Migration von OpenID Connect in eine bestehende Anwendungslandschaft

Roland H. Steinegger[1] · Alexander Hotz[2]
Nadina Hintz[2] · Sebastian Abeck[1]

[1]Karlsruher Institut für Technologie (KIT)
Forschungsgruppe Cooperation & Management (C&M)
{steinegger | abeck}@kit.edu

[2]iC Consult Gesellschaft für Systemintegration und Kommunikation mbH
{alexander.hotz | nadina.hintz}@ic-consult.de

Zusammenfassung

OpenID Connect (OIDC) hat sich in den letzten Jahren als Standard zum Austausch von verifizierten Identitätsinformationen eines authentifizierten Benutzers und somit auch zur Authentifizierung in Web-Anwendungen etabliert. Ebenso steigt in Unternehmen der Druck, Endnutzern Dienste zur Verfügung zu stellen, wodurch diese sich bspw. per Social Login und somit OpenID Connect bei den Unternehmen authentisieren können sollen. Risikobehaftet ist dabei die Migration von der alten Authentifizierungsinfrastruktur hin zu OpenID Connect. Die Authentifizierung ist der zentrale Schlüssel zu den Web-Anwendungen des Unternehmens, deren Ausfall komplette Geschäftsbereiche lahmlegen kann. Um das Risiko zu minimieren sollten einerseits die verschiedenen Aspekte des Problems verstanden werden und andererseits ein etabliertes Vorgehen genutzt werden. Wir nutzen Sicherheitsmuster zur Beschreibung des Problems und dessen verschiedenen Ebenen und stellen ein Vorgehen zur schrittweisen Migration hin zu OpenID Connect in einer bestehenden Infrastruktur vor. Das Vorgehen und mögliche Probleme werden anhand zweier Fallbeispiele diskutiert. Diese betrachten die Einführung von OpenID Connect zur Authentifizierung im Rahmen einer Microservice-Architektur im universitären Umfeld einerseits und den Einsatz innerhalb eines großen Unternehmens mit zahlreichen Web-Anwendungen und mehreren tausend Benutzern andererseits.

1 Einleitung

Die Authentifizierung an Web-Anwendungen per OpenID Connect hat sich in den letzten Jahren zunehmend etabliert. Treiber des Standards sind unter anderem große Unternehmen mit Angeboten für Endkunden im Internet, was einer der Gründe für den Erfolg von OIDC ist. Für Unternehmen, die ihr Angebot für Endkunden auf- oder ausbauen wollen, bietet die Umstellung auf OIDC verschiedene Vorteile [SHM+16]. Beispielsweise reduziert sich der Verwaltungsaufwand von Endkundenkonten, da das primäre Konto beim externen Identity Provider liegt – das Unternehmen muss somit keine Anfragen wegen vergessener Passwörter bearbeiten.

Allerdings ist die Umstellung einer bereits existierenden Authentifizierungslösung auf OIDC kein leichtes Unterfangen. Die Authentifizierung ist eine zentrale Funktionalität für Web-Anwendungen, die Endkundendienste basierend auf ihren Kundendaten anbieten; funktioniert

P. Schartner · A. Baumann (Hrsg.) · D•A•CH Security 2017 · syssec (2017) 81-91.

diese nicht, kann auch der Dienst nicht in Anspruch genommen werden. Das Risiko von Problemen bei der Umstellung sollte somit möglichst gering gehalten werden.

Um das Risiko besser einschätzen zu können, ordnen wir OIDC in eine Schichtenarchitektur ein und zeigen mit Sicherheitsmustern die Problemstellen bei einer Umstellung auf. Anhand dieser Einordnung beschreiben wir ein schrittweises Vorgehen zur Einführung von OIDC. Zwei Fallstudien sollen Probleme und mögliche Lösungen praxisnah aufzeigen. Die Fallstudien sind so gewählt, dass sie unterschiedliche Problemfelder aufzeigen: Einerseits die komplette Umstellung auf OIDC in einem größeren Unternehmen und andererseits den Parallelbetrieb des Standards einhergehend mit der Einführung einer Microservice-Architektur im universitären Umfeld.

2 Grundlagen und verwandte Arbeiten

Dieser Abschnitt beleuchtet einerseits die technischen Grundlagen zu OIDC und gibt andererseits einen kurzen Überblick über bereits getätigte Arbeiten in diesem Bereich.

OIDC wurde im Februar 2014 durch die OpenID Foundation als Standard verabschiedet [SBJ+14]. Trotz der Namensähnlichkeit zum ebenfalls existierenden OpenID-Standard ist OpenID Connect nicht abwärtskompatibel zu ersterem [Sir14].

Im Gegensatz zu OAuth 2.0, welches ein Framework zur delegierten Autorisierung bereitstellt, liegt der Einsatzzweck von OIDC auf der Bereitstellung von Authentifizierungsinformationen samt Single-Sign-On-Funktionalität. Hierzu baut es auf OAuth 2.0 auf und erweitert dieses um einige neue Bestandteile, welche in diesem Kapitel näher vorgestellt werden.

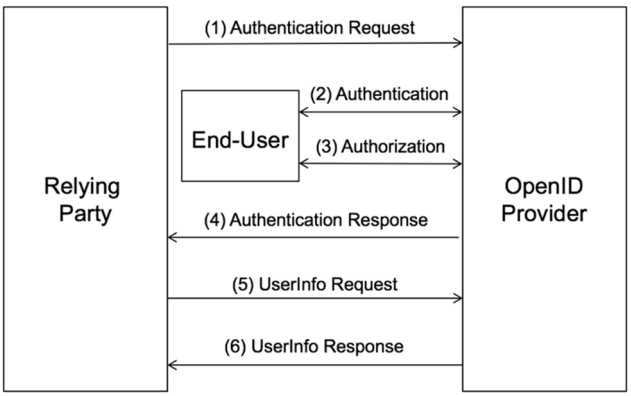

Abb. 1: Übersicht über die bei OIDC miteinander interagierenden Akteure und deren Kommunikation

Insgesamt beschreibt das Protokoll die Interaktion zwischen drei Akteuren (siehe Abbildung 1): dem End-User, der sogenannten Relying Party (im Folgenden mit RP abgekürzt) und dem OpenID Provider (im Folgenden mit OP abgekürzt). Der End-User ist dabei eine Person, die sich gegenüber der RP authentisieren möchte. Anstatt diese Authentisierung direkt bei der RP durchzuführen, leitet dieser den End-User in einem ersten Schritt zunächst an den OP weiter. Dieser präsentiert dem End-User eine Anmeldemaske, mit welcher sich der End-User mittels seiner Credentials authentisiert. Ob es sich hierbei um die klassische Kombination aus Benutzername und Passwort handelt oder zusätzlich ein zweiter Faktor herangezogen wird, ist nicht Teil der OIDC-Spezifikation und liegt im Ermessen des OP. So wäre es denkbar, dass für besonders sensible Ressourcen ein zweiter Faktor benötigt wird, für Ressourcen mit regulärem Schutzniveau jedoch nicht.

Nachdem der OP den End-User authentifiziert hat, wird ersterer dazu aufgefordert, die von der RP angeforderten Berechtigungen, engl. „Scopes“, freizugeben. Ein solcher Scope repräsentiert entweder eine bestimmte Menge an Profilinformationen, beispielsweise Anschrift oder Geburtstag, oder aber Zugriffsrechte auf andere Ressourcen des End-Users, deren Zugriff der End-User der RP einräumen möchte. Das Konzept der Scopes ist identisch mit dem aus OAuth 2.0, mit dem Unterschied, dass OIDC einen reservierten Scope namens „openid“ einführt, welcher dem OP die Nutzung von OIDC signalisiert und gemäß Spezifikation zwingend vorgeschrieben ist.

Im Anschluss stellt der OP der RP mehrere Tokens aus: ein OAuth Access Token, optional ein OAuth Refresh Token sowie ein sogenanntes ID-Token, welche die zentrale Neuerung der OIDC-Spezifikation gegenüber OAuth 2.0 darstellt.

Das ID-Token beinhaltet und transportiert Informationen über die Authentifizierung des Benutzers in Form von Schlüssel-Wert-Paaren, sogenannten *Claims*, an die dazugehörige RP. Solche Claims können beispielsweise Aussagen über die Benutzerkennung, den Zeitpunkt der Authentifizierung oder auch die verwendete Authentifizierungsmethode treffen. Je nach angeforderten Scopes können zudem auch bestimmte Profilangaben des End-Users als Claims in das ID-Token aufgenommen werden. Die OIDC-Spezifikation gibt eine Liste von insgesamt zwanzig sogenannter „Standard Claims“ vor, welche sich jedoch um eigene ergänzen lässt.

Während OAuth 2.0 keine konkreten Angaben darüber macht, in welchem Format ein Access oder Refresh Token vorliegen muss, schreibt OIDC für das ID-Token die Verwendung eines JSON Web Tokens (JWT) vor. Ein solches JWT besteht aus den bereits erwähnten Claims, welche in das JWT in Form von JSON Web Signature (JWS)- oder JSON Web Encryption (JWE)-Nutzdaten eingebettet werden. Somit verfügt ein JWT stets über eine digitale Signatur zur Integritätssicherung. Sofern auch die Vertraulichkeit der Claims gewünscht ist, kann JWE zum Einsatz kommen, welches in der Regel zusätzlich digitale Signaturen einsetzt.

Die letzten beiden Schritte in Abbildung 1 zeigen die Nutzung des ebenfalls neu eingeführten UserInfo-Endpoints. Dieser kann durch die RP aufgerufen werden, um weitere Profilinformationen, welche sich nicht bereits im ID-Token befinden, abzurufen. Welche Claims der OP im ID-Token mitsendet und welche er über den UserInfo Endpoint bereitstellt, liegt in dessen Ermessen. Es ist somit möglich, dass das ID-Token nur die unmittelbar durch die OIDC-Spezifikation vorgeschrieben Claims enthält, alle anderen Claims jedoch über den UserInfo Endpoint bezogen werden müssen. Um die Nutzung des UserInfo-Endpoints vor unautorisierter Nutzung zu schützen, muss die RP ein gültiges Access Token, welches sie zuvor erhalten hat, vorzeigen.

Der in Abbildung 1 abstrakt dargestellte Protokollfluss besitzt in der Praxis drei unterschiedliche Ausprägungen in Form sogenannter *Flows*, welche sich zwar inhaltlich leicht voneinander unterscheiden, als Resultat jedoch immer einen bei der RP authentifizierten End-User vorsehen. Zur Wahl stehen der *Authorization Code Flow*, der *Implicit Flow* sowie der *Hybrid Flow*. Die ersten beiden Varianten sind dem Authorization Code Grant respektive dem Implicit Grant aus OAuth 2.0 angelehnt. Neu gegenüber OAuth 2.0 ist der Hybrid Flow, welcher zusätzliche Flexibilität zum Erhalt der Tokens bietet.

OIDC versucht somit einen ähnlichen Funktionsumfang wie auch die Security Assertion Markup Language (SAML) abzudecken – mit dem großen Unterschied, dass SAML auf XML,

OIDC jedoch konsequent auf JSON und einen REST-Architekturstil setzt. Die beiden letztgenannten Punkte sind Ausdruck der aktuellen Entwicklung im Web und stellen somit für viele Entwickler bereits vertraute Technologien dar.

Zu OIDC wurden nicht zuletzt aufgrund der Popularität des Standards eine Vielzahl an Publikationen verfasst. Hier kann festgestellt werden, dass sich ein Großteil dieser Arbeiten mit konzeptionellen Details, beispielsweise der Sicherheit des Protokolls, befasst und weniger die Probleme bei einer Migration innerhalb existierender Anwendungslandschaften betrachtet werden. Zur Unterstützung der Migration sind zumindest eine Einordnung in Architekturkonzepte und bestehende Lösungsansätze hilfreich. Mit unserer Einordnung in die Sicherheitsmusterlandschaft in [SBJ+14] sind wir bereits einen ersten Schritt in diese Richtung gegangen.

3 Vorgehen zur Einführung von OpenID Connect

In diesem Abschnitt ordnen wir einerseits die verschiedenen Aspekte der Authentifizierung mit OpenID Connect in eine Schichtenarchitektur gemäß [VAC+08] ein und beschreiben andererseits basierend auf diesem Wissen ein Vorgehen zur schrittweisen Einführung von OpenID Connect.

Bei der Einführung von OIDC in die Anwendungslandschaft eines Unternehmens sind verschiedene Aspekte zu beachten. In der Regel soll OIDC die bestehende Authentifizierungsinfrastruktur ersetzen. Dabei soll mit dem OIDC Provider ein zentraler Authentifizierungsdienst samt Single Sign-On etabliert werden. Jedoch kann auf Grund von Altsystemen auch ein paralleler Betrieb der alten und neuen Infrastruktur notwendig sein, wodurch eine gemeinsame Sitzung zwischen alter und neuer Infrastruktur sichergestellt werden muss.

3.1 Einordnung in Schichtenarchitektur

Diese Problematik wird anhand der Schichteneinordnung ersichtlich. OIDC ist in solchen Szenarien ein Protokoll auf Anwendungsebene, das eine Authentifizierung von Benutzern in Anwendungen ermöglicht – der Fall der Authentifizierung von Services wird nicht betrachtet.

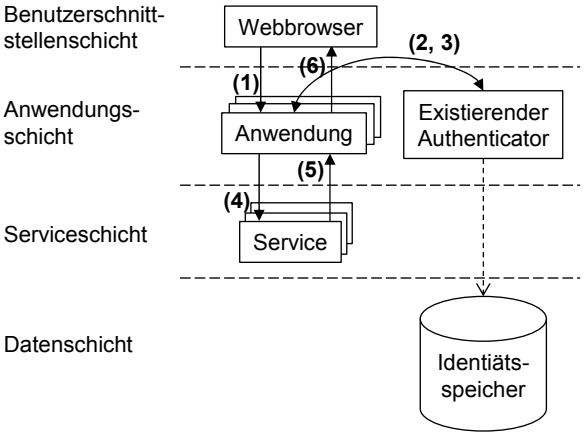

Abb. 2: Einordnung des Authentifizierungsablaufs in Architekturschichten

Abbildung 2 zeigt den Ablauf und die Schichten bei der Authentifizierung eines Benutzers in einer Web-Anwendung. Die Web-Anwendung leitet Anfragen von Benutzern an den Authenticator, in unserem Fall den OIDC Provider, zur Authentifizierung weiter. An diesem authenti-

siert sich der Benutzer, wodurch nach erfolgreicher Verifizierung der Daten eine Sitzung aufgebaut wird, beispielsweise indem Informationen in einem Cookie des Webbrowsers gespeichert werden. Die Daten des Benutzers lädt der OIDC Provider von einem Identitätsspeicher (engl. Identity Store) und nutzt sie sowohl zur Verifizierung der Identität als auch zur Weitergabe von Identitätsinformationen an die Web-Anwendung. Als Identitätsspeicher wird üblicherweise ein Verzeichnisdienst wie Active Directory verwendet; dieser ist der Datenschicht zuzuordnen und stellt eine einheitliche Schnittstelle zu Identitätsinformationen bereit, beispielsweise per Lightweight Directory Access Protocol (LDAP). Nach erfolgreicher Authentifizierung wird der Benutzer wieder an die Web-Anwendung weitergeleitet. Die weitergeleitete Anfrage enthält dabei bereits Authentifizierungs- und Identitätsinformationen.

Dieser Ablauf ist typisch für die externalisierte Authentifizierung in Web-Anwendungen und kann beispielsweise auch beim Shibboleth-Protokoll wiedergefunden werden. Der OIDC Provider nimmt die Rolle des Authenticators und des Identity Providers gemäß der gleichnamigen Sicherheitsmuster [Fern13] ein. Der Identitätsspeicher stellt ein weiteres Sicherheitsmuster zum Zugriff auf Identitätsinformationen auf Datenebene dar.

3.2 Schritte angelehnt an Sicherheitsmuster

Anhand der Sicherheitsmuster kann ein Vorgehen zur Integration von OIDC in eine bestehende Anwendungslandschaft abgeleitet werden. Im einfachsten Fall soll ein bestehender Authenticator durch einen neuen ersetzt werden. Dies entspricht der Einführung von OIDC als neue, zentrale Authentifizierungslösung. Der zweite Fall ist ein Parallelbetrieb von OIDC und einem weiteren Authenticator, der bereits zuvor im Unternehmen eingesetzt wurde, so dass die Web-Anwendungen den präferierten Authenticator wählen können. In beiden Fällen kann der Identitätsspeicher bestehen bleiben und wird von beiden Authenticators als Datenquelle genutzt. Dies stellt kein Problem dar, da zur Authentifizierung nur lesenden auf die Identitätsdaten zugegriffen werden muss. Die Migration auf Ebene der Datenschicht ist somit leicht möglich.

Problematisch kann hingegen die Migration auf Ebene der Anwendungsschicht sein. Web-Anwendungen sollen auf dieser Ebene bezüglich ihrer Authentifizierung von einem bestehenden Authenticator auf OIDC umgestellt werden. Wenn dies eine größere Zahl Web-Anwendungen betrifft, bietet sich als Migrationsstrategie das "Strangulieren" der Alt-Anwendung (engl. Strangler Application) nach Fowler [Fowl04, New15] zur Reduzierung des Fehlerrisikos an. Fowler beschreibt eine Lösung für das Problem der Einführung eines neuen Softwaresystems, das ein Altsystem ablösen soll. Es besagt, dass das Altsystem inkrementell durch das neue System ersetzt wird und es somit "stranguliert". In unserem Fall bedeutet dies, dass die Web-Anwendungen nach und nach von der alten Authentifizierungsmethode auf OIDC umgestellt werden. Auf diese Weise kann der neue OIDC Provider zuerst an eher unkritischen Web-Anwendungen getestet werden. Um bei diesem Vorgehen trotzdem Single Sign-On zu gewährleisten, müssen die Sitzungen der Authenticator untereinander abgeglichen werden.

Mit diesem Hintergrund besteht die Migration aus folgenden Schritten: 1) aufsetzen des OIDC Providers mit bestehendem Identitätsspeicher als Datenquelle, 2) abgleichen der Sitzungen zwischen den Authenticator, falls dies notwendig ist, 3) Umstellen der Web-Anwendungen auf OIDC und 4) abschalten der alten Authentifizierungslösung, falls kein Parallelbetrieb angestrebt wird.

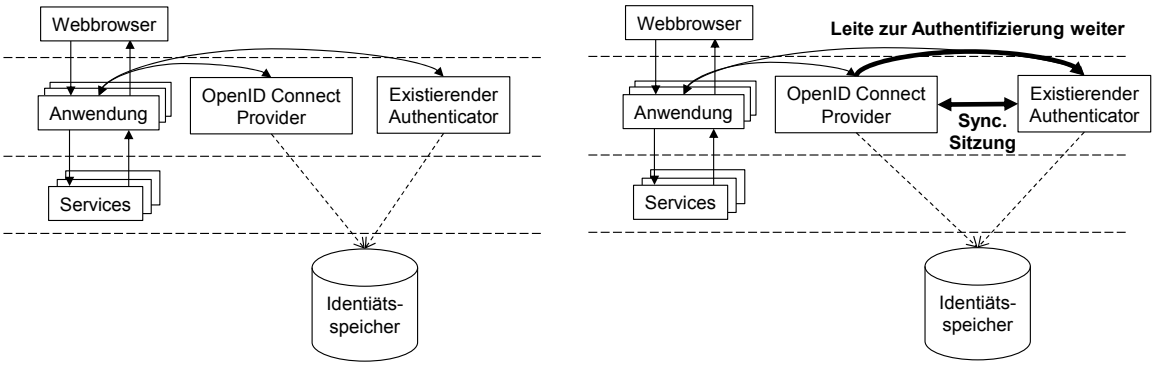

Abb. 3: Reihenschaltung der Authenticator oder Sitzungssynchronisation (links) und Parallelbetrieb von OpenID Connect und eines existierenden Authenticators (rechts).

1. Der erste Schritt birgt keine konzeptionellen Hürden und kann in der Regel problemlos durchgeführt werden. Auch die meisten Open-Source-Lösungen für OIDC Provider können meist in wenigen Tagen aufgesetzt werden, sofern es keine Probleme bei der Integration des existierenden Identitätsspeichers gibt.
2. Sofern ein Parallelbetrieb angestrebt wird, beispielsweise auch um das Strangler Pattern anzuwenden, sollten die Sitzungen der Authenticator angeglichen werden. Dies ist in der Regel kein einfach zu lösendes Problem, sofern die Umstellung für die Web-Anwendungen transparent sein soll. Abbildung 3 illustriert links zwei Lösungswege für dieses Problem. Entweder werden die Authenticator in Reihe geschaltet, so dass ein Authenticator Anfragen zur Authentifizierung an den anderen weiterleitet (oberer Pfeil in der Abbildung), oder die Implementierung der Authenticator wird angepasst, so dass deren Sitzungen synchronisiert werden (unterer Pfeil in der Abbildung). Oft ist der Eingriff in die Implementierung nicht möglich, beispielsweise weil der Quellcode des Systems nicht Open-Source ist oder kein Entwicklungs-Knowhow vorhanden ist, weswegen die Authenticator in der Regel in Reihe geschaltet werden.
3. Im dritten Schritt werden die Web-Anwendungen auf OIDC zur Authentifizierung umgestellt. Je nach Strategie geschieht diese Umstellung über einen kürzeren oder längeren Zeitraum. Abbildung 3 zeigt rechts den resultierenden Ablauf. Die einzelnen Web-Anwendungen erhalten eigene Konten samt Zugangsdaten (Client-ID und Client-Passwort), welche im OIDC Provider eingerichtet werden. Anschließend kann die Konfiguration der Web-Anwendungen mit diesen Zugangsdaten angepasst werden.
4. Schlussendlich kann das Altsystem abgeschaltet werden, vorausgesetzt ein Parallelbetrieb ist nicht vorgesehen. Die Anpassungen, die zum Abgleich der Sitzungen vorgenommen wurden, werden in diesem Schritt ebenfalls zurückgenommen.

4 Fallstudien

In diesem Abschnitt werden Fallstudien zur Anwendung des Vorgehens vorgestellt. Die erste Fallstudie beschäftigt sich mit der Einführung von OIDC in einem universitären Umfeld bei der ein Parallelbetrieb gemeinsam mit Shibboleth als bestehender Authentifizierungslösung vorgesehen ist. Die zweite Fallstudie stellt die Anwendung des Vorgehens in einem großen Unternehmen mit breit gefächerter Anwendungslandschaft vor. In diesem Umfeld wurde eine komplette Umstellung der Authentifizierung durchgeführt.

4.1 Migration in großem Unternehmen

Der Einsatz von OIDC in der Industrie insb. in Dax 30 Konzernen ist in den meisten Fällen nicht allein mit einer reinen Umsetzung bzw. Integration getan. Üblicherweise befinden sich in diesen Konzernen über viele Jahre monolithisch gewachsene Infrastrukturen, welche nicht ohne weiteres ersetzt oder angepasst werden können. Nachfolgend wird anhand Erfahrungswerten aus der Praxis beschrieben wie sich eine solche Umstellung auf OIDC realisieren lässt, weiter werden entsprechende Zusammenhänge beispielhaft aufgeführt.

Das einzusetzende OIDC kümmert sich rein um die Authentifizierung eines Benutzers gegenüber einer Anwendung, ohne dass die Anwendung Zugriff auf dessen Zugangsdaten erhält. Diese Aufgabe hat beispielweise bisher eine im Konzern bereits bestehende Software (nachfolgend als Software A bezeichnet) übernommen und die Identität des Benutzers zur weiteren Autorisierung der Zugriffe verwendet. Da Software A die Authentifizierung der Benutzer entzogen wird, kann dieser auch keine Autorisierung mehr vornehmen, wodurch weitere Maßnahmen im Rahmen der Umstellung nötig werden.

Neben den Möglichkeiten zur Integration von OIDC in die Anwendungen, müssen auch dazu passende Konzepte entwickelt werden.

4.1.1 Authentifizierung

Durch die Unterstützung der verschiedenen OAuth 2.0 Flows kann OIDC generell sowohl in Web- als auch mobilen Anwendungen integriert werden, um Benutzer zu authentifizieren (siehe Abbildung 4).

4.1.2 Session Handling

Sowohl das ID-Token als auch die UserInfo liegen nach Abschluss der Autorisierung auf dem Server des Client vor. Bisher wurde die Identität des Benutzers durch Software A in einem Cookie gespeichert, welches der User Agent automatisch bei jeder Anfrage an den Server mitschickt. Dieses Cookie ist nun nicht mehr vorhanden, weshalb sich die Anwendung um einen Ersatz kümmern muss. Dies geschieht weiterhin über Cookies, allerdings sollte nun beachtet werden, dass die ausgestellten Cookies mindestens die Lebensdauer des ID-Token aufweisen, um eine vorzeitige Re-Authentifizierung zu vermeiden

4.1.3 Umsetzung

Das grundsätzliche Szenario geht davon aus, dass die Anwendung die Informationen aus ID-Token und UserInfo-Endpunkt zwischenspeichert und so den Benutzer identifiziert.

Üblicherweise werden Anwendungen auf einem Application Server ausgeführt. Davor sorgt ein Webserver mit Software A für den Schutz durch Prüfung aller Anfragen. Durch dieses System kommen die Identitätsinformationen bereits vorbereitet in Form von einzelnen Headern bei der Anwendung an.

Um dieses Konzept beizubehalten und die Anwendung wenig bis gar nicht anpassen zu müssen, kann ein Gateway vorgelagert werden. Das Gateway würde einen OIDC-Flow zur Anwendung abfangen und für diese abwickeln. Die Anwendung würde in diesem Fall nur die Informationen der abgeschlossenen Authentifizierung wie bisher per Header erhalten.

4.1.4 Gateway

Um den Zugriff auf Backends, sowohl der App als auch der Web-Anwendung, zu regulieren, müssen diese durch eine vorgelagerte Reverse Proxy- bzw. Gateway-Komponente geschützt werden. Dieser Abschnitt legt einige Möglichkeiten zur Umsetzung einer solchen Komponente dar. In der Praxis hat sich gezeigt, dass Eigenentwicklungen in großen Unternehmen meist nicht zum Einsatz kommen, sondern kommerzielle Produkte eingesetzt werden. Der Vollständigkeit halber wird nachfolgend dennoch kurz auf die Möglichkeit der Eigenentwicklung eingegangen.

Bei einer Eigenentwicklung ist nicht ein vollständig eigens entwickeltes Produkt gemeint, sondern entweder ein Modul auf einem vorgelagerten Web Server, der als Reverse Proxy fungiert oder ein Filter im eigentlichen Application Server.

Aufgrund seiner hohen Verbreitung wird hier der Apache Web Server [Apache] als Beispiel gewählt. Dieser würde unter Zuhilfenahme des Moduls *mod_proxy* als Reverse Proxy agieren, während ein zweites selbst entwickeltes Modul für die Prüfung übergebener Access-Tokens sorgen würde. Die Aufgaben eines solchen Moduls wären im Einzelnen:

- Auswerten von Authorization-Header
- Abfrage von Informationen über das Token beim Authorization Server

Eine Implementierung im Application Server könnte in Form eines Filters erfolgen. Dieser würde gleichermaßen eingehende Tokens prüfen, weitere Informationen abrufen und für die Anwendung relevante Daten in deren Kontext übergeben.

In der Regel werden in der Industrie kommerzielle Produkte eingesetzt. Hierbei gibt es verschiedene Hersteller welche sich in bestimmten technischen Punkten unterscheiden und auch differenzierte preisliche Modelle haben.

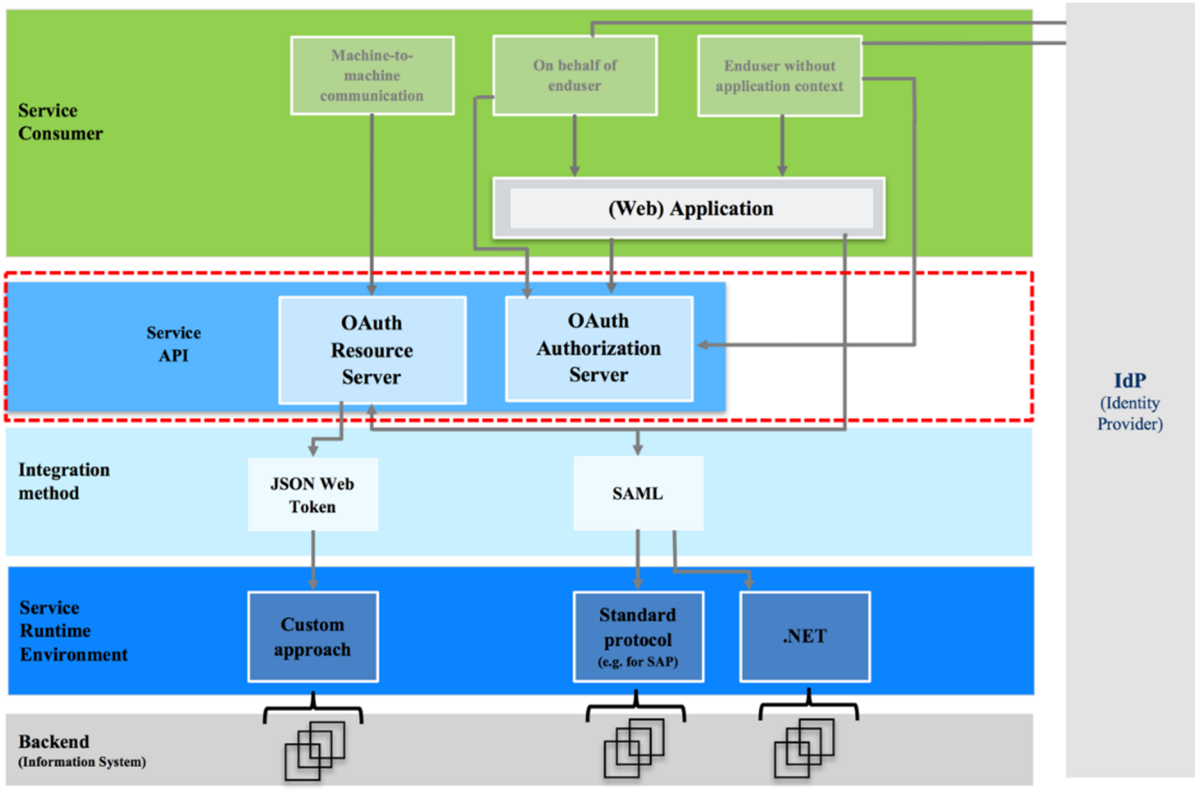

Abb. 4: Integrationsmöglichkeiten und Einordnung in Webanwendungsschichten

4.2 SmartCampus und Parallelbetrieb mit Shibboleth

Die SmartCampus-Anwendung ist eine in der Forschungsgruppe C&M am KIT entwickelte Web-Anwendung, die den Universitätsalltag für Studierende, Mitarbeitende und Gäste der Universität erleichtern soll. Zu diesen Services zählen zum Beispiel die Navigation auf dem Campus oder das Bereitstellen von zusätzlichen Informationsdiensten für Personen mit Behinderung. Die SmartCampus-Anwendung besitzt eine microservicebasierte Architektur und nutzt OIDC zur Authentifizierung sowie OAuth 2.0 zur Autorisierung an den diversen Microservices.

Gleichzeitig existiert am KIT bereits ein Identitätsanbieter auf Basis der freien Shibboleth-Software, welcher nicht nur von vielen anderen KIT-Anwendungen genutzt wird, sondern auch als Basis für die Föderation mit externen Institutionen dient. Shibboleth ist eine vom Internet2-Konsortium entwickelte freie Software, die einen Identitätsanbieter inklusive Single-Sign-On-Funktionalität auf Basis des SAML-Standards umsetzt. In dieser Hinsicht erfüllt es ähnliche Zwecke wie ein Identitätsanbieter auf OIDC-Basis, wenn auch unter Zuhilfenahme eines anderen zugrundeliegenden Protokolls.

Die Konstellation aus OIDC auf der einen und Shibboleth auf der anderen Seite sorgte dafür, dass zur Nutzung des SmartCampus und anderer KIT-Anwendungen stets zwei eigenständige Single-Sign-On-Sitzungen aufgebaut und somit auch eine zweifache Authentifizierung durch den Benutzer stattfinden musste.

Als Lösung kommt eine Anwendung der in Abbildung 3a) vorgestellten Reihenschaltung von Identitätsanbietern zum Einsatz. Anstatt auf eine Eigenlösung zu setzen, kommt die ebenfalls freie Software Keycloak von Red Hat zum Einsatz [Keycloak]. Keycloak ist eine Anwendung für das Identitäts- und Zugriffsmanagement, welches sich sowohl mit OIDC als auch mit SAML versteht. Des Weiteren bietet es die Möglichkeit, Verzeichnisdienste per LDAP anzubinden, um so auf weitere Attribute des Benutzers zuzugreifen. Die Software wird im Kontext des SmartCampus als zentraler OpenID Provider genutzt.

Keycloak kennt das Konzept des sogenannten „Identity Brokering“. Dieses sieht vor, dass ein Benutzer sich nicht zwingend bei Keycloak selbst, sondern auch an anderen Identitätsanbietern authentifizieren kann. Ein Beispiel, welches sich häufig im Internet beobachten lässt, ist ein Social Login auf Basis eines Google- oder Facebook-Accounts. Übertragen auf die vorgestellte Problemstellung lässt sich hier jedoch auch ein SAML-Identitätsanbieter und somit Shibboleth nutzen. Das grundsätzliche Vorgehen ist in Abbildung 5 dargestellt.

Sobald der Benutzer auf eine durch Keycloak geschützte Anwendung ohne gültige Sitzung zuzugreifen versucht, leitet Keycloak den Benutzer unmittelbar zu Shibboleth weiter. Im Hintergrund erstellt Keycloak hierbei einen SAML Authentication Request, den der Benutzer mithilfe seines Webbrowsers zu Shibboleth übermittelt. Nach erfolgter Authentifizierung des Benutzers beim Shibboleth-System, erstellt dieses zunächst eine Single-Sign-On-Sitzung und persistiert diese per Cookie im Webbrowser. Nach Weiterleitung der SAML-Assertion an Keycloak als SAML Service Provider prüft dieser die Assertion und stellt seinerseits ebenfalls eine eigene Single-Sign-On-Sitzung aus, welche erneut per Cookie im Webbrowser des Benutzers persistiert wird.

Das Ergebnis sind sowohl eine gültige Single-Sign-On-Sitzung bei Keycloak als auch bei Shibboleth. Der Benutzer kann zu beliebigen Anwendungen navigieren und einen Single Sign-On in Anspruch nehmen – unabhängig davon, ob diese Anwendung durch Keycloak oder Shibbo-

leth geschützt ist. Keycloak bietet zur Konfiguration des beschriebenen Vorgehens nach erfolgreicher Installation eine umfangreiche Weboberfläche an. Große Teile der bereitgestellten Funktionalität lassen sich über diese intuitiv einstellen.

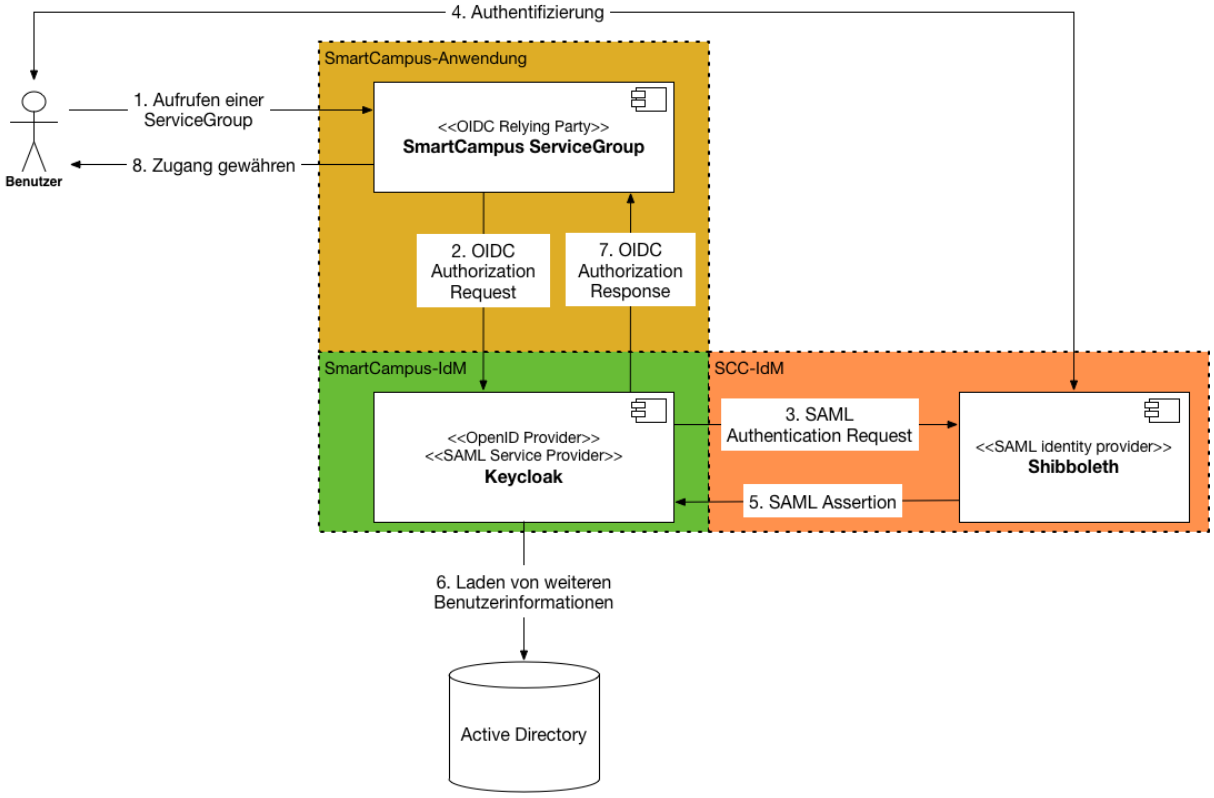

Abb. 5: Keycloak als Identity Broker zwischen OIDC- und SAML-Komponenten

Das Vorgehen hat zurzeit die Einschränkung, dass es nur dann zum Einsatz kommen kann, wenn der initiale Authentifizierungsvorgang über eine durch Keycloak geschützte Anwendung gestartet wird. Um auch den umgekehrten Weg zu unterstützen, wären darüber hinaus Anpassungen an Shibboleth notwendig, welche hier jedoch nicht weiter betrachtet werden. Ziel war es, die bestehende Umgebung möglichst unberührt zu lassen. Zusätzlich sollte darauf geachtet werden, die maximale Laufzeit der Single-Sign-On-Sitzungen bei beiden Identitätsanbietern auf einen einheitlichen Wert zu konfigurieren, sodass nicht eine der beiden Single-Sign-On-Sitzungen vor der anderen abläuft.

5 Zusammenfassung und Ausblick

In diesem Beitrag haben wir ein Vorgehen zur Migration einer bestehenden Authentifizierungslösung zu OIDC vorgestellt. Dieses Vorgehen erleichtert Software-Architekten und -Entwicklern die Migration, sorgt für ein gemeinsames Verständnis und reduziert die Risiken einer Umstellung auf OIDC. Als Grundlage des Vorgehens wurden OIDC und die verwendeten Sicherheitsmuster in eine Schichtenarchitektur eingeordnet. Dies erklärt, welche Komponenten in der Architektur bestehen bleiben und welche bei einer Migration ersetzt werden. Aufbauend auf diesem Wissen, wurde das schrittweise Vorgehen eingeführt und deren Anwendung in zwei

Fallstudien aufgezeigt. In den Fallstudien wird auf spezifische Probleme bei der Einführung in zwei unterschiedlichen Szenarien mit stark differierenden Randbedingungen eingegangen.

Obwohl OIDC ein noch vergleichsweiser junger Standard ist, ist bereits jetzt eine vergleichsweise hohe Verbreitung festzustellen. Für Unternehmen, die ihren Endkunden Web-basierte Dienste anbieten wollen, bietet OIDC eine Vielzahl von Vorteilen, wie zum Beispiel die vereinfachte Verwaltung von Benutzerkonten. Auf Grund der fortschreitenden Digitalisierung in etablierten Unternehmen ist eine weitere Verbreitung von OIDC abzusehen. Auch der Einsatz in Fahrzeugen, Smart-TVs (vgl. Device Flow bei OAuth 2.0) oder anderen Geräten im Internet der Dinge (engl. Internet of Things) ist auf Grund der Leichtgewichtigkeit des Protokolls wahrscheinlich. In diesen Bereichen werden weitere Detailprobleme auftreten, die weitere Arbeiten zur Einordnung in Architekturansätze und die Sicherheitsmusterlandschaft erforderlich machen.

Literatur

[Fern13] E. B. Fernandez: Security Patterns in Practice, Wiley, 2013.

[Fowl04] M. Fowler: Strangler Application, 2004. Abgerufen am 16.06.2017 von https://www.martinfowler.com/bliki/StranglerApplication.html

[New15] S. Newman: Building Microservices, O'Reilly Media Inc., 2015.

[SBJ+14] N. Sakimura, J. Bradley, M. Jones, B. Medeiros und C. Mortimore: OpenID Connect Core 1.0, The OpenID Foundation, 2014.

[SHM+16] R. H. Steinegger, N. Hintz, M. Müller, P. Schleier, S. Abeck: OAuth und OpenID Connect - Erfahrungen und Konzepte, D-A-CH Security, 2016.

[Sir14] P. Siriwardena: "Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE", Apress, 2014.

[VAC+08] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, T. Kehrer, U. Mehlig, U. Zdun: "Software-Architektur: Grundlagen-Konzepte-Praxis", Springer Science & Business Media, 2008.

[Keycloak] Open Source Identity and Access Management. Abgerufen am 16.06.2017 von http://www.keycloak.org/

[Apache] Apache HTTP Server Project. Abgerufen am 16.06.2017 von https://httpd.apache.org/