.

Darüber hinaus können Implicit Grants (und auch andere Grants bei Verwendung von `response_mode=fragment`, wie in [OAuth.Responses] definiert) einer weiteren Art von Angriff ausgesetzt sein. Der Angriff nutzt die Tatsache aus, dass Benutzeragenten Fragmente wieder an die Ziel-URL einer Umleitung anhängen, wenn der Location-Header kein Fragment enthält (siehe Kapitel 17.11 von [RFC9110]). Der hier beschriebene Angriff kombiniert dieses Verhalten mit dem Client als offenem Redirector (siehe Kapitel 4.11.1), um Access Token zu erhalten. Dies ermöglicht die Umgehung selbst sehr enger Redirection URI-Patterns, jedoch nicht einer strikten URL-Übereinstimmung.

Angenommen, das registrierte URL-Pattern für den Client `s6BhdRkqt3` lautet `https://client.somesite.example/cb?*`, d. h., jeder Parameter ist für Weiterleitungen zu `https://client.somesite.example/cb` zulässig. Leider setzt der Client einen offenen Redirector ein. Dieser Endpunkt unterstützt einen Parameter `redirect_to`, der eine Ziel-URL entgegennimmt und den Browser mithilfe eines HTTP-Location-Headers Redirect 303 zu dieser URL weiterleitet.

Der Angriff kann nun wie folgt durchgeführt werden:

Zunächst muss der Angreifer, wie oben beschrieben, den Benutzer dazu verleiten, eine manipulierte URL in seinem Browser zu öffnen, die eine Seite unter der Kontrolle des Angreifers startet, beispielsweise `https://www.evil.example`.

Anschließend initiiert die Website eine Autorisierungsanfrage, die derjenigen im Angriff auf den Code-Flow sehr ähnlich ist. Anders als oben nutzt sie den offenen Redirect, indem sie `redirect_to=https://attacker.example` in die Parameter der Redirection URI eincodiert, und sie verwendet den Antworttyp `token`:

```
GET /authorize?
response_type=token&state=9ad67f13&client_id=s6BhdRkqt3&redirect_uri=https%3A%2F%2
Fclient.somesite.example%2Fcb%26redirect_to%253Dhttps%253A%252F%252Fattacker.examp
le%252F
HTTP/1.1
Host: server.somesite.example
```

Da die Redirection URI mit dem registrierten Pattern übereinstimmt, lässt der Authorization Server die Anfrage zu und sendet das resultierende Access Token in einer 303-Umleitung (einige Antwortparameter wurden aus Gründen der Lesbarkeit weggelassen):

```
HTTP/1.1 303 See Other
Location: https://client.somesite.example/cb?
redirect_to%3Dhttps%3A%2F%2Fattacker.example%2Fcb#access_token=2YotnFZFEjr1zCsicMW
pAA&...
```

Bei `client.somesite.example` trifft die Anfrage beim offenen Redirector ein. Der Endpunkt liest den Umleitungsparameter und gibt einen HTTP 303 Location-Header-Redirect an die URL

`https://attacker.example/` aus.

```
HTTP/1.1 303 See Other
Location: https://attacker.example/
```

Da der Redirector bei `client.somesite.example` kein Fragment im Location-Header enthält, hängt der User-Agent das ursprüngliche Fragment `#access_token=2YotnFZFEjr1zCsicMWpAA&...` wieder an die URL an und navigiert zur folgenden URL:

```
https://attacker.example/#access_token=2YotnFZFEjr1z...
```

Die Seite des Angreifers auf `attacker.example` kann dann auf das Fragment zugreifen und das Access Token erhalten.

4.1.3. Gegenmaßnahmen

Die Komplexität der korrekten Implementierung und Verwaltung von Musterabgleichen verursacht offensichtlich Sicherheitsprobleme. Dieses Dokument empfiehlt daher, die erforderliche Logik und Konfiguration durch die Verwendung eines exakten Redirection-URI-Abgleichs zu vereinfachen. Das bedeutet, dass der Authorization Server sicherstellen MUSS, dass die beiden URIs identisch sind; siehe Kapitel 6.2.1 von [RFC3986], „Simple String Comparison" (Einfacher String-Vergleich). Die einzige Ausnahme bilden native Apps, die eine `localhost`-URI verwenden: In diesem Fall MUSS der Authorization Server variable Portnummern zulassen, wie in Kapitel 7.3 von [RFC8252] beschrieben.

Zusätzliche Empfehlungen:

- Webserver, auf denen Redirection URI gehostet werden, DÜRFEN KEINE offenen Redirects nutzen (siehe Kapitel 4.11).
- Browser fügen URL-Fragmente nur dann wieder an Umleitungs-URLs im Location-Header an, wenn die URL im Location-Header noch kein Fragment enthält. Daher KÖNNEN Server verhindern, dass Browser Fragmente wieder an Umleitungs-URLs anhängen, indem sie einen beliebigen Fragmentidentifikator, z. B. `#_`, an URLs in Location-Headern anhängt.
- Clients SOLLTEN den Antworttyp `code` anstelle von Antworttypen verwenden, die die Ausgabe von Access Token am Autorisierungsendpunkt verursachen. Dies bietet Gegenmaßnahmen gegen die Wiederverwendung von geleakten Anmeldedaten durch den Austauschprozess mit dem Authorization Server und gegen die Wiederholung von Token durch das Sender-Constraining der Access Token.

Wenn die Herkunft und Integrität der Autorisierungsanfrage, die die Redirection URI enthält, überprüft werden kann, kann der Authorization Server beispielsweise bei Verwendung von [RFC9101] oder [RFC9126] mit Client-Authentifizierung, der Redirection URI ohne weitere Überprüfungen vertrauen.

4.2. Weitergabe von Anmeldedaten über Referer-Header

Der Inhalt der Autorisierungsanfrage-URI oder der Autorisierungsantwort-URI kann unbeabsichtigt über den Referer-HTTP-Header (siehe Kapitel 10.1.3 von [RFC9110]) an Angreifer weitergegeben werden, indem er entweder von der Website des Authorization Servers oder der Website des Clients geleakt ist. Vor allem

können auf diese Weise Autorisierungscodes oder Statuswerte offengelegt werden. Obwohl in Kapitel 10.1.3 von [RFC9110] anders angegeben, kann aufgrund von Problemen bei der Browserimplementierung dasselbe mit Access Token geschehen, die in URI-Fragmenten übertragen werden, wie ein (mittlerweile behobenes) Problem im Chromium-Projekt [bug.chromium] zeigt.

4.2.1. Leakage aus dem OAuth-Client

Eine Leakage aus dem OAuth-Client setzt voraus, dass der Client als Ergebnis einer erfolgreichen Autorisierungsanfrage eine Seite, die

- Links zu anderen Seiten unter der Kontrolle des Angreifers enthält, und ein Benutzer auf einen solchen Link klickt, oder
- Inhalte von Drittanbietern (Werbung in iframes, Bilder usw.) enthält, beispielsweise wenn die Seite benutzergenerierte Inhalte (Blog) enthält.

Sobald der Browser zur Seite des Angreifers navigiert oder die Inhalte von Drittanbietern lädt, erhält der Angreifer die Autorisierungsantwort-URL und kann `code` oder `state` (und möglicherweise `access_token`) extrahieren.

4.2.2. Leakage vom Authorization Server

In ähnlicher Weise kann ein Angreifer den `state` aus der Autorisierungsanfrage erfahren, wenn der Autorisierungsendpunkt auf dem Authorization Server wie oben Links oder Inhalte von Drittanbietern enthält.

4.2.3. Konsequenzen

Ein Angreifer, der über einen Referer-Header einen gültigen Code oder Access Token ermittelt, kann die in den Kapiteln 4.1.1, 4.5 und 4.6 beschriebenen Angriffe durchführen. Wenn der Angreifer den Status ermittelt, geht der durch die Verwendung des Status erzielte CSRF-Schutz verloren, was zu CSRF-Angriffen führt, wie in Kapitel 4.4.1.8 von [RFC6819] beschrieben.

4.2.4. Gegenmaßnahmen

Die als Ergebnis der OAuth-Autorisierungsantwort gerenderte Seite und der Autorisierungsendpunkt SOLLTEN KEINE Ressourcen von Drittanbietern oder Links zu externen Websites enthalten.

Die folgenden Maßnahmen verringern die Wahrscheinlichkeit eines erfolgreichen Angriffs weiter:

- Unterdrücken Sie den Referer-Header, indem Sie eine geeignete Referrer-Richtlinie [W3C.webappsec-referrer -policy] auf das Dokument anwenden (entweder als Teil des Meta-Attributs „referrer“ oder durch Setzen eines Referrer-Policy-Headers). Beispielsweise unterdrückt der Header Referrer-Policy: no-referrer in der Antwort den Referer-Header in allen Anfragen, die von dem resultierenden Dokument stammen, vollständig.
- Verwenden Sie einen Autorisierungscode anstelle von Antworttypen, die die Ausgabe von Access Token vom Autorisierungsendpunkt verursachen.
- Binden Sie den Autorisierungscode an einen Confidential Client oder eine PKCE-Challenge. In diesem Fall fehlt dem Angreifer das Geheimnis, um den Codeaustausch anzufordern.
- Wie in Kapitel 4.1.2 von [RFC6749] beschrieben, MÜSSEN Autorisierungscodes nach ihrer ersten Verwendung am Token-Endpunkt vom Authorization Server ungültig gemacht werden. Wenn

beispielsweise ein Authorization Server den Code ungültig macht, nachdem der legitime Client ihn eingelöst hat, kann der Angreifer diesen Code später nicht mehr austauschen.
- Dies mindert den Angriff nicht, wenn es dem Angreifer gelingt, den Code vor dem legitimen Client gegen ein Token auszutauschen. Daher empfiehlt [RFC6749] weiter, dass der Authorization Server bei einem Versuch, einen Code zweimal einzulösen, alle zuvor auf der Grundlage dieses Codes ausgestellten Token widerrufen SOLLTE.
- Der `state` SOLLTE vom Client nach seiner ersten Verwendung am Redirection Endpoint ungültig gemacht werden. Wenn dies implementiert ist und ein Angreifer über den Referer-Header von der Website des Clients ein Token erhält, wurde der `state` bereits verwendet, vom Client ungültig gemacht und kann vom Angreifer nicht erneut verwendet werden. (Dies hilft nicht, wenn der `state` von der Website des Authorization Servers geleakt ist, da der `state` dann noch nicht am Redirection Endpoint beim Client verwendet wurde.)
- Verwenden Sie für die Autorisierungsantwort den Form-POST-Antwortmodus anstelle eines Redirects (siehe [OAuth.Post]).

4.3. Verlust von Anmeldedaten über den Browserverlauf

Autorisierungscodes und Access Token können im Browserverlauf der besuchten URLs landen, wodurch die folgenden Angriffe möglich werden.

4.3.1. Autorisierungscode im Browserverlauf

Wenn ein Browser aufgrund einer Umleitung vom Autorisierungsendpunkt eines Anbieters zu `client.example/redirection_endpoint?code=abcd` navigiert, kann die URL mit dem Autorisierungscode im Browserverlauf landen. Ein Angreifer mit Zugriff auf das Gerät könnte den Code abgreifen und versuchen, ihn wiederholt zu verwenden.

Gegenmaßnahmen:

- Verhindern der Wiederholung von Codes für die Autorisierung, wie in Kapitel 4.4.1.1 von [RFC6819] und Kapitel 4.5 beschrieben.
- Verwenden Sie für die Autorisierungsantwort den Form-POST-Antwortmodus anstelle eines Redirects (siehe [OAuth.Post]).

4.3.2. Access Token im Browserverlauf

Ein Access Token kann im Browserverlauf landen, wenn ein Client oder eine Website, die bereits über ein Token verfügt, absichtlich zu einer Seite wie `provider.com/get_user_profile?access_token=abcdef` navigiert. [RFC6750] rät von dieser Vorgehensweise ab und empfiehlt die Übertragung von Tokens über einen Header, In der Praxis übergeben Websites jedoch häufig Access Token in Abfrageparametern.

Im Falle von Implicit Grant kann eine URL wie `client.example/redirection_endpoint#access_token=abcdef` ebenfalls im Browserverlauf landen, wenn eine Weiterleitung vom Autorisierungsendpunkt eines Anbieters erfolgt.

Gegenmaßnahmen:

- Clients DÜRFEN Access Token NICHT in einem URI-Abfrageparameter übergeben, wie in Kapitel 2.3 von [RFC6750] beschrieben. Zu diesem Zweck kann der Authorization Code Grant oder alternative OAuth-

Antwortmodi wie der Form-POST-Antwortmodus [OAuth.Post] verwendet werden.

4.4. Mix-Up-Angriffe

Mix-Up-Angriffe können in Szenarien auftreten, in denen ein OAuth-Client mit zwei oder mehr Authorization Servern interagiert und mindestens ein Authorization Server unter der Kontrolle des Angreifers steht. Dies kann beispielsweise der Fall sein, wenn der Angreifer eine dynamische Registrierung verwendet, um den Client bei seinem eigenen Authorization Server zu registrieren, oder wenn ein Authorization Server kompromittiert wird.

Das Ziel des Angriffs ist es, einen Autorisierungscode oder ein Access Token für einen nicht kompromittierten Authorization Server zu erhalten . Dies wird erreicht, indem der Client dazu gebracht wird, diese Anmeldedaten an den kompromittierten Authorization Server (den Angreifer) zu senden, anstatt sie am jeweiligen Endpunkt des nicht kompromittierten Authorization-/Resource Servers zu verwenden.

4.4.1. Beschreibung des Angriffs

Die Beschreibung hier folgt [arXiv.1601.01229], mit Varianten des Angriffs, die unten beschrieben werden.

Voraussetzungen: Damit diese Variante des Angriffs funktioniert, wird davon ausgegangen, dass

- der Implicit Grant oder der Authorization Code Grant mit mehreren Authorization Servern verwendet wird, von denen einer als „ehrlich" (honest) (H-AS) und einer vom Angreifer betrieben wird (A-AS), und dass
- der Client den vom Benutzer gewählten Authorization Server in einer an den Browser des Benutzers gebundenen Sitzung speichert und für jeden Authorization Server dieselbe Redirection URI verwendet.

> (Hinweis: Der Implicit Grant ist gemäß Kapitel 5.1 des Begleitdokuments „Föderale Sicherheitsvorgaben für den Schutzbedarf Normal" verboten; die Voraussetzung ist daher nur für den Authorization Code Grant relevant.)

Im Folgenden wird ferner angenommen, dass der Client bei H-AS (URI: `https://honest.as.example`, Client-ID: `7ZGZldHQ`) und bei A-AS (URI: `https://attacker.example`, Client-ID: `666RVZJTA`) registriert ist. Die im folgenden Beispiel gezeigten URLs sind zur Darstellung verkürzt und enthalten nur die für den Angriff relevanten Parameter.

Angriff auf den Authorization Code Grant:

1. Der Benutzer wählt den Grant über A-AS aus (z. B. durch Klicken auf eine Schaltfläche auf der Website des Clients).
2. Der Client speichert in der Sitzung des Benutzers, dass der Benutzer „A-AS" ausgewählt hat, und leitet den Benutzer zum Authorization Code Grant-Endpunkt von A-AS mit einem Location-Header, der die URL `https://attacker.example/authorize?response_type=code&client_id=666RVZJTA` enthält.
3. Wenn der Browser des Benutzers zum Autorisierungsendpunkt des Angreifers navigiert, leitet der Angreifer den Browser sofort zum Autorisierungsendpunkt von H-AS weiter. In der Autorisierungsanfrage ersetzt der Angreifer die Client-ID des Clients bei A-AS durch die Client-ID bei H-AS. Daher erhält der Browser eine Weiterleitung (303 See Other) mit einem Location-Header, der auf `https://honest.as.example/authorize?response_type=code&client_id=7ZGZldHQ` verweist.
4. Der Benutzer autorisiert den Client für den Zugriff auf seine Ressourcen bei H-AS. (Beachten Sie, dass ein aufmerksamer Benutzer an dieser Stelle möglicherweise feststellen könnte, dass er eigentlich A-AS

statt H-AS verwenden wollte. Die erste aufgeführte Angriffsvariante hat diese Einschränkung nicht.) H-AS gibt einen Code aus und sendet ihn (über den Browser) zurück an den Client.
5. Da der Client weiterhin davon ausgeht, dass der Code von A-AS ausgegeben wurde, versucht er, den Code am Token-Endpunkt von A-AS einzulösen.
6. Der Angreifer erhält somit den Code und kann ihn entweder gegen ein Access Token (für Public Clients) einlösen oder einen Autorisierungscode-Injection-Angriff durchführen, wie in Kapitel 4.5 beschrieben.

Varianten:

- Verwechslung mit Abfangen: Diese Variante funktioniert nur, wenn der Angreifer die erste Anfrage/Antwortpaar vom Browser eines Benutzers an den Client abfangen und manipulieren kann (bei dem der Benutzer einen bestimmten Authorization Server auswählt und dann vom Client zu diesem Authorization Server umgeleitet wird), wie in Angreifer (A2) (siehe Kapitel 3). Diese Fähigkeit kann beispielsweise das Ergebnis eines Man-in-the-Middle-Angriffs auf die Verbindung des Benutzers zum Client sein. Bei dem Angriff beginnt der Benutzer den Ablauf mit H-AS. Der Angreifer fängt diese Anfrage ab und ändert die Auswahl des Benutzers zu A-AS. Der Rest des Angriffs verläuft wie in Schritt 2 (siehe oben) und den folgenden Schritten oben.
- Implicit Grant: Bei dem Implicit Grant erhält der Angreifer anstelle des Codes in Schritt 4 (siehe oben) ein Access Token. Der Authorization Server des Angreifers erhält das Access Token, wenn der Client entweder eine Anfrage an den A-AS-`userinfo`-Endpunkt (definiert in [Open ID.Core]) oder eine Anfrage an den Resource Server des Angreifers sendet (da der Client glaubt, den Ablauf mit A-AS abgeschlossen zu haben).
- Per-AS-Redirection URIs: Wenn Clients unterschiedliche Redirection URIs für verschiedene Authorization Server verwenden, speichern Clients den ausgewählten Authorization Server nicht in der Sitzung des Benutzers, und Authorization Server überprüfen die Redirection URIs nicht ordnungsgemäß, sodass Angreifer einen Angriff namens „Cross Social-Network Request Forgery" starten können. Diese Angriffe wurden in der Praxis beobachtet. Weitere Informationen finden Sie unter [research.jcs_14].
- OpenID Connect: Einige Varianten können für Angriffe auf OpenID Connect verwendet werden. Bei diesen Angriffen missbraucht der Angreifer Funktionen des OpenID Connect Discovery-Mechanismus [OpenID.Discovery] oder wiederholt Access Token oder ID-Token, um einen Mix-up-Angriff durchzuführen. Die Angriffe werden ausführlich in Anhang A von [arXiv.1704.08539] und Kapitel 6 von [arXiv.1508.04324v2] („Malicious Endpoints Attacks") beschrieben.

> Da der Implicit Grant gemäß Kapitel 5.1 des Begleitdokuments „Föderale Sicherheitsvorgaben für den Schutzbedarf Normal" verboten ist, ist diese Angriffsvariante bei konformer Umsetzung nicht anwendbar.

4.4.2. Gegenmaßnahmen

Wenn ein OAuth-Client nur mit einem Authorization Server interagieren kann, ist eine Mix-up-Abwehr nicht erforderlich. In Szenarien, in denen ein OAuth-Client mit zwei oder mehr Authorization Servern interagiert, MÜSSEN Clients jedoch Mix-up-Angriffe verhindern. Im Folgenden werden zwei verschiedene Methoden erläutert.

Bei beiden Abwehrmaßnahmen MÜSSEN Clients für jede Autorisierungsanfrage den Issuer speichern, an den sie die Autorisierungsanfrage gesendet haben, und diese Informationen an den Benutzeragenten binden. Der Issuer dient über die zugehörigen Metadaten als abstrakte Kennung für die Kombination aus Autorisierungsendpunkt und Token-Endpunkt, die im Ablauf verwendet werden sollen. Wenn keine

Issuerkennung verfügbar ist (z. B. wenn weder OAuth Authorization Server Metadaten [RFC8414] noch OpenID Connect Discovery [OpenID.Discovery] verwendet werden), kann stattdessen eine andere eindeutige Kennung für dieses Tupel oder das Tupel selbst verwendet werden. Der Kürze halber wird eine solche einsatzspezifische Kennung im Folgenden unter dem Issuer (oder der Issuerkennung) subsumiert.

Es ist wichtig zu beachten, dass die bloße Speicherung der URL des Authorization Server nicht ausreicht, um Verwechslungsangriffe zu identifizieren. Ein Angreifer könnte die URL des Authorization Server eines nicht kompromittierten Authorization Server Endpunkt-URL eines nicht kompromittierten Authorization Server als „seine" Authorization Server-URL deklarieren, aber einen Token-Endpunkt unter seiner eigenen Kontrolle deklarieren.

4.4.2.1. Verteidigung gegen Verwechslungen durch Identifizierung des Issuers

Diese Verteidigung erfordert, dass der Authorization Server seine Issuer-Kennung in der Autorisierungsantwort an den Client sendet. Beim Empfang der Autorisierungsantwort MUSS der Client die empfangene Issuer-Kennung mit der gespeicherten Issuer-Kennung vergleichen. Bei einer Nichtübereinstimmung MUSS der Client die Interaktion abbrechen.

Es gibt verschiedene Möglichkeiten, wie diese Issuer-Kennung an den Client übertragen werden kann:

- Die Informationen des Issuer können beispielsweise über einen separaten Antwortparameter `iss` übertragen werden, der in [RFC9207] definiert ist.
- Wenn OpenID Connect verwendet wird und ein ID-Token in der Autorisierungsantwort zurückgegeben wird, kann der Client das `iss`-Claim im ID-Token auswerten.

In beiden Fällen MUSS der `iss`-Wert gemäß [RFC9207] ausgewertet werden.

Diese Abwehrmaßnahme erfordert zwar möglicherweise die Bereitstellung neuer OAuth-Funktionen für die Übertragung der Issuer-Informationen, ist jedoch eine robuste und relativ einfache Abwehrmaßnahme gegen Verwechslungen.

4.4.2.2. Mix-Up-Abwehr über eindeutige Redirection URIs

Für diese Abwehrmaßnahme MÜSSEN Clients für jeden Issuer, mit dem sie interagieren, eine eindeutige Redirection URI verwenden.

Clients MÜSSEN überprüfen, ob die Autorisierungsantwort vom richtigen Issuer empfangen wurde, indem sie die eindeutige Redirection URI für den Issuer mit der URI vergleichen, unter der die Autorisierungsantwort empfangen wurde. Bei einer Nichtübereinstimmung MUSS der Client den Ablauf abbrechen.

Diese Abwehrmaßnahme baut zwar auf der bestehenden OAuth-Funktionalität auf, kann jedoch nicht in Szenarien verwendet werden, in denen sich Clients nur einmal für die Nutzung vieler verschiedener Issuer registrieren (wie in einigen Open-Banking-Systemen), und aufgrund der engen Integration mit der Client-Registrierung ist sie schwieriger automatisch zu implementieren.

Darüber hinaus könnte ein Angreifer den durch diese Abwehrmaßnahme gebotenen Schutz umgehen, indem er einen neuen Client beim „ehrlichen" Authorization Server registriert, wobei er die Redirection URI verwendet, die der Client dem Authorization Server des Angreifers zugewiesen hat. Der Angreifer könnte dann den oben beschriebenen Angriff ausführen und dabei die Client-ID durch die Client-ID seines neu erstellten Clients ersetzen.

Diese Abwehrmaßnahme SOLLTE daher nur verwendet werden, wenn keine anderen Optionen verfügbar sind.

4.5. Autorisierungscode-Injection

Ein Angreifer, der Zugriff auf einen in einer Autorisierungsantwort enthaltenen Autorisierungscode erhalten hat (siehe Angreifer (A3) in Kapitel 3), kann versuchen, den Autorisierungscode gegen ein Access Token einzulösen oder den Autorisierungscode anderweitig zu nutzen.

Falls der Autorisierungscode für einen Public Client erstellt wurde, kann der Angreifer den Autorisierungscode an den Token-Endpunkt des Authorization Servers senden und so ein Access Token erhalten. Dieser Angriff wurde in Kapitel 4.4.1.1 von [RFC6819] beschrieben.

Bei Confidential Clients oder in bestimmten Situationen kann der Angreifer einen Angriff mit Injection von Autorisierungscodes ausführen, wie im Folgenden beschrieben.

Bei einem Angriff auf die Autorisierung versucht der Angreifer, einen gestohlenen Autorisierungscode in seine eigene Sitzung mit dem Client zu injizieren. Das Ziel besteht darin, die Sitzung des Angreifers beim Client mit den Ressourcen oder der Identität des Opfers zu verknüpfen, um so dem Angreifer zumindest eingeschränkten Zugriff auf die Ressourcen des Opfers zu verschaffen.

Neben der Umgehung der Client-Authentifizierung von Confidential Clients gibt es weitere Anwendungsfälle für diesen Angriff:

- Der Angreifer möchte auf bestimmte Funktionen in diesem bestimmten Client zugreifen. Beispielsweise möchte der Angreifer sich in einer bestimmten App oder auf einer bestimmten Website als sein Opfer ausgeben.
- Die Authorization Server oder Resource Server sind auf bestimmte Netzwerke beschränkt, auf die der Angreifer keinen direkten Zugriff hat.

Außer in diesen Sonderfällen ist die Autorisierungscode-Injection in der Regel nicht interessant, wenn der Code für einen Public Client erstellt wurde, da das Senden des Codes an den Token-Endpunkt eine einfachere und leistungsfähigere Angriffsmethode ist, wie oben beschrieben.

4.5.1. Angriffsbeschreibung

Der Autorisierungscode-Injection-Angriff funktioniert wie folgt:

1. Der Angreifer erhält einen Autorisierungscode (siehe Angreifer (A3) in Kapitel 3). Für den Rest des Angriffs sind nur die Fähigkeiten eines Web-Angreifers (A1) erforderlich.
2. Vom Gerät des Angreifers aus startet der Angreifer einen regulären OAuth-Autorisierungsprozess mit dem legitimen Client.
3. In der Antwort des Authorization Servers an den legitimen Client ersetzt der Angreifer den neu erstellten Autorisierungscode durch den gestohlenen Autorisierungscode. Da diese Antwort über das Gerät des Angreifers läuft, kann der Angreifer jedes Tool verwenden, das die Autorisierungsantwort zu diesem Zweck abfangen und manipulieren kann. Der Angreifer muss das Netzwerk nicht kontrollieren.
4. Der legitime Client sendet den `code` zusammen mit der `redirect_uri` und der Client-ID und dem Client-Secret des Clients (oder anderen Mitteln zur Client-Authentifizierung) an den Token-Endpunkt des Authorization Servers.
5. Der Authorization Server überprüft das Client-Secret, ob der `code` für den bestimmten Client ausgestellt wurde und ob die tatsächliche Redirection URI mit dem Parameter `redirect_uri` übereinstimmt (siehe

[RFC6749]).

6. Alle Überprüfungen sind erfolgreich und der Authorization Server stellt dem Client Zugriffs- und andere Token aus. Der Angreifer hat nun seine Sitzung mit dem legitimen Client mit den Ressourcen und/oder der Identität des Opfers verknüpft.

4.5.2. Diskussion

Offensichtlich schlägt der Check-in-Schritt (Schritt 5 aus Kapitel 4.5.1) fehl, wenn der `code` an eine andere Client-ID ausgegeben wurde, z.B. einem vom Angreifer eingerichteten Client. Die Überprüfung schlägt auch fehl, wenn der Autorisierungscode bereits vom legitimen Benutzer eingelöst wurde und nur einmalig verwendet werden konnte.

Ein Versuch, einen über eine manipulierte Redirection URI erhaltenen Code einzuschleusen, sollte ebenfalls erkannt werden, wenn der Authorization Server die vollständige Redirection URI gespeichert hat, die in der Autorisierungsanfrage verwendet wurde, und diese mit dem Parameter `redirect_uri` vergleicht.

Kapitel 4.1.3 von [RFC6749] verlangt vom Authorization Server, sicherzustellen, dass der Parameter `redirect_uri` vorhanden ist, wenn der Parameter `redirect_uri` in der ursprünglichen Autorisierungsanfrage enthalten war, wie in Kapitel 4.1.1 von [RFC6749], und wenn ja, sicherzustellen, dass ihre Werte identisch sind.

In dem in Kapitel 4.5.1 beschriebenen Angriffsszenario würde der legitime Client die korrekte Redirection URI verwenden, die er immer für Autorisierungsanfragen verwendet. Diese Redirection URI würde jedoch nicht mit der vom Angreifer verwendeten manipulierten Redirection URI übereinstimmen (ansonsten würde die Umleitung nicht auf der Seite des Angreifers landen). Der Authorization Server würde also den Angriff erkennen und den Austausch des Codes verweigern.

Diese Überprüfung könnte auch Versuche erkennen, einen Autorisierungscode einzuschleusen, der von einer anderen Instanz desselben Clients auf einem anderen Gerät erhalten wurde, wenn bestimmte Bedingungen erfüllt sind:

- Die Redirection URI selbst enthält eine `nonce` oder eine andere Art von einmalig verwendbaren geheimen Daten, und
- der Client hat diese Daten an diese bestimmte Instanz des Clients gebunden.

Dieser Ansatz steht jedoch im Widerspruch zu der Idee, eine exakte Übereinstimmung der Redirection URI am Endpunkt der Autorisierung zu erzwingen. Darüber hinaus wurde beobachtet, dass Anbieter die Anforderung der Überprüfung der `redirect_uri` in dieser Phase sehr oft ignorieren, möglicherweise weil es beim Lesen der Spezifikation nicht sicherheitskritisch zu sein scheint.

Andere Anbieter gleichen den Parameter `redirect_uri` lediglich mit dem registrierten Redirection URI-Muster ab. Dadurch muss der Authorization Server nicht für jede Transaktion die Verknüpfung zwischen der tatsächlichen Redirection URI und dem entsprechenden Autorisierungscode speichern. Diese Art der Überprüfung entspricht jedoch offensichtlich nicht der Absicht der Spezifikation, da die manipulierte Redirection URI nicht berücksichtigt wird. Daher wird jeder Versuch, einen Autorisierungscode einzuschleusen, der mit der `client_id` eines legitimen Clients oder unter Verwendung des legitimen Clients auf einem anderen Gerät erhalten wurde, wird in den jeweiligen Deployments nicht erkannt.

Es wird auch davon ausgegangen, dass die in Kapitel 4.1.3 von [RFC6749] definierten Anforderungen die Komplexität der Client-Implementierung erhöhen, da Clients die korrekte Redirection URI für den Aufruf des

Token-Endpunkts speichern oder rekonstruieren müssen.

Asymmetrische Methoden zur Client-Authentifizierung verhindern diesen Angriff nicht, da sich der legitime Client am Token-Endpunkt authentifiziert.

Dieses Dokument empfiehlt daher stattdessen, jeden Autorisierungscode im Kontext einer bestimmten Transaktion unter Verwendung eines der nachfolgend beschriebenen Mechanismen an eine bestimmte Client-Instanz auf einem bestimmten Gerät (oder in einem bestimmten Benutzeragenten) zu binden.

4.5.3. Gegenmaßnahmen

Es gibt zwei gute technische Lösungen, um Autorisierungscodes an Client-Instanzen zu binden, wie folgt.

4.5.3.1. PKCE

Der in [RFC7636] spezifizierte PKCE-Mechanismus kann als Gegenmaßnahme verwendet werden (auch wenn er ursprünglich zur Sicherung nativer Apps entwickelt wurde). Wenn der Angreifer versucht, einen Code für die Autorisierung einzuschleusen, schlägt die Überprüfung des `code_verifier` fehl: Der Client verwendet seinen korrekten Verifizierer, aber der Code ist mit einem `code_challenge` verknüpft, der nicht mit diesem Verifizierer übereinstimmt.

PKCE schützt nicht nur vor Angriffen durch das Einfügen von Autorisierungscodes, sondern auch vor Autorisierungscodes, die für Public Clients erstellt wurden: PKCE stellt sicher, dass ein Angreifer einen gestohlenen Autorisierungscode nicht am Token-Endpunkt des Authorization Servers einlösen kann, ohne den `code_verifier` zu kennen.

4.5.3.2. `nonce`

Der in OpenID Connect vorhandene `nonce`-Parameter kann vor Autorisierungscode-Injection-Angriffen schützen. Der `nonce`-Wert ist einmalig verwendbar und wird vom Client erstellt. Der Client soll ihn an die User-Agent-Sitzung binden und mit der ersten Anfrage an den OpenID-Anbieter (OP) senden. Der OP fügt den empfangenen `nonce`-Wert in den ID-Token ein, der im Rahmen des Codeaustauschs am Token-Endpunkt ausgestellt wird. Wenn ein Angreifer einen Autorisierungscode in die Autorisierungsantwort injiziert, stimmen der `nonce`-Wert in der Client-Sitzung und der `nonce`-Wert im ID-Token, das vom Token-Endpunkt empfangen wurde, nicht überein und der Angriff wird erkannt. Dabei wird davon ausgegangen, dass ein Angreifer keinen Zugriff auf den User-Agent-Status auf dem Gerät des Opfers hat (von dem der Angreifer den entsprechenden Code für die Autorisierung gestohlen hat).

Es ist wichtig zu beachten, dass diese Gegenmaßnahme nur funktioniert, wenn der Client den `nonce`-Parameter im ID-Token, das vom Token-Endpunkt erhalten wurde, ordnungsgemäß überprüft und kein ausgegebenes Token verwendet, bis diese Überprüfung erfolgreich war. Genauer gesagt, ein Client, der sich mit dem `nonce`-Parameter vor Code-Injections schützt

1. MUSS den `nonce` im ID-Token, das vom Token-Endpunkt erhalten wurde, validieren, auch wenn ein anderes ID-Token aus der Autorisierungsantwort erhalten wurde (z. B. `response_type=code+id_token`) erhalten wurde, und
2. MUSS sicherstellen, dass alle Token (ID-Token und das Access Token) ignoriert und nicht für andere Zwecke verwendet werden, bis diese Überprüfung erfolgreich war.

Es ist wichtig zu beachten, dass `nonce` die Autorisierungscodes von Public Clients nicht schützt, da ein Angreifer keinen Autorisierungscode-Injection-Angriff ausführen muss. Stattdessen kann ein Angreifer den Token-Endpunkt direkt mit dem gestohlenen Autorisierungscode aufrufen.

4.5.3.3. Andere Lösungen

Andere Lösungen wie die Bindung des `state` an den `code`, das Sender-Constraining des `code` mithilfe kryptografischer Mittel oder instanzbezogene Client-Anmeldedaten sind denkbar, werden jedoch aktuell nicht unterstützt und bringen neue Sicherheitsanforderungen mit sich.

PKCE ist die naheliegendste Lösung für OAuth-Clients, da sie zum Zeitpunkt der Erstellung dieses Dokuments verfügbar ist, während `nonce` für OpenID Connect-Clients geeignet ist.

4.5.4. Einschränkungen

Ein Angreifer kann die oben beschriebenen Gegenmaßnahmen umgehen, wenn er die `nonce`- oder `code_challenge`-Werte ändern kann, die in der Autorisierungsanfrage des Opfers verwendet werden. Der Angreifer kann diese Werte so ändern, dass sie mit denen übereinstimmen, die der Client in seiner eigenen Sitzung in Schritt 2 aus Kapitel 4.5.1 des oben beschriebenen Angriffs ausgewählt hat. (Dazu muss die Sitzung des Opfers mit dem Client nach dem Start der Sitzung des Angreifers mit dem Client beginnen.) Wenn der Angreifer dann in der Lage ist, den Autorisierungscode vom Opfer zu erfassen, kann er den gestohlenen Code in Schritt 3 aus Kapitel 4.5.1 einfügen, selbst wenn PKCE oder `nonce` verwendet werden.

Dieser Angriff ist komplex und erfordert eine enge Interaktion zwischen dem Angreifer und der Sitzung des Opfers. Dennoch müssen Maßnahmen ergriffen werden, um Angreifer daran zu hindern, den Inhalt der Autorisierungsantwort zu lesen, wie in den Kapiteln 4.1, 4.2, 4.3 , 4.4 und 4.11 beschrieben.

4.6. Access Token Injection

Bei einem Access Token Injection-Angriff versucht der Angreifer, ein gestohlenes Access Token in einen legitimen Client (der nicht unter der Kontrolle des Angreifers steht) einzuschleusen. Dies geschieht in der Regel, wenn der Angreifer ein geleaktes Access Token nutzen möchte, um sich in einem bestimmten Client als Benutzer auszugeben.

Um den Angriff durchzuführen, startet der Angreifer einen OAuth-Flow mit dem Client unter Verwendung des Implicit Grant und modifiziert die Autorisierungsantwort, indem er das vom Authorization Server ausgestellte Access Token ersetzt oder direkt eine Autorisierungsserverantwort einschließlich des geleakten Access Tokens erstellt. Da die Antwort den vom Client für diese bestimmte Transaktion generierten Statuswert enthält, behandelt der Client die Antwort nicht als CSRF-Angriff und verwendet das vom Angreifer injizierte Access Token.

4.6.1. Gegenmaßnahmen

Es gibt keine Möglichkeit, einen solchen Injection-Angriff in reinen OAuth-Flows zu erkennen, da das Token ohne Bindung an die Transaktion oder den bestimmten Benutzeragenten ausgestellt wird.

In OpenID Connect kann der Angriff abgeschwächt werden, da die Autorisierungsantwort zusätzlich einen ID-Token enthält, der den Claim `at_hash` enthält. Der Angreifer muss daher sowohl das Access Token als auch den ID-Token in der Antwort ersetzen. Der Angreifer kann den ID-Token nicht fälschen, da er mit einer

Authentifizierung signiert oder verschlüsselt ist. Der Angreifer kann auch kein geleaktes ID-Token injizieren, das mit dem gestohlenen Access Token übereinstimmt, da der `nonce`-Claim im geleakten ID-Token (mit sehr hoher Wahrscheinlichkeit) einen anderen Wert als den in der Autorisierungsantwort erwarteten enthalten wird.

Beachten Sie, dass weiterhin ein zusätzlicher Schutz, wie z. B. Sender-Constrained Access Token, erforderlich ist, um zu verhindern, dass Angreifer das Access Token direkt am Ressourcenendpunkt verwenden können.

Die Empfehlungen in Sicherheitsvorgaben für die Absicherung von APIs für den Schutzbedarf Normal leiten sich daraus ab.

4.7. Cross-Site Request Forgery

Ein Angreifer könnte versuchen, eine Anfrage an die Redirection URI des legitimen Clients auf dem Gerät des Opfers einzuschleusen, um beispielsweise den Client dazu zu bringen, auf Ressourcen unter der Kontrolle des Angreifers zuzugreifen. Dies ist eine Variante eines Angriffs, der als Cross-Site Request Forgery (CSRF) bekannt ist.

4.7.1. Gegenmaßnahmen

Die seit langem etablierte Gegenmaßnahme besteht darin, dass Clients einen Zufallswert, auch bekannt als CSRF-Token, im `state`-Parameter übergeben, das die Anfrage mit der Redirection URI mit der User-Agent-Sitzung verknüpft, wie beschrieben. Diese Gegenmaßnahme wird in Kapitel 5.3.5 von [RFC6819] ausführlich beschrieben. Der gleiche Schutz wird durch PKCE oder den OpenID Connect-`nonce`-Wert geboten.

Bei der Verwendung von PKCE anstelle von `state` oder `nonce` für den CSRF-Schutz ist Folgendes zu beachten:

- Clients MÜSSEN sicherstellen, dass der Authorization Server PKCE unterstützt, bevor sie PKCE für den CSRF-Schutz verwenden. Wenn ein Authorization Server PKCE nicht unterstützt, MÜSSEN state oder `nonce` für den CSRF-Schutz verwendet werden.
- Wenn state für die Übertragung des Anwendungsstatus verwendet wird und die Integrität seines Inhalts ein Problem darstellt, MÜSSEN Clients den Status vor Manipulation und Austausch schützen. Dies kann erreicht werden, indem der Inhalt des Status an die Browsersitzung gebunden und/oder die Statuswerte signiert/verschlüsselt werden. Ein Beispiel hierfür wird im abgelaufenen Internet-Entwurf [JWT-ENCODED-STATE] diskutiert.

Der Authorization Server MUSS daher eine Möglichkeit bieten, seine Unterstützung für PKCE zu erkennen. Die Verwendung von Authorization Server Metadaten gemäß [RFC8414] wird EMPFOHLEN, aber Authorization Server KÖNNEN stattdessen auch eine einsatzspezifische Möglichkeit bereitstellen, um die PKCE-Unterstützung sicherzustellen oder festzustellen.

PKCE bietet einen robusten Schutz vor CSRF-Angriffen, selbst wenn ein Angreifer die Autorisierungsantwort lesen kann (siehe Angreifer (A3) in Kapitel 3). Wenn der Status verwendet wird oder ein ID-Token in der Autorisierungsantwort zurückgegeben wird (z. B. `response_type=code+id_token`) zurückgegeben wird, erfährt der Angreifer entweder den Statuswert und kann ihn in die gefälschte Autorisierungsantwort einfügen, oder er kann die `nonce` aus dem ID-Token extrahieren und sie in einer neuen Anfrage an den Authorization Server verwenden, um ein ID-Token mit derselben `nonce` zu erstellen. Das neue ID-Token kann dann für den CSRF-Angriff verwendet werden.

4.8. PKCE-Downgrade-Angriff

Ein Authorization Server, der PKCE unterstützt, dessen Verwendung jedoch nicht für alle Flows zwingend vorgeschrieben ist, kann anfällig für einen PKCE-Downgrade-Angriff sein.

Die erste Voraussetzung für diesen Angriff ist, dass in der Autorisierungsanfrage ein vom Angreifer kontrollierbares Flag vorhanden ist, das PKCE für den jeweiligen Ablauf aktiviert oder deaktiviert. Das Vorhandensein oder Fehlen des Parameters `code_challenge` eignet sich für diesen Zweck, d. h., der Authorization Server aktiviert und erzwingt PKCE, wenn dieser Parameter in der Autorisierungsanfrage vorhanden ist, erzwingt PKCE jedoch nicht, wenn der Parameter fehlt.

Die zweite Voraussetzung für diesen Angriff ist, dass der Client überhaupt keinen Status verwendet (z. B. weil der Client zur CSRF-Verhinderung auf PKCE setzt) oder dass der Client den Status nicht korrekt überprüft.

Grob gesagt handelt es sich bei diesem Angriff um eine Variante eines CSRF-Angriffs. Der Angreifer erreicht dasselbe Ziel wie bei dem in Kapitel 4.7 beschriebenen Angriff: Der Angreifer injiziert einen Code für die Autorisierung (und damit ein Access Token), der an die Ressourcen des Angreifers gebunden ist, in eine Sitzung zwischen seinem Opfer und dem Client.

4.8. 1. Angriffsbeschreibung

1. Der Benutzer hat eine OAuth-Sitzung mit einem Client auf einem Authorization Server gestartet. In der Autorisierungsanfrage hat der Client den Parameter `code_challenge=hash(abc)` als PKCE-Code-Challenge festgelegt (mit der Hash-Funktion und der Parameterkodierung, wie in [RFC7636] definiert). Der Client wartet nun auf die Autorisierungsantwort vom Browser des Benutzers.
2. Um den Angriff durchzuführen, startet der Angreifer mit seinem eigenen Gerät einen Prozess der Autorisierung mit dem Ziel-Client. Der Client verwendet nun eine andere PKCE-Code-Challenge, beispielsweise `code_challenge=hash(xyz)`, in der Autorisierungsanfrage. Der Angreifer fängt die Anfrage ab und entfernt den gesamten Parameter `code_challenge` aus der Anfrage. Da dieser Schritt auf dem Gerät des Angreifers durchgeführt wird, hat der Angreifer vollständigen Zugriff auf den Inhalt der Anfrage, beispielsweise mithilfe von Browser-Debugging-Tools.
3. Wenn der Authorization Server Flows ohne PKCE zulässt, erstellt er einen Code, der nicht an eine PKCE-Code-Challenge gebunden ist.
4. Der Angreifer leitet nun den Browser des Benutzers zu einer Autorisierungsantwort-URL um, die den Code für die Sitzung des Angreifers mit dem Authorization Server enthält.
5. Der Browser des Benutzers sendet den Autorisierungscode an den Client, der nun versucht, den Code beim Authorization Server gegen ein Access Token einzulösen. Der Client sendet `code_verifier=abc` als PKCE-Code-Verifizierer in der Token-Anfrage.
6. Da der Authorization Server sieht, dass dieser Code nicht an eine PKCE-Code-Challenge gebunden ist, überprüft er weder das Vorhandensein noch den Inhalt des Parameters `code_verifier`. Er stellt dem Client unter der Kontrolle des Benutzers ein Access Token (das zur Ressource des Angreifers gehört) aus.

4.8.2. Gegenmaßnahmen

Die ordnungsgemäße Verwendung des `state` würde diesen Angriff verhindern. Die Praxis hat jedoch gezeigt, dass viele OAuth-Clients den `state` nicht ordnungsgemäß verwenden oder überprüfen.

Daher MÜSSEN Authorization Server diesen Angriff abwehren.

Beachten Sie, dass aus Sicht des Authorization Servers bei dem oben beschriebenen Angriff ein `code_verifier`-Parameter am Token-Endpunkt empfangen wird, obwohl in der Autorisierungsanfrage für

den OAuth-Ablauf, in dem der Autorisierungscode abgeschaltet wurde, kein `code_challenge`-Parameter vorhanden war.

Diese Tatsache kann genutzt werden, um hier gegenzuwirken. [RFC7636] schreibt bereits vor, dass

- ein Authorization Server, der PKCE unterstützt, überprüfen MUSS, ob die Autorisierungsanfrage eine Code-Challenge enthält, und diese Information an den ausgegebenen Code binden MUSS; und
- wenn ein `code` am Token-Endpunkt ankommt und die Autorisierungsanfrage, für die dieser `code` ausgegeben wurde, einen `code_challenge` enthielt, muss in der Token-Anforderung ein gültiger `code_verifier` vorhanden sein.

Darüber hinaus MUSS der Authorization Server, um PKCE-Downgrade-Angriffe zu verhindern, sicherstellen, dass eine Anforderung an den Token-Endpunkt, die einen `code_verifier` enthält, abgelehnt wird, wenn in der Autorisierungsanfrage kein `code_challenge` vorhanden war.

Authorization Server, die die Verwendung von PKCE vorschreiben (im Allgemeinen oder für bestimmte Clients), implementieren diese Sicherheitsmaßnahme implizit.

4.9. Access Token-Leakage auf dem Resource Server

Access Token können unter bestimmten Umständen von einem Resource Server geleakt werden.

4.9.1. Phishing von Access Token durch gefälschten Resource Server

Ein Angreifer kann einen eigenen Resource Server einrichten und einen Client dazu verleiten, Access Token an diesen zu senden, die für andere Resource Server gültig sind (siehe Angreifer (A1) und (A5) in Kapitel 3). Wenn der Client ein gültiges Access Token an diesen gefälschten Resource Server sendet, kann der Angreifer dieses Token wiederum verwenden, um im Namen des Resource Owners auf andere Dienste zuzugreifen.

Dieser Angriff setzt voraus, dass der Client zum Zeitpunkt der Entwicklung nicht an einen bestimmten Resource Server (und dessen URL) gebunden ist, sondern dass Client-Instanzen zur Laufzeit mit der URL des Resource Servers versehen werden. Diese Art der späten Bindung ist typisch für Situationen, in denen der Client einen Dienst nutzt, der eine standardisierte API implementiert (z.B. für E-Mail, Kalender, E-Health oder Open Banking) und der Client von einem Benutzer oder Administrator konfiguriert wird.

4.9.2. Kompromittierter Resource Server

Ein Angreifer kann einen Resource Server kompromittieren, um Zugriff auf die Ressourcen der jeweiligen Bereitstellung zu erhalten. Eine solche Kompromittierung kann von einem teilweisen Zugriff auf das System, z. B. auf dessen Protokolldateien, bis hin zur vollständigen Kontrolle über den jeweiligen Server reichen, wobei in diesem Fall alle Kontrollen umgangen und auf alle Ressourcen zugegriffen werden kann. Der Angreifer wäre auch in der Lage, andere Access Token zu erhalten, die auf dem kompromittierten System gespeichert sind und möglicherweise für den Zugriff auf andere Resource Server gültig sind.

Die Verhinderung von Server-Breaches durch die Absicherung und Überwachung von Serversystemen gilt als Standardverfahren und fällt daher nicht in den Geltungsbereich dieses Dokuments. Kapitel 4.9 konzentriert sich auf die Auswirkungen von OAuth-bezogenen Verletzungen und die Wiederholung von abgefangenen Access Tokens.

4.9.3. Gegenmaßnahmen

Die folgenden Maßnahmen sollten von Entwicklern berücksichtigt werden, um mit der Wiederholung von Access Tokens durch böswillige Akteure korrekt umgehen zu können:

- Sender-Constrained Access Token, wie in Kapitel 4.10.1 beschrieben, SOLLTEN verwendet werden, um zu verhindern, dass der Angreifer die Access Tokens auf anderen Resource Servern wiederholt. Wenn ein Angreifer nur teilweisen Zugriff auf das kompromittierte System hat, z. B. Lesezugriff auf Webserver-Protokolle, können Sender-Constrained Access Tokens auch die Wiederholung auf dem kompromittierten System verhindern.
- Die in Kapitel 4.10.2 beschriebene `Audience`-Beschränkung SOLLTE verwendet werden, um die Wiederholung abgefangener Access Token auf anderen Resource Servern zu verhindern.
- Der Resource Server MUSS Access Token wie andere sensible Geheimnisse behandeln und darf sie nicht im Klartext speichern oder übertragen.

Die erste und zweite Empfehlung gelten auch für andere Szenarien, in denen Access Token verloren gehen (siehe Angreifer (A5) in Kapitel 3).

4.10. Missbrauch gestohlener Access Tokens

Access Tokens können von einem Angreifer auf verschiedene Weise gestohlen werden, beispielsweise durch die in den Kapiteln 4.1, 4.2, 4.3, 4.4 und 4.9 beschriebenen Angriffe. Einige dieser Angriffe können durch spezifische Sicherheitsmaßnahmen, wie in den jeweiligen Abschnitten beschrieben, abgeschwächt werden. In einigen Fällen sind diese Maßnahmen jedoch nicht ausreichend oder werden nicht korrekt umgesetzt. Authorization Server SOLLTEN daher sicherstellen, dass Access Token wie im Folgenden beschrieben absendergebunden und `audience`-beschränkt sind. Architektur- und Performancegründe können die Verwendung dieser Maßnahmen in einigen Bereitstellungen verhindern.

4.10.1. Sender-Constrained Access Token

Wie der Name schon sagt, beschränken Sender-Constrained Access Tokens die Anwendbarkeit eines Access Tokens auf einen bestimmten Absender. Dieser Absender muss als Voraussetzung für die Akzeptanz dieses Tokens auf einem Resource Server die Kenntnis eines bestimmten Secrets nachweisen.

Ein typischer Ablauf sieht wie folgt aus:

1. Der Authorization Server verknüpft Daten mit dem Access Token, wodurch dieses bestimmte Token an einen bestimmten Client gebunden wird. Die Bindung kann die Identität des Clients nutzen, aber in den meisten Fällen verwendet der Authorization Server Schlüsselmaterial (oder aus dem Schlüsselmaterial abgeleitete Daten), die dem Client bekannt sind.
2. Dieses Schlüsselmaterial muss auf irgendeine Weise verteilt werden. Entweder existiert das Schlüsselmaterial bereits, bevor der Authorization Server die Bindung erstellt, oder der Authorization Server erstellt kurzlebige Schlüssel. Die Art und Weise, wie bereits vorhandenes Schlüsselmaterial verteilt wird, variiert je nach Ansatz. Beispielsweise können X.509-Zertifikate verwendet werden, wobei die Verteilung explizit während des Registrierungsprozesses erfolgt. Oder das Schlüsselmaterial wird auf der TLS-Ebene erstellt und verteilt, in diesem Fall kann dies automatisch während des Aufbaus einer TLS-Verbindung geschehen.
3. Der Resource Server muss die eigentliche Überprüfung des Besitznachweises durchführen. Dies geschieht in der Regel auf Anwendungsebene und ist oft an bestimmtes Material gebunden, das von der Transportschicht (z.B. TLS) bereitgestellt wird. Der Resource Server muss auch sicherstellen, dass eine Wiederholung des Besitznachweises nicht möglich ist.

Zwei Methoden für Sender-Constrained Access Token unter Verwendung eines Besitznachweises wurden von der OAuth-Arbeitsgruppe definiert und werden in der Praxis verwendet:

- „OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens" [RFC8705]: Der in diesem Dokument beschriebene Ansatz ermöglicht die Verwendung von Mutual TLS sowohl für die Client-Authentifizierung als auch für Sender-Constrained Access Token. Für die Zwecke von Sender-Constrained Access Tokens wird der Client gegenüber dem Resource Server durch den Fingerabdruck seines öffentlichen Schlüssels identifiziert. Während der Verarbeitung einer Zugriffstokenanforderung ruft der Authorization Server den öffentlichen Schlüssel des Clients aus dem TLS-Stack ab und ordnet dessen Fingerabdruck den jeweiligen Access Token zu. Der Resource Server ruft auf die gleiche Weise den öffentlichen Schlüssel aus dem TLS-Stack ab und vergleicht dessen Fingerabdruck mit dem Fingerabdruck, der dem Access Token zugeordnet ist.
- „OAuth 2.0 Demonstrating Proof of Possession (DPoP)" [RFC9449]: DPoP beschreibt einen Mechanismus auf Anwendungsebene für den Zugriff auf Sender-Constrained Access Token und Refresh Token. Es verwendet einen Besitznachweis auf der Grundlage eines öffentlichen/privaten Schlüsselpaars und einer Signatur auf Anwendungsebene. DPoP kann mit Public Clients verwendet werden und im Falle von Confidential Clients mit jeder Client-Authentifizierungsmethode kombiniert werden.

Beachten Sie, dass die Sicherheit von Sender-Constrained Token untergraben wird, wenn ein Angreifer Zugriff auf das Token und das Schlüsselmaterial erhält. Dies ist insbesondere bei beschädigter Client-Software und Cross-Site-Scripting-Angriffen der Fall (wenn der Client im Browser ausgeführt wird). Wenn das Schlüsselmaterial in einem Hardware- oder Software-Sicherheitsmodul geschützt oder nur indirekt zugänglich ist (wie in einem TLS-Stack), schützen sender-contrained Token zumindest vor der Verwendung des Tokens, wenn der Client offline ist, d. h. wenn das Sicherheitsmodul oder die Schnittstelle für den Angreifer nicht verfügbar ist. Dies gilt sowohl für Access Token als auch für Refresh Token (siehe Kapitel 4.14).

4.10.2. Access Token mit eingeschränktem Publikum

Die `audience`-beschränkung beschränkt Access Token im Wesentlichen auf einen bestimmten Resource Server. Der Authorization Server ordnet den Access Token dem jeweiligen Resource Server zu, und der Resource Server muss dann die beabsichtigte `audience` überprüfen. Wenn der Access Token die Validierung der beabsichtigten `audience` nicht besteht, lehnt der Resource Server die Bearbeitung der entsprechenden Anfrage ab.

Im Allgemeinen begrenzt die `audience`-beschränkung die Auswirkungen einer Token-Leakage. Im Falle eines gefälschten Resource Servers kann sie (wie unten beschrieben) auch den Missbrauch eines durch Phishing erlangten Access Tokens auf dem legitimen Resource Server verhindern.

Die `audience` kann durch logische Namen oder physische Adressen (wie URLs) ausgedrückt werden. Um Phishing zu verhindern, muss die tatsächliche URL verwendet werden, an die der Client Anfragen sendet. Im Falle von Phishing verweist diese URL auf den gefälschten Resource Server. Wenn der Angreifer versucht, den Access Token auf dem legitimen Resource Server zu verwenden (der eine andere URL hat) zu verwenden, erkennt der Resource Server die Nichtübereinstimmung (falsche `audience`) und lehnt die Bearbeitung der Anfrage ab.

In Bereitstellungen, in denen der Authorization Server die URLs aller Resource Server kennt, kann der Authorization Server einfach die Ausgabe von Access Token für unbekannte Resource Server-URLs verweigern.

Damit dies funktioniert, muss der Client dem Authorization Server den vorgesehenen Resource Server mitteilen. Hierfür kann der Mechanismus in [RFC8707] verwendet werden, oder die Informationen können im Scope-Wert Kapitel 3.3 von [RFC6749]) kodiert werden.

Anstelle der URL kann auch der Fingerabdruck des X.509-Zertifikats des Resource Servers als Zielwert verwendet werden. Diese Variante würde auch die Erkennung von Versuchen ermöglichen, die URL des legitimen Resource Servers durch Verwendung eines gültigen TLS-Zertifikats zu fälschen, das von einer anderen Zertifizierungsstelle bezogen wurde. Es könnte auch als Vorteil für den Datenschutz angesehen werden, die URL des Resource Servers vor dem Authorization Server zu verbergen.

Die `audience`-beschränkung scheint einfacher zu verwenden zu sein, da sie keine Kryptografie auf der Client-Seite erfordert. Da jedoch jedes Access Token an einen bestimmten Resource Server gebunden ist, muss der Client auch ein einzelnes ressourcenserverspezifisches Access Token beschaffen. (Ressource Indicators, wie in [RFC8707] spezifiziert, können dabei helfen.) [TOKEN-BINDING] hatte die gleiche Eigenschaft, da dem Access Token verschiedene Token-Bindungs-IDs zugeordnet werden müssen. Die Verwendung von Mutual TLS for OAuth 2.0 [RFC8705] ermöglicht es einem Client hingegen, das Access Token auf mehreren Resource Servern zu verwenden.

Es ist zu beachten, dass `audience`-beschränkungen – oder allgemein gesagt, eine Angabe des Clients gegenüber dem Authorization Server, wo er das Access Token verwenden möchte – zusätzliche Vorteile bieten, die über den Schutz vor Token-Leaks hinausgehen. Sie ermöglichen es dem Authorization Server, ein anderes Access Token zu erstellen, dessen Format und Inhalt speziell für den jeweiligen Server geprägt sind. Dies hat enorme funktionale und datenschutzrechtliche Vorteile bei Implementierungen, die strukturierte Access Tokens verwenden.

4.10. 3. Diskussion: Verhinderung von Leaks über Metadaten

Ein Authorization Server könnte dem Client zusätzliche Informationen über die Orte bereitstellen, an denen die Verwendung seiner Access Token sicher ist. Dieser Ansatz und die Gründe, warum er nicht empfohlen wird, werden im Folgenden erläutert.

In der einfachsten Form würde dies erfordern, dass der Authorization Server eine Liste seiner bekannten Resource Server veröffentlicht, wie im folgenden Beispiel unter Verwendung eines nicht nicht standardmäßige Authorization Server Metadaten-Parameter `resource_servers` veranschaulicht:

```
HTTP/1.1 200 OK

Content-Type: application/json

{

    "issuer": "https://server.somesite.example",
    "authorization_endpoint": "https://server.somesite.example/authorize",

    "resource_servers“: [

        "email.somesite.example",
        "storage.somesite.example",
        "video.somesite.example"
```

```
    ]

    ...

}
```

Der Authorization Server könnte auch die URL(s) zurückgeben, für die ein Access Token gültig ist, wie im Beispiel und im nicht standardmäßigen Rückgabeparameter `access_token_resource_server` dargestellt:

```
HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "access_token_resource_server":
"https://hostedresource.somesite.example/path1",

    ...

}
```

Diese Abhilfemaßnahme würde sich darauf verlassen, dass der Client die Sicherheitsrichtlinie durchsetzt und Access Token nur an legitime Ziele sendet. Ergebnisse von Sicherheitsuntersuchungen im Zusammenhang mit OAuth (siehe beispielsweise [research.ubc] und [research.cmu]) deuten darauf hin, dass ein großer Teil der Client-Implementierungen Sicherheitskontrollen wie `state`-Prüfungen nicht oder nicht ordnungsgemäß implementiert. Daher ist es wahrscheinlich, dass auch eine Verlässlichkeit auf Clients zur Verhinderung von Phishing mit Access Token nicht gegeben ist. Angesichts des Verhältnisses von Clients zu Authorization Servern und Resource Servern wird es zudem als praktikablerer Ansatz angesehen, so viel wie möglich von der sicherheitsrelevanten Logik auf diese Server zu verlagern. Natürlich muss der Client zur Gesamtsicherheit beitragen. Es gibt jedoch alternative Gegenmaßnahmen, wie in den Kapiteln 4.10.1 und 4.10.2 beschrieben sind, die ein besseres Gleichgewicht zwischen den beteiligten Parteien herstellen.

4.11. Offene Umleitung

Die folgenden Angriffe können auftreten, wenn ein Authorization Server oder Client über einen offenen Redirector verfügt. Solche Redirection Endpoints werden manchmal implementiert, um beispielsweise eine Meldung anzuzeigen, bevor ein Benutzer zu einer externen Website umgeleitet wird, oder um Benutzer zurück zu einer URL umzuleiten, die sie vor der Unterbrechung besuchen wollten, z.B. durch eine Anmeldeaufforderung.

4.11.1. Client als offener Redirector

Clients DÜRFEN KEINE offenen Redirectors verwenden. Angreifer können offene Redirectors verwenden, um URLs zu erstellen, die auf den Client verweisen, und diese nutzen, um Autorisierungscodes und Access Token zu abzufangen, wie in Kapitel 4.1.2 beschrieben. Ein weiterer Missbrauchsfall ist die Erstellung von URLs, die scheinbar auf den Client verweisen. Dies könnte Benutzer dazu verleiten, der URL zu vertrauen und sie in ihrem Browser aufzurufen. Dies kann für Phishing missbraucht werden.

Um einen offenen Redirect zu verhindern, sollten Clients nur dann umleiten, wenn die Ziel-URLs zulässig sind oder wenn die Herkunft und Integrität einer Anfrage authentifiziert werden kann. Gegenmaßnahmen gegen offene Weiterleitungen werden von OWASP [owasp.redir] beschrieben.

4.11.2. Authorization Server als offener Redirector

Genau wie bei Clients könnten Angreifer versuchen, das Vertrauen eines Benutzers in den Authorization Server (und insbesondere dessen URL) für Phishing-Angriffe auszunutzen. OAuth-Authorization Server leiten Benutzer regelmäßig zu anderen Websites (die Clients) weiter, müssen dies jedoch auf sichere Weise tun.

Kapitel 4.1.2.1 von [RFC6749] verhindert bereits offene Weiterleitungen, indem er festlegt, dass der Authorization Server den Benutzeragenten im Falle einer ungültigen Kombination von `client_id` und `redirect_uri` NICHT automatisch weiterleiten darf.

Ein Angreifer könnte jedoch auch eine korrekt registrierte Redirection URI nutzen, um Phishing-Angriffe durchzuführen. Der Angreifer könnte beispielsweise einen Client über die dynamische Client-Registrierung [RFC7591] registrieren und einen der folgenden Angriffe ausführen:

1. Absichtlich eine fehlerhafte Autorisierungsanfrage senden, z. B. durch Verwendung eines ungültigen `scope`-Werts, wodurch der Authorization Server angewiesen wird, den Benutzeragenten auf seine Phishing-Website umzuleiten.
2. Absichtlich eine gültige Autorisierungsanfrage mit einer vom Angreifer kontrollierten `client_id` und `redirect_uri` senden. Nachdem sich der Benutzer authentifiziert hat, fordert der Authorization Server den Benutzer auf, seine Zustimmung zu der Anfrage zu erteilen. Wenn der Benutzer ein Problem mit der Anfrage bemerkt und diese ablehnt, leitet der Authorization Server den Benutzeragenten dennoch auf die Phishing-Website um. In diesem Fall wird der Benutzeragent unabhängig von der vom Benutzer ergriffenen Maßnahme auf die Phishing-Website umgeleitet.
3. Absichtlich eine gültige "stille" Authentifizierungsanforderung (`prompt=none`) mit `client_id` und `redirect_uri`, die vom Angreifer kontrolliert werden. In diesem Fall leitet der Authorization Server den Benutzeragenten automatisch auf die Phishing-Website um.

Der Authorization Server MUSS Vorkehrungen treffen, um diese Bedrohungen zu verhindern. Der Authorization Server MUSS immer zuerst den Benutzer authentifizieren und, mit Ausnahme des Anwendungsfalls der stillen Authentifizierung, den Benutzer bei Bedarf zur Eingabe seiner Anmeldedaten auffordern, bevor er ihn umleitet. Auf der Grundlage seiner Risikobewertung muss der Authorization Server entscheiden, ob er der Umleitungs-URI vertrauen kann oder nicht. Er könnte interne oder externe URI-Analysen berücksichtigen, um die Glaubwürdigkeit und Vertrauenswürdigkeit der hinter der URI stehenden Inhalte sowie die Quelle der Redirection URI und andere Client-Daten zu bewerten.

Der Authorization Server SOLLTE den Benutzeragenten nur dann automatisch umleiten, wenn er der Redirection URI vertraut. Wenn die URI nicht vertrauenswürdig ist, KANN der Authorization Server den Benutzer informieren und sich darauf verlassen, dass der Benutzer die richtige Entscheidung trifft.

4.12. 307 Umleitung

Am Authorization Endpoint besteht ein typischer Protokollablauf darin, dass der Authorization Server den Benutzer auffordert, seine Anmeldedaten in ein Formular einzugeben, das dann (unter Verwendung der HTTP-POST-Methode) an den Authorization Server zurückgesendet wird. Der Authorization Server überprüft die Anmeldedaten und leitet den Benutzeragenten bei Erfolg zum Redirection Endpoint des Clients weiter.

In [RFC6749] wird zu diesem Zweck der HTTP-Statuscode 302 (Found) verwendet, aber "jede andere Methode, die über den Benutzeragenten verfügbar ist, um diese Umleitung durchzuführen, ist zulässig". Wenn stattdessen der Statuscode 307 für die Umleitung verwendet wird, sendet der Benutzeragent die Anmeldedaten des Benutzers über HTTP POST an den Client.

Dadurch werden die sensiblen Anmeldedaten an den Client weitergegeben. Wenn der Client böswillig ist, kann er die Anmeldedaten verwenden, um sich beim Authorization Server als der Benutzer auszugeben.

Dieses Verhalten mag für Entwickler unerwartet sein, ist jedoch in Kapitel 15.4.8. von [RFC9110] definiert. Dieser Statuscode (307) erfordert nicht, dass der User-Agent die POST-Anfrage in eine GET-Anfrage umschreibt und dadurch die Formulardaten im POST-Body verwirft.

Im HTTP-Standard [RFC9110] erzwingt nur der Statuscode 303 eindeutig die Umschreibung der HTTP-POST-Anfrage in eine HTTP-GET-Anfrage. Bei allen anderen Statuscodes, einschließlich des gängigen 302, können Benutzeragenten sich dafür entscheiden, POST-Anfragen nicht in GET-Anfragen umzuschreiben, wodurch die Anmeldedaten des Benutzers dem Client offenbart werden. (In der Praxis zeigen die meisten Benutzeragenten dieses Verhalten jedoch nur bei 307-Weiterleitungen.)

Authorization Server, die eine Anfrage weiterleiten, die möglicherweise die Anmeldedaten des Benutzers enthält, dürfen daher NICHT den HTTP-Statuscode 307 für die Weiterleitung verwenden. Wenn für eine solche Anfrage eine HTTP Redirect (und nicht beispielsweise JavaScript) verwendet wird, sollte der Authorization Server den HTTP-Statuscode 303 (siehe „Sonstiges“) verwenden.

4.13. TLS-terminierende Reverse-Proxys

Eine gängige Bereitstellungsarchitektur für HTTP-Anwendungen besteht darin, den Anwendungsserver hinter einem Reverse-Proxy zu verbergen, der die TLS-Verbindung terminiert und die eingehenden Anfragen an die jeweiligen Anwendungsserverknoten weiterleitet.

In diesem Abschnitt werden einige Angriffspunkte dieser Bereitstellungsarchitektur im Zusammenhang mit OAuth aufgezeigt und Empfehlungen für Sicherheitskontrollen gegeben.

In einigen Situationen muss der Reverse-Proxy sicherheitsrelevante Daten zur weiteren Verarbeitung an die vorgelagerten Anwendungsserver weiterleiten. Beispiele hierfür sind die IP-Adresse des Request-Origins, ID-Tokens und authentifizierte TLS-Client-Zertifikate. Diese Daten werden in der Regel in HTTP-Headern weitergeleitet, die der vorgelagerten Anforderung hinzugefügt werden. Während es sich bei den Headern oft um benutzerdefinierte, anwendungsspezifische Header handelt, sind standardisierte Header-Felder für Client-Zertifikate und Client-Zertifikatsketten in [RFC9440] definiert.

Wenn der Reverse-Proxy alle von außen gesendeten Header weiterleitet, könnte ein Angreifer versuchen, die gefälschten Header-Werte direkt über den Proxy an den Anwendungsserver zu senden, um auf diese Weise die Sicherheitskontrollen zu umgehen. Beispielsweise ist es gängige Praxis von Reverse-Proxys, X-Forwarded-For-Header zu akzeptieren und einfach den Ursprung der eingehenden Anfrage hinzuzufügen (wodurch eine

Liste entsteht). Je nach der im Anwendungsserver ausgeführten Logik könnte der Angreifer einfach eine zulässige IP-Adresse zum Header hinzufügen und den Schutz damit unwirksam machen.

Ein Reverse-Proxy MUSS daher alle eingehenden Anfragen bereinigen, um die Authentizität und Integrität aller für die Sicherheit der Anwendungsserver relevanten Header-Werte sicherzustellen.

Wenn ein Angreifer Zugriff auf das interne Netzwerk zwischen dem Proxy und dem Anwendungsserver erhalten könnte, könnte er auch versuchen, die vorhandenen Sicherheitskontrollen zu umgehen. Daher ist es unerlässlich, die Authentizität der kommunizierenden Entitäten sicherzustellen. Darüber hinaus MUSS die Kommunikationsverbindung zwischen dem Reverse-Proxy und dem Anwendungsserver vor Abhören, Injection und Wiederholung von Nachrichten geschützt werden.

4.14. Schutz von Refresh Tokens

Refresh Tokens sind eine bequeme und benutzerfreundliche Möglichkeit, neue Access Tokens zu erhalten. Sie tragen auch zur Sicherheit von OAuth bei, da sie es dem Authorization Server ermöglichen, Access Tokens mit kurzer Lebensdauer und reduziertem Umfang auszugeben, wodurch die potenziellen Auswirkungen von Access Token Leaks reduziert werden.

4.14.1. Diskussion

Refresh Token sind ein attraktives Ziel für Angreifer, da sie den gesamten Umfang des einem bestimmten Client gewährten Zugriffs darstellen und nicht weiter auf eine bestimmte Ressource beschränkt sind. Wenn ein Angreifer in der Lage ist, ein Refresh Token abzufangen und erfolgreich wiederzugeben, kann er Access Token erstellen und diese verwenden, um im Namen des Resource Owners auf Resource Server zuzugreifen.

[RFC6749] bietet bereits einen robusten Basisschutz, indem es Folgendes verlangt:

- die Vertraulichkeit der Refresh Tokens während der Übertragung und Speicherung,
- die Übertragung von Refresh Tokens über TLS-geschützte Verbindungen zwischen Authorization Server und Client
- den Authorization Server, die Bindung eines Refresh Tokens an einen bestimmten Client aufrechtzuerhalten und zu überprüfen und diesen Client während der Token-Aktualisierung zu authentifizieren, sofern möglich, und
- dass Refresh Tokens nicht generiert, geändert oder erraten werden können.

[RFC6749] legt auch die Grundlage für weitere (implementierungsspezifische) Sicherheitsmaßnahmen, wie z. B. das Ablaufen und Widerrufen von Refresh Token sowie die Refresh Token Rotation, indem entsprechende Fehlercodes und Antwortverhalten definiert werden.

Diese Spezifikation enthält Empfehlungen, die über den Umfang von [RFC6749] hinausgehen, sowie Klarstellungen.

4.14.2. Empfehlungen

Authorization Server MÜSSEN auf der Grundlage einer Risikobewertung entscheiden, ob sie Refresh Tokens an einen bestimmten Client ausgeben. Wenn der Authorization Server beschließt, keine Refresh Tokens auszugeben, KANN der Client ein neues Access Token erhalten, indem er andere Grant-Types verwendet, z.B. den Authorization Code Grant. In einem solchen Fall kann der Authorization Server Cookies und persistente Berechtigungen verwenden, um die Benutzererfahrung zu optimieren.

Wenn Refresh Token ausgestellt werden, MÜSSEN diese Refresh Token an den Umfang und die Resource Server gebunden sein, wie vom Resource Owner genehmigt. Dies dient dazu, eine Ausweitung der Berechtigungen durch den legitimen Client zu verhindern und die Auswirkungen einer Offenlegung von Refresh Token zu verringern.

Für Confidential Clients [RFC6749] gilt bereits, dass Refresh Tokens nur von dem Client verwendet werden dürfen, für den sie ausgestellt wurden.

Authorization Server MÜSSEN eine der folgenden Methoden verwenden, um die Wiederholung von Refresh Tokens durch böswillige Akteure für Public Clients zu erkennen:

- **Sender-Constrained Tokens:** Der Authorization Server bindet das Refresh Token kryptografisch an eine bestimmte Client-Instanz, z.B. unter Verwendung von [RFC8705] oder [RFC9449].
- **Refresh Token Rotation:** Der Authorization Server stellt bei jeder Aktualisierung des Access Tokens ein neues Refresh Token aus. Das vorherige Refresh Token wird ungültig, aber die Informationen über die Beziehung werden vom Authorization Server gespeichert. Wenn ein Refresh Token kompromittiert und anschließend sowohl vom Angreifer als auch vom legitimen Client verwendet wird, wird einer von beiden ein ungültiges Refresh Token vorlegen, wodurch der Authorization Server über die Verletzung informiert wird. Der Authorization Server kann nicht feststellen, welche Partei das ungültige Refresh Token vorgelegt hat, aber er widerruft das aktive Refresh Token. Dadurch wird der Angriff gestoppt, allerdings muss der legitime Client eine neue Autorisierung einholen.
- Hinweis zur Implementierung: Die Berechtigung, zu der ein Refresh Token gehört, kann in den Refresh Token selbst kodiert werden. Auf diese Weise kann ein Authorization Server effizient feststellen, zu welcher Berechtigung ein Refresh Token gehört, und damit auch alle Refresh Tokens, die widerrufen werden müssen. Authorization Server MÜSSEN in diesem Fall die Integrität des Refresh Token-Werts sicherstellen, beispielsweise mithilfe von Signaturen.

Authorization Server KÖNNEN Refresh Token automatisch widerrufen, wenn ein Sicherheitsvorfall eintritt, z. B.:

- Passwortänderung oder
- Abmeldung beim Authorization Server.

Refresh Tokens SOLLTEN ablaufen, wenn der Client für eine gewisse Zeit inaktiv war, d.h. wenn das Refresh Token für eine gewisse Zeit nicht zum Abrufen neuer Access Tokens verwendet wurde. Die Ablaufzeit liegt im Ermessen des Authorization Servers. Es kann sich um einen globalen Wert handeln oder basierend auf der Client-Richtlinie oder der mit dem Refresh Token verbundenen Berechtigung (und deren Sensibilität) festgelegt werden.

4.15. Client, der sich als Resource Owner ausgibt

Resource Server können Zugriffskontrollentscheidungen auf der Grundlage der Identität eines Resource Owners, für den ein Access Token ausgestellt wurde, oder auf der Grundlage der Identität eines Clients im Client Credentials Grant treffen. Beispielsweise beschreibt [RFC9068] (JSON Web Token (JWT) Profile for OAuth 2. 0 Access Token) eine Datenstruktur für Access Token beschrieben, die einen wie folgt definierten Sub-Claim enthält:

Bei Access Token, die durch Berechtigungen mit Beteiligung eines Resource Owners erworben wurden, wie z.B. der Authorization Code Grant, SOLLTE der Wert von `sub` der Subjektkennung des Resource Owners entsprechen. Bei Access Token, die durch Berechtigungen ohne Beteiligung eines Resource Owners erworben

wurden, wie z.B. der Client Credentials Grant, SOLLTE der Wert von `sub` einer Kennung entsprechen, die der Authorization Server verwendet, um die Client-Anwendung zu identifizieren.

Wenn beide Optionen möglich sind, kann ein Resource Server die Identität eines Clients mit der Identität eines Resource Owners verwechseln. Wenn ein Client beispielsweise bei der Registrierung beim Authorization Server seine eigene `client_id` wählen kann, kann ein böswilliger Client diese auf einen Wert setzen, der einen Resource Owner identifiziert (z. B. einen `sub`-Wert, wenn OpenID Connect verwendet wird). Wenn der Resource Server nicht richtig zwischen Access Token unterscheiden kann, die unter Beteiligung des Resource Owners erhalten wurden, und solchen, bei denen dies nicht der Fall ist, kann der Client versehentlich auf Ressourcen zugreifen, die dem Resource Owner gehören.

Dieser Angriff betrifft potenziell nicht nur Implementierungen, die [RFC9068] verwenden, sondern auch ähnliche, maßgeschneiderte Lösungen.

4.15.1. Gegenmaßnahmen

Authorization Server SOLLTEN es Clients NICHT gestatten, ihre `client_id` oder andere Claims zu beeinflussen, die zu Verwechslungen mit einem echten Resource Owner führen könnten, wenn ein gemeinsamer Namensraum für Client-IDs und Benutzerkennungen existiert, wie beispielsweise im Beispiel für die `sub`-Angabe aus [RFC9068] in Kapitel 4.15 oben. Wenn dies nicht vermieden werden kann, MÜSSEN Authorization Server dem Resource Server andere Mittel zur Verfügung stellen, um zwischen den beiden Arten von Access Token zu unterscheiden.

4 .16. Clickjacking

Wie in Kapitel 4.4.1.9 von [RFC6819] beschrieben, ist die Autorisierungsanfrage anfällig für Clickjacking-Angriffe, auch als User Interface Redressing bezeichnet. Bei einem solchen Angriff bettet ein Angreifer die Benutzeroberfläche des Autorisierungsendpunkts in einen harmlosen Kontext ein. Ein Benutzer, der glaubt, mit diesem Kontext zu interagieren, beispielsweise durch Klicken auf Schaltflächen, interagiert stattdessen unbeabsichtigt mit der Benutzeroberfläche des Autorisierungsendpunkts. Das Gegenteil kann ebenfalls erreicht werden: Ein Benutzer, der glaubt, mit dem Autorisierungsendpunkt zu interagieren, könnte versehentlich ein Passwort in ein vom Angreifer bereitgestelltes Eingabefeld eingeben, das über die ursprüngliche Benutzeroberfläche gelegt wurde. Clickjacking-Angriffe können so gestaltet werden, dass Benutzer den Angriff kaum bemerken, beispielsweise durch die Verwendung von fast unsichtbaren Iframes, die über andere Elemente gelegt werden.

Ein Angreifer kann diesen Vektor nutzen, um die Authentifizierungsdaten des Benutzers zu erhalten, den Umfang der dem Client gewährten Zugriffsrechte zu ändern und möglicherweise auf die Ressourcen des Benutzers zuzugreifen.

Authorization Server MÜSSEN Clickjacking-Angriffe verhindern. In [RFC6819] werden mehrere Gegenmaßnahmen beschrieben, darunter die Verwendung des HTTP-Antwort-Header-Felds „X-Frame-Options“ und Frame-Busting-JavaScript. Darüber hinaus SOLLTEN Authorization Server auch Content Security Policy (CSP) Level 2 [W3C.CSP-2] oder höher verwenden.

Um wirksam zu sein, muss CSP am Autorisierungsendpunkt und gegebenenfalls an anderen Endpunkten verwendet werden, die zur Authentifizierung des Benutzers und zur Autorisierung des Clients dienen (z. B. dem Geräteautorisierungsendpunkt, Anmeldeseiten, Fehlerseiten usw.). Dies verhindert das Framing durch nicht autorisierte Ursprünge in Benutzeragenten, die CSP unterstützen. Der Client KANN das Framing durch

eine andere Quelle als die im Redirection Endpoint verwendete zulassen. Aus diesem Grund SOLLTEN Authorization Server Administratoren die Möglichkeit geben, zulässige Quellen für bestimmte Clients zu konfigurieren und/oder Clients diese dynamisch zu registrieren.

Durch die Verwendung von CSP können Authorization Server mehrere Ursprünge in einem einzigen Antwort-Header-Feld angeben und diese mithilfe flexibler Muster einschränken (weitere Informationen finden Sie unter [W3C.CSP-2]). Level 2 von CSP bietet einen robusten Mechanismus zum Schutz vor Clickjacking, indem Richtlinien verwendet werden, die den Ursprung von Frames (mithilfe von frame-ancestors) zusammen mit denen einschränken, die die Quellen von Skripten einschränken, die auf einer HTML-Seite ausgeführt werden dürfen (mithilfe von script-src). Ein nicht normatives Beispiel für eine solche Richtlinie ist in der folgenden Auflistung dargestellt:

```
HTTP/1.1 200 OK

Content-Security-Policy: frame-ancestors https://ext.example.org:8000
Content-Security-Policy: script-src ‚self‘
X-Frame-Options: ALLOW-FROM https://ext.example.org:8000

...
```

Da einige Benutzeragenten [W3C.CSP-2] nicht unterstützen, SOLLTE diese Technik mit anderen kombiniert werden, einschließlich der in [RFC6819] beschriebenen, es sei denn, solche älteren Benutzeragenten werden vom Authorization Server ausdrücklich nicht unterstützt. Selbst in solchen Fällen SOLLTEN dennoch zusätzliche Gegenmaßnahmen ergriffen werden.

4.17. Angriffe auf Kommunikationsflüsse innerhalb des Browsers

Wenn die Autorisierungsantwort mit browserinternen Kommunikationstechniken wie postMessage [WHATWG.postmessage_api] anstelle von HTTP Redirects gesendet wird, können Nachrichten versehentlich an bösartige Ursprünge gesendet oder aus bösartigen Ursprüngen injiziert werden.

4.17.1. Beispiele

Die folgenden nicht normativen Pseudocode-Beispiele für Angriffe unter Verwendung der browserinternen Kommunikation sind in [research.rub] beschrieben.

4.17.1.1. Unzureichende Beschränkung der Empfängerursprünge

Beim Senden der Autorisierungsantwort oder Token-Antwort über `postMessage` sendet der Authorization Server die Antwort an den Wildcard-Ursprung `\*` anstelle des Ursprungs des Clients. Wenn das Fenster, an das die Antwort gesendet wird, von einem Angreifer kontrolliert wird, kann der Angreifer die Antwort lesen.

```
window.opener.postMessage({
        code: "ABC",
        state: "123"
    },
```

```
    „\*“ // jede Website im "opener"-fenster kann die Nachricht empfangen
)
```

4.17.1.2. Unzureichende URI-Validierung

Beim Senden der Autorisierungsantwort oder Token-Antwort über `postMessage` überprüft der Authorization Server möglicherweise nicht die Herkunft des Empfängers anhand der Redirection URI und sendet die Antwort stattdessen beispielsweise an eine von einem Angreifer angegebene Herkunft. Dies entspricht dem in Kapitel 4.1 beschriebenen Angriff.

```
window.opener.postMessage({
        code: "ABC",
        state: "123"
    },
    "https://attacker.example" // vom Angreifer bereitgestellter Wert
)
```

4.17.1.3. Injection nach unzureichender Validierung der Absenderherkunft

Ein Client, der die Autorisierungsantwort oder Token-Antwort über `postMessage` erwartet, validiert möglicherweise nicht die Absenderherkunft der Nachricht. Dies kann es einem Angreifer ermöglichen, eine Autorisierungsantwort oder Token-Antwort in den Client einzuschleusen.

Im Falle einer böswillig eingeschleusten Autorisierungsantwort handelt es sich um eine Variante der in Kapitel 4.7 beschriebenen CSRF-Angriffe. Die in Kapitel 4.7 beschriebenen Gegenmaßnahmen gelten auch für diesen Angriff.

Im Falle einer böswillig eingeschleusten Token-Antwort können unter bestimmten Umständen die in Kapitel 4.10.1 beschriebenen Sender-Constrained Access Tokens den Angriff verhindern, aber in der Regel sind zusätzliche Gegenmaßnahmen erforderlich, wie sie in Kapitel 4.17.2 beschrieben sind.

4.17.2. Empfehlungen

Beim Vergleich der Origins der Client-Empfänger mit den vorab registrierten Origins MÜSSEN Authorization Server eine exakte Zeichenfolgenübereinstimmung verwenden, wie in Kapitel 4.1.3 beschrieben. Authorization Server MÜSSEN `postMessages` an vertrauenswürdige Client-Empfänger-Origins senden, wie im folgenden, nicht normativen Beispiel gezeigt:

```
window.opener.postMessage({
        code: "ABC",
        state: "123"
    },
    "https://client.example" // explizite Client-Herkunft verwenden
)
```

Wildcard-Herkünfte wie * in postMessage DÜRFEN NICHT verwendet werden, da Angreifer sie nutzen können, um die Browser-Nachrichten des Opfers an bösartige Herkunftsorte weiterzugeben. Beide Maßnahmen tragen dazu bei, die Weitergabe von Autorisierungscodes und Access Tokens zu verhindern (siehe Kapitel 4.1).

Clients MÜSSEN eine Injection von Nachrichten im Browser am Client-Empfänger-Endpunkt verhindern. Clients MÜSSEN eine exakte Zeichenfolgenübereinstimmung verwenden, um die Origin einer Nachricht im Browser mit der Origin des Authorization Servers zu vergleichen, wie im folgenden, nicht normativen Beispiel gezeigt:

```
window.addEventListener("message", (e) => {
    // genaue Authorization Server-Herkunft validieren
    if (e.origin === "https://honest.as.example") {
        // e.data.code und e.data.state verarbeiten
    }
})
```

Da nur In-Browser-Kommunikationsflüsse eine andere Kommunikationstechnik anwenden (d. h. postMessage anstelle von HTTP Redirect), MÜSSEN alle in Sicherheitsvorgaben für die Absicherung von APIs für den Schutzbedarf Normal aufgeführten Maßnahmen zum Schutz der Autorisierungsantwort gleichermaßen angewendet werden.

5. Verweise

5.1. Normative Verweise

[DIN 820-2]

Normungsarbeit - Teil 2: Gestaltung von Dokumenten (ISO/IEC Directives - Part 2:2021, modifiziert); Deutsche und Englische Fassung CEN-CENELEC-Geschäftsordnung - Teil 3:2022, https://www.dinmedia.de/de/norm/din-820-2/358748335

[draft-ietf-oauth-v2-1-14]

Hardt, D., Parecki, A., Lodderstedt, T., "OAuth 2.1 Authorization Framework", Version 14, 19. Oktober 2025, https://datatracker.ietf.org/doc/draft-ietf-oauth-v2-1/

[ISO29100]

ISO/IEC, "ISO/IEC 29100 Information technology – Security techniques – Privacy framework", https://www.iso.org/standard/85938.html.

[OIDC]

Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", 8 November 2014, http://openid.net/specs/openid-connect-core-1_0.html.

[RFC3986]

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, https://www.rfc-editor.org/info/rfc3986.

[RFC6749]

Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, https://www.rfc-editor.org/info/rfc6749.

[RFC6750]

Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, https://www.rfc-editor.org/info/rfc6750.

[RFC6819]

Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, https://www.rfc-editor.org/info/rfc6819.

[RFC7523]

Jones, M., Campbell, B., and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, DOI 10.17487/RFC7523, May 2015, https://www.rfc-editor.org/info/rfc7523.

[RFC8252]

Denniss, W. and J. Bradley, "OAuth 2.0 for Native Apps", BCP 212, RFC 8252, DOI 10.17487/RFC8252, October 2017, https://www.rfc-editor.org/info/rfc8252.

[RFC8414]

Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, https://www.rfc-editor.org/info/rfc8414.

[RFC8705]

Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, https://www.rfc-editor.org/info/rfc8705.

[RFC9068]

Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, https://www.rfc-editor.org/info/rfc9068.

[RFC9449]

Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, https://www.rfc-editor.org/info/rfc9449.

[RFC9700]

Lodderstedt, T., Bradley, J., Labunets, A., and D. Fett, "Best Current Practice for OAuth 2.0 Security", BCP 240, RFC 9700, DOI 10.17487/RFC9700, January 2025, https://www.rfc-editor.org/info/rfc9700.

5.2. Informative Verweise

[arXiv.1508.04324v2]

Mladenov, V., Mainka, C., and J. Schwenk, "On the security of modern Single Sign-On Protocols: Second-Order Vulnerabilities in OpenID Connect", arXiv:1508.04324v2, DOI 10.48550/arXiv.1508.04324, 7 January 2016, https://arxiv.org/abs/1508.04324v2/.

[arXiv.1601.01229]

Fett, D., Küsters, R., and G. Schmitz, "A Comprehensive Formal Security Analysis of OAuth 2.0", arXiv:1601.01229, DOI 10.48550/arXiv.1601.01229, 6 January 2016, https://arxiv.org/abs/1601.01229/.

[arXiv.1704.08539]

Fett, D., Küsters, R., and G. Schmitz, "The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines", arXiv:1704.08539, DOI 10.48550/arXiv.1704.08539, 27 April 2017, https://arxiv.org/abs/1704.08539/.

[arXiv.1901.11520]

Fett, D., Hosseyni, P., and R. Küsters, "An Extensive Formal Security Analysis of the OpenID Financial-grade API", arXiv:1901.11520, DOI 10.48550/arXiv.1901.11520, 31 January 2019, https://arxiv.org/abs/1901.11520/.

[bug.chromium]

"Referer header includes URL fragment when opening link using New Tab", Chromium Issue Tracker, Issue ID: 40076763, https://issues.chromium.org/issues/40076763.

[JWT-ENCODED-STATE]

Bradley, J., Lodderstedt, T., and H. Zandbelt, "Encoding claims in the OAuth 2 state parameter using a JWT", Work in Progress, Internet-Draft, draft-bradley-oauth-jwt-encoded-state-09, 4 November 2018, https://datatracker.ietf.org/doc/html/draft-bradley-oauth-jwt-encoded-state-09.

[OAuth.Post]

Jones, M. and B. Campbell, "OAuth 2.0 Form Post Response Mode", The OpenID Foundation, 27 April 2015, https://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html.

[OAuth.Responses]

de Medeiros, B., Ed., Scurtescu, M., Tarjan, P., and M. Jones, "OAuth 2.0 Multiple Response Type Encoding Practices", The OpenID Foundation, 25 February 2014, https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html.

[OpenID.Core]

Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 2", The OpenID Foundation, 15 December 2023, https://openid.net/specs/openid-connect-core-1_0.html.

[OpenID.Discovery]

Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 2", The OpenID Foundation, 15 December 2023, https://openid.net/specs/openid-connect-discovery-1_0.html.

[OpenID.JARM]

Lodderstedt, T. and B. Campbell, "Financial-grade API: JWT Secured Authorization Response Mode for OAuth 2.0 (JARM)", The OpenID Foundation, 17 October 2018, https://openid.net/specs/openid-financial-api-jarm.html.

[owasp.redir]

OWASP Foundation, "Unvalidated Redirects and Forwards Cheat Sheet", OWASP Cheat Sheet Series, https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html.

[research.cmu]

Chen, E., Pei, Y., Chen, S., Tian, Y., Kotcher, R., and P. Tague, "OAuth Demystified for Mobile Application Developers", CCS '14: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 892-903, DOI 10.1145/2660267.2660323, November 2014, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/OAuthDemystified.pdf.

[research.jcs_14]

Bansal, C., Bhargavan, K., Delignat-Lavaud, A., and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis", Journal of Computer Security, vol. 22, no. 4, pp. 601-657, DOI 10.3233/JCS-140503, 23 April 2014, https://www.doc.ic.ac.uk/~maffeis/papers/jcs14.pdf.

[research.rub]

Jannett, L., Mladenov, V., Mainka, C., and J. Schwenk, "DISTINCT: Identity Theft using In-Browser Communications in Dual-Window Single Sign-On", CCS '22: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, DOI 10.1145/3548606.3560692, 7 November 2022, https://dl.acm.org/doi/pdf/10.1145/3548606.3560692.

[research.rub2]

Fries, C., "Security Analysis of Real-Life OpenID Connect Implementations", Master's thesis, Ruhr-Universität Bochum (RUB), 20 December 2020, https://www.nds.rub.de/media/ei/arbeiten/2021/05/03/masterthesis.pdf.

[research.ubc]

Sun, S.-T. and K. Beznosov, "The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems", Proceedings of the 2012 ACM conference on Computer and communications security (CCS '12), pp. 378-390, DOI 10.1145/2382196.2382238, October 2012, https://css.csail.mit.edu/6.858/2012/readings/oauth-sso.pdf.

[research.udel]

Liu, D., Hao, S., and H. Wang, "All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records", CCS '16: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1414-1425, DOI 10.1145/2976749.2978387, 24 October 2016, https://dl.acm.org/doi/pdf/10.1145/2976749.2978387.

[RFC7591]

Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, https://www.rfc-editor.org/info/rfc7591.

[RFC7636]

Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, https://www.rfc-editor.org/info/rfc7636.

[RFC8707]

Campbell, B., Bradley, J., and H. Tschofenig, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, https://www.rfc-editor.org/info/rfc8707.

[RFC9101]

Sakimura, N., Bradley, J., and M. Jones, "The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)", RFC 9101, DOI 10.17487/RFC9101, August 2021, https://www.rfc-editor.org/info/rfc9101.

[RFC9110]

Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, https://www.rfc-editor.org/info/rfc9110.

[RFC9126]

Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, https://www.rfc-editor.org/info/rfc9126.

[RFC9207]

Meyer zu Selhausen, K. and D. Fett, "OAuth 2.0 Authorization Server Issuer Identification", RFC 9207, DOI 10.17487/RFC9207, March 2022, https://www.rfc-editor.org/info/rfc9207.

[RFC9396]

Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, https://www.rfc-editor.org/info/rfc9396.

[RFC9440]

Campbell, B. and M. Bishop, "Client-Cert HTTP Header Field", RFC 9440, DOI 10.17487/RFC9440, July 2023, https://www.rfc-editor.org/info/rfc9440.

[RFC9449]

Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, https://www.rfc-editor.org/info/rfc9449.

[TOKEN-BINDING]

Jones, M., Campbell, B., Bradley, J., and W. Denniss, "OAuth 2.0 Token Binding", Work in Progress, Internet-Draft, draft-ietf-oauth-token-binding-08, 19 October 2018, https://datatracker.ietf.org/doc/html/draft-ietf-oauth-token-binding-08.

[W3C.CSP-2]

West, M., Barth, A., and D. Veditz, "Content Security Policy Level 2", W3C Recommendation, December 2016, https://www.w3.org/TR/2016/REC-CSP2-20161215/. Latest version available at https://www.w3.org/TR/CSP2/.

[W3C.webappsec-referrer-policy]

Eisinger, J. and E. Stark, "Referrer Policy", 26 January 2017, https://www.w3.org/TR/2017/CR-referrer-policy-20170126/. Latest version available at https://www.w3.org/TR/referrer-policy/.

[WHATWG.CORS]

WHATWG, "CORS protocol", Fetch: Living Standard, Section 3.2, 17 June 2024, https://fetch.spec.whatwg.org/#http-cors-protocol.

[WHATWG.postmessage_api]

WHATWG, "Cross-document messaging", HTML: Living Standard, Section 9.3, 19 August 2024, https://html.spec.whatwg.org/multipage/web-messaging.html#web-messaging.

Anhang A. Hinweise zu Rechten am Dokument

Das vorliegende Dokument ist ein aus dem ‚Best Current Practice for OAuth 2.0 Security' abgeleitetes Dokument, weshalb die dafür genannten Copyright Regeln der IETF zu beachten sind:

```
Copyright (c) 2025 IETF Trust
```

Dieses Dokument unterliegt BCP 78 und den rechtlichen Bestimmungen des IETF Trust in Bezug auf IETF-Dokumente (https://trustee.ietf. org/license-info) in der zum Zeitpunkt der Veröffentlichung dieses Dokuments gültigen Fassung. Bitte lesen Sie diese Dokumente sorgfältig durch, da sie Ihre Rechte und Einschränkungen in Bezug auf dieses Dokument beschreiben. Aus diesem Dokument extrahierte Code-Komponenten müssen den in Abschnitt 4.e der rechtlichen Bestimmungen des Trusts beschriebenen Text der überarbeiteten BSD-Lizenz enthalten und werden ohne Gewährleistung gemäß der überarbeiteten BSD-Lizenz bereitgestellt.

Anhang B. Wesentliche Unterschiede zu BCP OAuth 2.0 Security [RFC9700]

Kapitel	**"Best Current Practice for OAuth 2.0 Security**	**Dieses Dokument**	**Gründe**
Allg.		Dieses Dokument basiert nur auf Kapitel 3 und 4 des Originals. Kapitel 2 im Schutzbedarf Normal enthalten.	
Allg.	OAuth 2.0	OAuth 2.0 und 2.1	OAuth 2.1 [draft-ietf-oauth-v2-1-14] ergänzt
Schlüsselwörter		DIN Norm	Deutsch