Mobile Geräte wie iPhones oder Android Phones, die über genügend Rechenleistung, aber ein deutlich größenreduziertes User-Interface (UI) verfügen, verändern die Bedeutung des Browsers dramatisch. Zugriffe auf Webseiten sind jedoch genauso wie der Zugriff auf interne Unternehmens-Anwendungen aufgrund der Gerätegröße schwerfällig und bisweilen sehr anstrengend.

Access Management 11*g* R2 für iOS und Cloud

Steffo Weber, ORACLE Deutschland B.V. & Co. KG

Viele Unternehmen wie Facebook und Amazon führen mittlerweile eigene Apps ein, die eine auf mobile Endgeräte angepasste User Experience versprechen. Dadurch verliert man jedoch den Komfort und die Sicherheit klassischer WebSSO-Lösungen wie OpenSSO, Oracle Access Manager etc., da diese im Allgemeinen nur mit einem Browser, aber nicht mit Apps funktionieren. Der Artikel zeigt, wie man mit der Oracle-Access-Management-Lösung in Apple XCode ein Single Sign-on für Apps einbauen kann.

Mit dem Release 2 des Access Manager 11*g* unterstützt Oracle erstmals mithilfe nativer Objective-C-Libraries das Single Sign-on (SSO) für mobile iOS-Anwendungen. Objective-C ist die für iOS und Mac OS X von Apple primär eingesetzte Sprache. Dies setzt natürlich voraus, dass man die Apps selbst entwickelt. Nachfolgend ist näher erläutert, warum SSO auf mobilen Endgeräten wie iPhones, iPads oder Android ein Problem ist, wie man das Problem lösen kann und für welche Art von Anwendungen diese Lösung geeignet ist. Der Artikel richtet sich sowohl an IT-Architekten als auch an Entwickler.

Android, iPhone und die Apps

Mobile Geräte (im Folgenden iOS- und Android-Geräte) sind anderen Bedrohungen ausgesetzt als klassische Desktop-Rechner. In diesem Kontext sind folgende Punkte des Nutzerverhaltens wichtig:

- Mobile Geräte werden von den Anwendern als persönliches Gerät angesehen und nicht als Firmen-Telefon oder -Laptop, auch wenn es sich um Eigentum des Unternehmens handelt. Anwender möchten Apps installieren sowie Fotos/Filme machen etc. Mobile-Device-Management-Lösungen (MDM) schützen diese Geräte gegenüber unbefugten Zugriffen.
- Nutzer erwarten eine smarte Bedienbarkeit der mobilen Geräte und ihrer Anwendungen (Apps). User Experience ist dabei sehr wichtig. Unerwünschte Pop-ups, schwer erreichbare Eingabefelder (wie HTML-Forms) und schlecht platzierte Werbung (oder Werbung überhaupt) können die Akzeptanz einer App stark reduzieren. Insbesondere für die Sicherheit, die oftmals gern mit erschwerter Bedienbarkeit erkauft wurde, ist dieser Punkt wichtig.
- Nutzer greifen meist von verschiedenen mobilen Geräten (Phone, Tablet) aus auf Firmen-Anwendungen zu. Hinzu kommt, dass Geräte von den Herstellern bei Verdacht auf einen Defekt schneller ausgetauscht werden – so sieht Apple für den Austausch eines iPhones aufgrund eines Display-Schadens eine Service-Pauschale vor und das Gerät ist innerhalb weniger Minuten gegen ein völlig anderes getauscht. Für die Sicherheit bedeutet dies, dass eine Zuordnung zwischen Nutzer und Gerät zwar sinnvoll ist, aber nicht allzu starr sein darf.
- Nutzer greifen von unterwegs auf (Geschäfts-)Anwendungen zu. Hierbei kann sich die IP-Adresse des mobilen Geräts ändern. Für eine Sicherheitslösung bedeutet dies, dass sie Artefakte oder Steuerungs-Komponenten nicht mehr an die IP-Adresse binden kann.

Nicht nur das Verhalten der Nutzer, auch das der Apps ist anders, als von Desktop-Anwendungen gewohnt. Mobile iOS und Android Apps nutzen (zumeist REST-basierte) Web-Services; die Wetter-App greift auf meteorologische Daten zu, die Navigation-App auf Karten-Dienste etc. Der klassische Präsentations-Layer verlagert sich zunehmend auf das mobile Endgerät. Diese Hintergrund-Dienste müssen aber so geschützt sein, dass sie nur durch autorisierte Anwendungen und authentisierte Anwender genutzt werden können. Hierzu sind entsprechende Security-Token, die nur nach vorheriger Authentisierung des Nutzers ausgestellt werden, bei jedem Web-Service-Aufruf mitzusenden. Es geht also beim SSO nicht um die Authentisierung gegenüber einer lokalen App, sondern um die Authentisierung gegenüber dem Web-Service, den die App nutzt.

Da mobile Endgeräte begrenzte Betriebsmittel (CPU, Speicher) haben, werden statt der klassischen, SOAP-basierten Web-Services die einfacheren, REST-basierten verwendet. Dies be-

deutet aber auch, dass eine Access-Management-Lösung für mobile Clients eine REST-basierte Schnittstelle für Identity-Dienste (Create/Update/Delete für User, Authentisierung, Token-Validierung) anbieten muss.

Klassisches WebSSO

WebSSO-Lösungen (wie OpenSSO, Oracle Access Manager oder Tivoli AM) nutzen den Cookie-Mechanismus des Browsers, um sogenannte „SSO-Token" zu speichern und bei Bedarf als Authentisierungsbeweis zu senden. Diese werden von Agenten (die je nach Produkt „WebGate", „AccessAgent" oder „AccessGate" heißen) geprüft. Bei nicht erfolgreicher Prüfung wird ein Zugriff auf die URL der Web-Applikation durch den Agent verhindert.

Das SSO zwischen den Web-Applikationen „otn.oracle.com/downloads" und „support.oracle.com/epmos" funktioniert folgendermaßen:

1. Der Browser greift auf „otn.oracle.com/downloads" zu. Da dies der erste Zugriff ist, muss sich der Nutzer authentisieren und erhält ein SSO-Cookie für die Domain „otn.oracle.com" (alternativ können auch Domain-Cookies ausgestellt werden). Mithilfe dieses Cookies kann im Folgenden auf „otn.oracle.com/downloads" zugegriffen werden.
2. Der Browser greift auf „support.oracle.com/epmos" zu. Das Host-Cookie aus dem obigen Request wird nicht mitgesendet. Der Agent, der die Web-Applikation schützt, stellt jedoch fest, dass der Nutzer bereits bei „otn.oracle.com" angemeldet ist, und stellt ein Cookie mit der Domain „support.oracle.com" aus.

Entscheidend für diesen Artikel ist, dass der Nutzer ein SSO-Token (hier für die Cookie Domain „support.oracle.com") erhält, weil er bereits ein anderes SSO-Token besitzt (hier: otn.oracle.com). Analog ließe sich unter Zuhilfenahme von SAML das SSO zwischen „otn.oracle.com" und „www.doag.de" implementieren; der Nutzer besitzt dann später noch ein zusätzliches Cookie mit Domain „www.doag.de", das zudem von einem völlig anderen Access-Management-Produkt (wie OpenSSO) kommen kann. Da der Nutzer über ein SSO-Token für „otn.oracle.com" verfügt, kann er dort auch ein SAML-Token erhalten und sich mit diesem über SAML-Protokoll am Access-Management-System (wie OpenSSO) der Site „www.doag.org" anmelden. Dort erhält er dann ein SSO-Token für „www.doag.org".

Zusammengefasst: WebSSO funktioniert auf Basis von SSO-Tokens, die als Cookie gespeichert werden. Der Browser schickt dann die erforderlichen Cookies mit dem HTTP-Request mit und das WebGate prüft die Gültigkeit des Cookies.

Mobile Anwendungen

Der obige SSO-Mechanismus mittels Cookies funktioniert, weil ein einziges Programm (Browser) auf verschiedene Web-Anwendungen (URLs) zugreift und dabei alle notwendigen SSO-Token in seinem Speicher vorrätig hat. Das Ganze geht schon dann nicht mehr, wenn der Nutzer für den zweiten Zugriff eine andere Browser-Instanz verwendet. Bei mobilen Apps haben wir jedoch genau dieselbe Situation wie bei der zweiten Browser-Instanz: Jede neue App, die gestartet wird, besitzt noch kein SSO-Token/Cookie. Die Frage ist also: Wie kann sich eine frisch gestartete App von einer anderen App ein bereits vorhandenes SSO-Token besorgen?

Die Oracle-Lösung besteht darin, dass sich jede App zunächst ein bestimmtes, auf dem mobilen Gerät vorhandenes SSO-Token besorgt, das dann zum Erhalt weiterer Token berechtigt. Das ist der gleiche Trick wie beim SSO zwischen „support.oracle.com" und „www.doag.org". Hierzu muss der Programmierer der iOS App nichts weiter tun, als ein paar Zeilen Code und eine von Oracle gelieferte iOS Library hinzuzufügen. Die App kontaktiert dann zunächst den Oracle

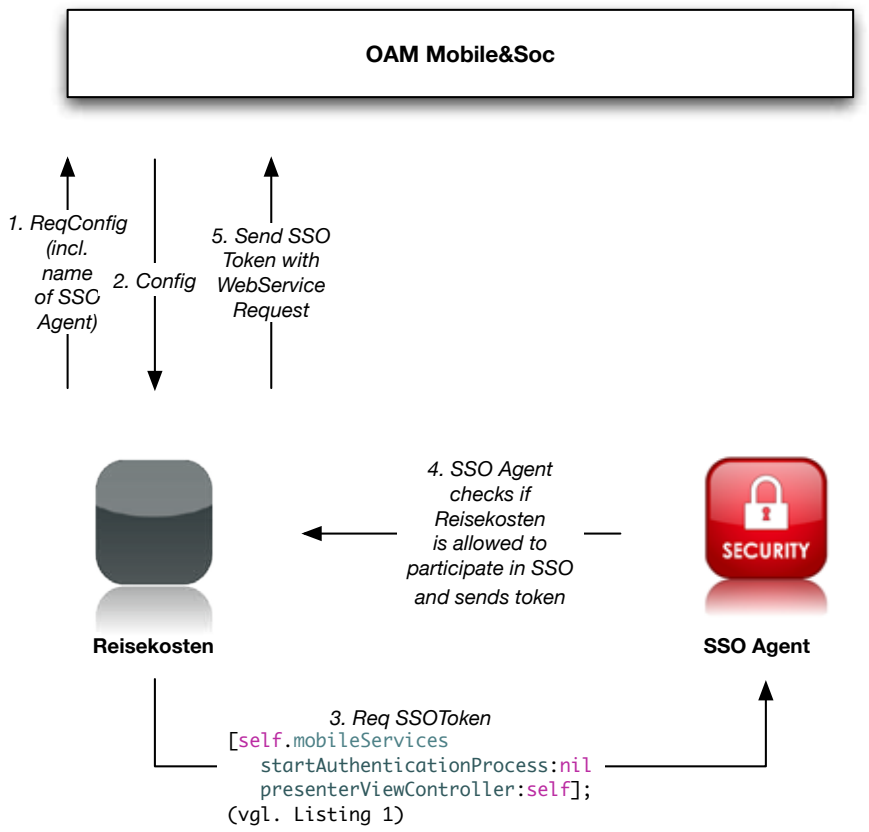

Abbildung 1: Eine iOS-App startet und erhält von der SSO-Agent-App das notwendige SSO-Token

Access Manager und erfragt, wo sie das initiale SSO-Token erhalten kann. Typischerweise erhält sie das Token per iOS-Interprozess-Kommunikation von einer SSO-Agent-App, die ebenfalls auf dem Gerät installiert ist. Abbildung 1 verdeutlicht den Prozess.

Der SSO-Agent prüft bei der Anfrage durch eine iOS-Applikation, ob diese berechtigt ist, ein SSO-Token zu erhalten. Falls ein Token vorhanden ist, wird dieses an die anfragende App übersendet. Oracle verwendet hierzu das sogenannte „OpenURL"-Schema für die iOS-Interprozess-Kommunikation.

Falls der SSO-Agent über kein aktuell gültiges Token verfügt, das er übersenden kann, fordert er den Benutzer auf, Log-in-Name und Passwort einzugeben. Mit diesen Informationen besorgt sich der SSO-Agent neue Token vom Access Manager.

XCode und iOS-Programmierung

Wie erwähnt, verbindet sich die Reisekosten-App zunächst mit dem Access-Manager, um zu erfragen, wie der SSO-Agent heißt. Aus Sicherheits- und Flexibilitätsgründen hat sich Oracle dazu entschieden, diese Konfigurations-Parameter zentral vorzuhalten. Dies geschieht durch Instanziierung eines Objekts der Klasse „OMMobileSecurityService". Sobald der Nutzer auf den „Log-in"-Button der Reisekosten-App drückt, versucht die App, mit dem SSO-Agent Kontakt aufzunehmen und ein entsprechendes Token anzufordern (siehe Listing 1).

Aus dem Objekt „mss" können nun der Name des authentisierten Nutzers, SSO-Token etc. erfragt werden (siehe Listing 2). Nachdem die App nun über ein SSO-Token verfügt, kann sie diese den Web-Service-Aufrufen hinzufügen. Dies ist über „curl" (für Demo-Umgebungen) oder mittels Werkzeugen wie dem „RESTKit" möglich. Die von Oracle gelieferte iOS Library „IDMMobileSDK" kapselt – bequem für den Programmierer – nicht nur die Interprozess-Kommunikation zwischen App und SSO-Agent, sondern auch die Kommunikation mit den REST-basierten Identity Services des Oracle Access Manager.

```
#import "IDMMobileSDK.h"

/* we have @property (nonatomic,retain) OMMobileSecurityService *mobile-
Services; from header */


- (void)connectToOICServerAndSetup
{
          ……
          OMMobileSecurityService *mss = [[OMMobileSecurityService alloc]
                                          initWithURL:self.oicURL
                                          appName:self.applicationName
                                          domain:self.oicServiceDomainName
                                          delegate:self];
          self.mobileServices = mss;
          ……
          UIBarButtonItem *rightButton = [[UIBarButtonItem alloc]
                                          initWithTitle:@"Login"
                                         style:UIBarButtonItemStyleBordered
                                          target:self
                                          action:@selector(doLogin:)];
}

- (IBAction)doLogin:(id)object
{
          …..
         NSError *error = nil;
         error = [self.mobileServices startAuthenticationProcess:nil
                                          presenterViewController:self];
```

Listing 1

```
OMUserRoleProfileService* ups = [mss userRoleProfileService];
 OMUserManager *um = [ups getuserManager];
 OMUser *user = [um searchUser:context.userName attributes:nil
shouldPreFetch:NO error:&error];
 self.user = user;
 [detailPaneController showProfileButton];
```

Listing 2

Identity-Dienste (nicht nur für Cloud-Anwendungen)

SaaS-Cloud-Anwendungen sind in ihren Erweiterungsmöglichkeiten zumeist limitiert; dies macht ihren ökonomischen Vorteil aus: „One size fits all". Sieht eine gemietete Cloud-Anwendung die Abfrage von Profildaten (etwa per LDAP) nicht vor, so kann sie nicht einfach vom Kunden verändert werden (durch Hinzufügen entsprechender Java-Klassen, die das gewünschte Feature bereitstellen). Die SaaS-Betreiber sehen oftmals nur serverseitiges JavaScript zur Anpassung der Anwendungen vor. Aus diesem Grund bietet Oracles Access Manager REST-APIs an, die sich leicht von JavaScript aus aufrufen lassen. Die APIs umfassen REST-Dienste für folgende Funktionen:

- Nutzer-Anlage, -Modifizierung und -Löschung
- Authentisierung und Autorisierung
- Token-Erstellung und -Validierung

Mithilfe dieser Dienste können Nutzer von der entfernten SaaS-Anwendung aus im LDAP des Access Manager an-

gelegt werden. Listing 3 zeigt, wie das (unter der Verwendung von „curl“) aussieht.

Selbstverständlich kann der Dienst so konfiguriert werden, dass sich der Aufrufer vorher authentisieren muss. Um ein UserToken (siehe Glossar) zu erhalten, wird ein „curl“-Befehl abgesetzt (siehe Listing 4).

Zum Erhalt des UserToken muss sich das Gerät mit einem ClientToken beziehungsweise einem ClientHandle identifizieren (der „curl“-Befehl, wie man zu diesem Token kommt, ist herausgelassen). Listing 5 zeigt, was der Service antwortet.

Das im Antwort-Parameter „X-Idaas-Rest-Token-Value“ enthaltene Token kann bei Bedarf vom Token-Validierungs-Dienst geprüft werden. Bis jetzt hat der REST-Consumer zwei Token angefordert und erhalten: ein ClientHandle (herausgelassen) und

```
curl -H “Content-Type: application/json“ --request POST
    http://solaris.oracle.demo:14100/oic_rest/rest/uprofile/people/
    -d ,{
      “uid“:“Steffo“,
      “description“:“Another Test User“,
      “lastname“:“Weber“,
      “commonname“:“Steffo Weber“,
      “firstname“:“Steffo“,
      “password“:“test123“,
      “mail“:“steffo@sun.com“}‘
```

Listing 3

```
curl -i -H “Content-Type: application/json“
   -H “X-IDAAS-SERVICEDOMAIN:SteffoMobileServiceDomain“
   -H ‘X-Idaas-Rest-Authorization:TOKEN <CLIENTREGHANDLE>‘
   --request POST
     http://solaris.oracle.demo:14100/oic_rest/rest/jwt/authenticate
-d
   ‘{“X-Idaas-Rest-Subject-Type“:“USERCREDENTIAL“,
     “X-Idaas-Rest-Subject-Username“:“fred@sun.com“,
     “X-Idaas-Rest-Subject-Password“:“abcd1234“,
     “X-Idaas-Rest-New-Token-Type-To-Create“:“USERTOKEN“}‘
```

Listing 4

Glossar

Authentisieren	Die Identität beglaubigen; dies ist ein Nachweis darüber, dass jemand der ist, der er vorgibt zu sein.
Autorisieren	Etwas erlauben wie den Zugriff auf einen Dienst, eine Datei etc. Sieht man von anonymen Nutzern ab, findet eine Authentisierung immer vor der Autorisierung statt.
Web-Service (SOAP/REST)	Ein auf einem (entfernten) Rechner angebotener Dienst (ähnlich dem Remote Procedure Call). SOAP-basierte Dienste verwenden XML-Dokumente zum Datenaustausch, REST-basierte Dienste verwenden einfache HTTP/URL-Strukturen wie „GET wetter/22761“, um die Wettervorhersage für 22761 Hamburg abzurufen.
Security Assertion Markup Language (SAML)	Ein XML-Dokument-Format, in dem mittels klassischer, asymmetrischer Kryptografie analog einem Ausweis Attribute des Nutzers (wie Name, E-Mail-Adresse, LDAP-Gruppen) beglaubigt aufgeführt werden. Das SAML-Protokoll legt fest, wie man (etwa unter Verwendung von HTTP POST/Redirect etc.) ein solches Dokument erhält und an Dritte (zum Beispiel zur Authentisierung) weiterleiten kann.
OAuth	Ein Protokoll sowohl zur Autorisierung als auch zur Authentisierung. Viele Social-Logins (Facebook, Twitter etc.) verwenden OAuth beziehungsweise eine Variante davon. Wichtig ist, dass OAuth lediglich das Protokoll definiert, aber nichts darüber aussagt, welches Format die übertragenen Token haben müssen.
UserToken	Ein kryptographisches Token, das den Nutzer identifiziert. Der Ausdruck stammt aus der OAuth-Welt.
AccessToken	Ein kryptografisches Token, das den Zugriff auf eine Web-Anwendung oder einen Web-Service ermöglicht. Das Access-Token erhält man üblicherweise, indem man ein gültiges UserToken vorweist. Für unterschiedliche Web-Services kommen verschiedene AccessToken zum Einsatz. Dies verhindert, dass ein Web-Service (etwa der REST-WS, den die Wetter-App aufruft) mit dem Token andere Dienste (wie Lufthansa Check-in) aufrufen kann. Das Konzept der Access Token stammt ebenfalls aus OAuth.
OpenURL-Scheme	Ein Mechanismus zur iOS-Interprozess-Kommunikation. Der Doppelklick auf eine Datei im Finder oder Explorer sendet die Datei (etwa ein Textdokument) an das Textverarbeitungs-Programm. Über den iOS-BundleID-Mechanismus kann der Sender vom Empfänger identifiziert werden (und umgekehrt). So wird verhindert, dass sensible Daten bei einem falschen Empfänger landen.
libIDMMobileSDK	Eine iOS-Bibliothek von Oracle, mit deren Hilfe Single Sign-on implementiert werden kann. Die Datei (inkl. Beispiele) kann unter www.oracle.com/technetwork/indexes/samplecode/oamms-samples-1872333.html heruntergeladen werden.

```
HTTP/1.1 200 OK Date: Tue, 16 Oct 2012 10:55:15
GMT Transfer-Encoding: chunked
Content-Type: application/json X-IDAAS-REST-VERSION: v1
Set-Cookie: JSESSIONID=dlgtQ99TQ4WFR8OQ1K7O5DJSGnOytQwRlBycxg6yTkxMQ9
xpRh2J!413534987; path=/; HttpOnly
X-ORACLE-DMS-ECID: 237e061257eacced:5e3e7f92:13a68e15a
8e:-8000-0000000000000061
X-Powered-By: Servlet/2.5 JSP/2.1
{“X-Idaas-Rest-Token-Value“:“<viele-lustige-Zeichen>“,
 “X-Idaas-Rest-User-Principal“:“fred@sun.com“,
 “X-Idaas-Rest-Provider-Type“:“JWT“,
 “X-Idaas-Rest-Token-Type“:“USERTOKEN“}
```

Listing 5

```
curl -H “Content-Type: application/json“
     -H “X-IDAAS-SERVICEDOMAIN:SteffoMobileServiceDomain“
     --request POST
     http://solaris.oracle.demo:14100/oic_rest/rest/oamtest/access
     -d ‘{“X-Idaas-Rest-Subject-Type”:”TOKEN”,
          “X-Idaas-Rest-Subject-Value”:”<USERTOKEN>”,
          “X-Idaas-Rest-New-Token-Type-To-Create”:”ACCESSTOKEN”,
          “X-Idaas-Rest-Application-Resource”:
               “http://wetter.oracle.demo:7777/meteo/hamburg”}’
```

Listing 6

ein UserToken. Als Nächstes fordert er ein AccessToken an. Die Unterscheidung zwischen diesen Token-Typen leitet sich direkt aus der OAuth-Spezifikation ab. Obwohl an dieser Stelle kein OAuth-Protokoll verwendet wird, sind einige Begriffe aus der OAuth-Welt in die Access-Manager-Welt eingeflossen. Zum Erhalt eines Access-Token übermittelt der REST-Consumer ein gültiges UserToken zusammen mit dem REST-Endpunkt des Dienstes, auf den er zugreifen möchte (siehe Listing 6).

Eine vollständige Beschreibung der Dienste inklusive Bespielaufrufe (mittels „curl“) stehen auf der Oracle-Webseite. Neben den skizzierten REST-Aufrufen zum Erhalt der verschiedenen Token sind noch weitere Aufrufe erforderlich. Man sieht schnell, dass mit „curl“ zwar alles geht, aber doch auf etwas umständliche Art und Weise. Genau hier hilft die erwähnte iOS-Bibliothek: Hinter den Kulissen des in Abschnitt „Xcode und iOS-Programmierung“ gezeigten Codes werden exakt die per „curl“ simulierten HTTP-Aufrufe verwendet.

Weitere Aspekte

Es wurde erläutert, wie die REST-basierten Web-Services von Oracle Access Manager (durch Cloud-Dienste) genutzt werden können und wie mithilfe einer von Oracle gelieferten Bib liothek, die die REST-Aufrufe kapselt, SSO bei mobilen iOS-Anwendungen erreicht werden kann. Eine entsprechende Bibliothek existiert auch für Java; für Android ist sie in Vorbereitung.

Aus der Liste mit den Besonderheiten im Hinblick auf Nutzerverhalten sind die Punkte „wechselnde Geräte“ und „wechselnde IP-Adressen“ offen. Oracle Access Management beinhaltet dafür ein Plug-in zum Oracle Adaptive Access Manager, das analog zu einer Art „Intrusion Detection“ prüft, ob das Gerät bekannt ist, ob die Änderung der IP-Adresse dem im mobilen Umfeld normalen Verhalten entspricht oder ob dem Gerät auf Basis von Attributen (wie Jailbreak, IMEI etc.) der Zugriff verwehrt werden soll.

Dr. Steffo Weber
steffo.weber@oracle.com