An Oracle White Paper
February 2013

Oracle Access Manager Mobile and Social

A Case Study – Piggy Bank

Introduction

Oracle Access Management's newly introduced Mobile and Social component provides a simple means to integrate Mobile applications with the security capabilities provided by Oracle's Identity and Access Management platform. The Mobile and Social feature extends Oracle Access Manager (OAM) and Oracle Adaptive Access Manager (OAAM) to secure mobile "Apps" running on iOS and Android devices and to secure services used by those Apps and protected by OAM and OAAM. OAM Mobile and Social also integrates with Oracle Enterprise Gateway to support use case where an Application Gateway is necessary or required.

Oracle Identity and Access Management has built in auditing, monitoring and platform level high availability capabilities. OAM Mobile and Social builds upon this existing infrastructure, leveraging those capabilities with no additional work on the part of developers or operations staff.

This white paper discusses the effort involved in executing a Proof of Concept with a major international bank. While the PoC exercise was real and the requirements described in this paper implemented, certain details have been changed to protect the identity of the bank and its security architecture and simplified for those new to OAM Mobile and Social.

The Proof of Concept detailed in this white paper involved three main tasks, including (1) creating a simple electronic banking application; (2) the REST/JSON services for the application; and (3) securing the application and services with the Oracle IAM technology stack.

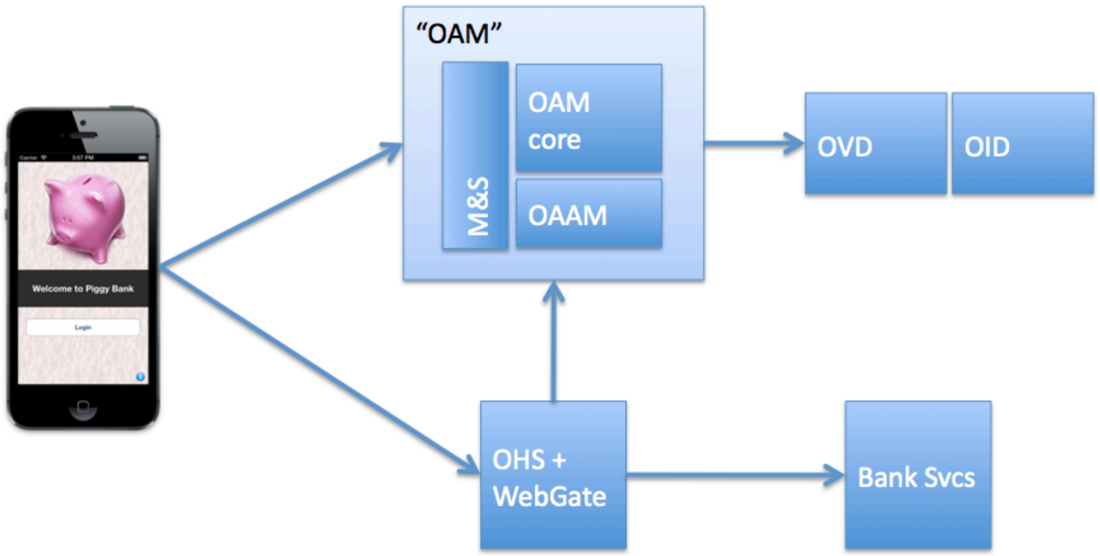
Piggy Bank Use Case

To demonstrate the Mobile and Social SDK and the Mobile and Social service, we created a custom "App" for iPhone and two simple REST services to be used by the application. The App itself is a simple e-banking application that allows a user to sign on, view savings and checking accounts, and transfer money between those accounts. The REST services are used to return a list of bank accounts with balances and process money transfers between accounts.

Piggy Bank System Architecture

The Piggy Bank architecture is very similar to most mobile application architectures – the mobile app interacts with a number of REST services to perform "login", "logout" actions and to execute transactions. In the case of the Piggy Bank architecture, some of the services are provided by Oracle's Mobile and Social Access Service and others are custom developed. The application-specific services list the user's bank accounts and their balances and allow the user to transfer money from one account to another.

The system architecture for this PoC is shown in the diagram below:



The client app communicates with the Mobile and Social Services through REST calls. Information on wire protocol syntax, data and usage of these calls is available in the OAM documentation. Oracle provides native libraries for common mobile platforms, which simplifies application configuration, device fingerprinting, login, and logout and other services; this library is called the Mobile and Social SDK and is available for download from OTN and/or E-Delivery. The Piggybank App uses the Mobile and Social SDK for all communication. Developers can also interact directly with the REST services.

The box marked "OAM" contains three main sets of services:

- Mobile and Social Services used by the client application through the Mobile and Social (M&S) SDK
- Oracle Access Manager core services used by OAM WebGates to protect web servers, communicate with LDAP directories to authenticate users and make authorization decisions, when users attempt to access resources.
- Oracle Adaptive Access Manager services are used to fingerprint devices, support "black listing" of lost or stolen devices and provide strong authentication via One Time Password (OTP) or Knowledge-Based Authentication (KBA)

The box marked "Bank Svcs" contains a set of REST services, which in this case are written in Java and are running inside another separate WebLogic Server instance.

An Oracle HTTP Server with an OAM WebGate is located between the Mobile device and the Bank Services. The WebGate operates as a Policy Enforcement Point (checking whether the accessed resource is protected, validating the user's OAM session, and then checking whether the user is authorized to access the resource) just as it would with any other WebGate-protected resource. The only difference in this case is that the session information is provided by the M&S SDK rather than the normal browser HTTP redirection flow. The Mobile and Social SDK automatically manages the security tokens needed to access resources protected by an OAM 11gR2 WebGate.

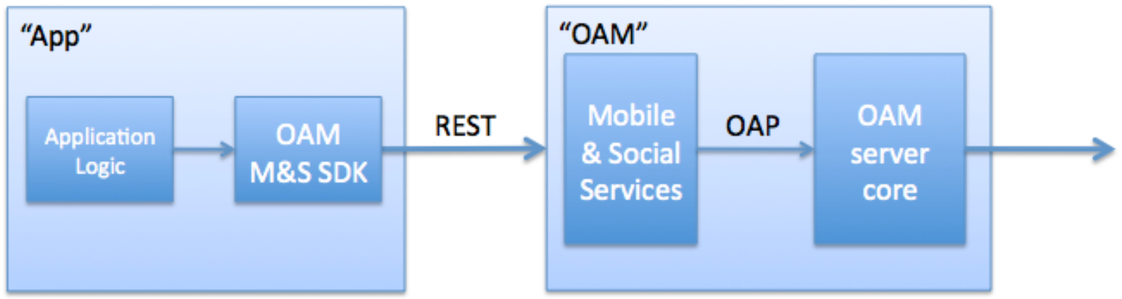
Mobile and Social Architecture

OAM's Mobile and Social services are layered on top of the existing OAM server and provide a number of additional capabilities. The PiggyBank PoC utilized three of these capabilities – login, logout, and access to a URL protected by an OAM WebGate.

To authenticate users the interaction is as follows:

- The iOS Application calls the OAM Mobile and Social SDK via API calls.
- The M&S SDK makes calls to the Mobile and Social Services via REST over HTTP.
- The Mobile and Social Services in turn call the OAM services through the Oracle Access Manager Protocol (OAP).

The architecture diagram below shows the interactions at a high level:

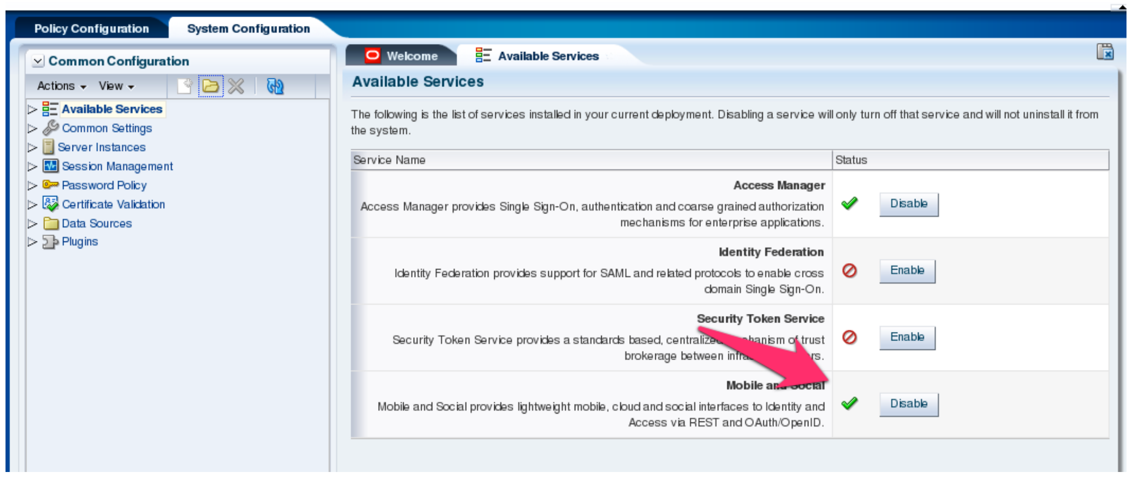


Mobile and Social Configuration

Mobile and Social Services are installed as part of Oracle Identity and Access Management 11gR2. The Administrator's Guide for Oracle Access Management discusses the configuration settings for Mobile and Social in detail. This white paper does not discuss all possible settings; rather it focuses on those used for the Piggy Bank App.

Enabling Mobile and Social Services

After installing the software the Mobile and Social services are enabled in the Oracle Access Management Console by navigating to the System Configuration -» Available Services and clicking the "Enable" button next to Mobile and Social.

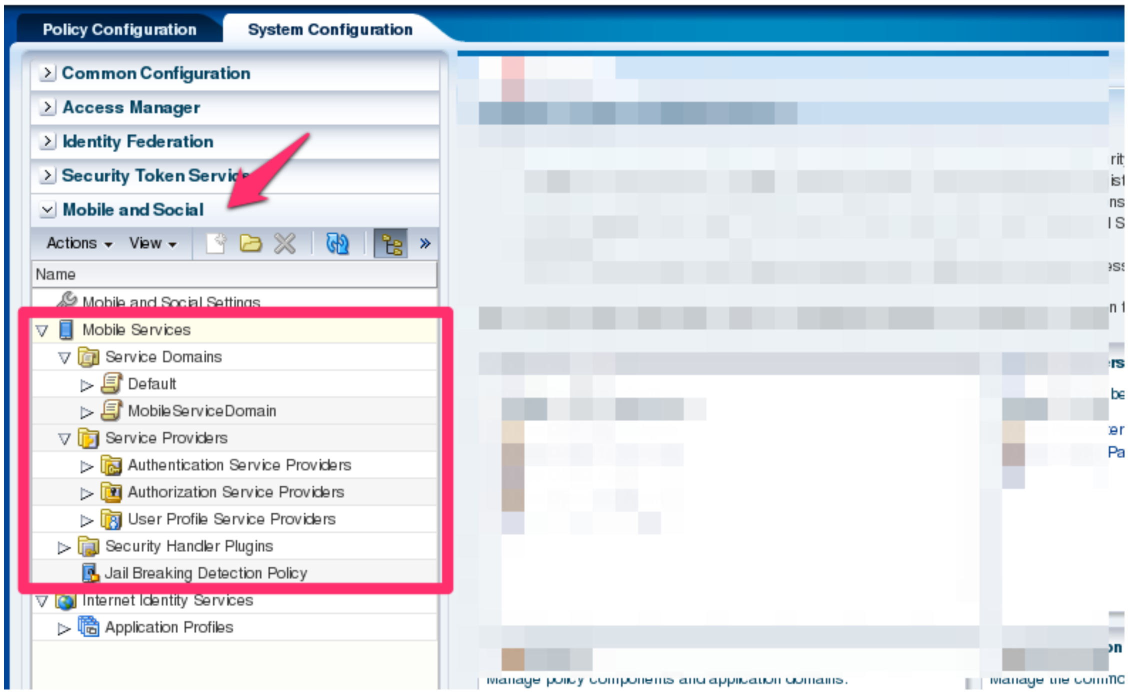


Available Services

The following is the list of services installed in your current deployment. Disabling a service will only turn off that service and will not uninstall it from the system.

Service Name	Status	
Access Manager Access Manager provides Single Sign-On, authentication and coarse grained authorization mechanisms for enterprise applications.	✓	Disable
Identity Federation Identity Federation provides support for SAML and related protocols to enable cross domain Single Sign-On.	⊘	Enable
Security Token Service Security Token Service provides a standards based, centralize[illegible]hanism of trust brokerage between infra[illegible]rs.	⊘	Enable
Mobile an[illegible] Mobile and Social provides lightweight mobile, cloud and social interfaces to Identity and Access via REST and OAuth/OpenID.	✓	Disable

Once enabled the Mobile and Social Services need to be configured. On the left hand navigation panel is a section for Mobile and Social:



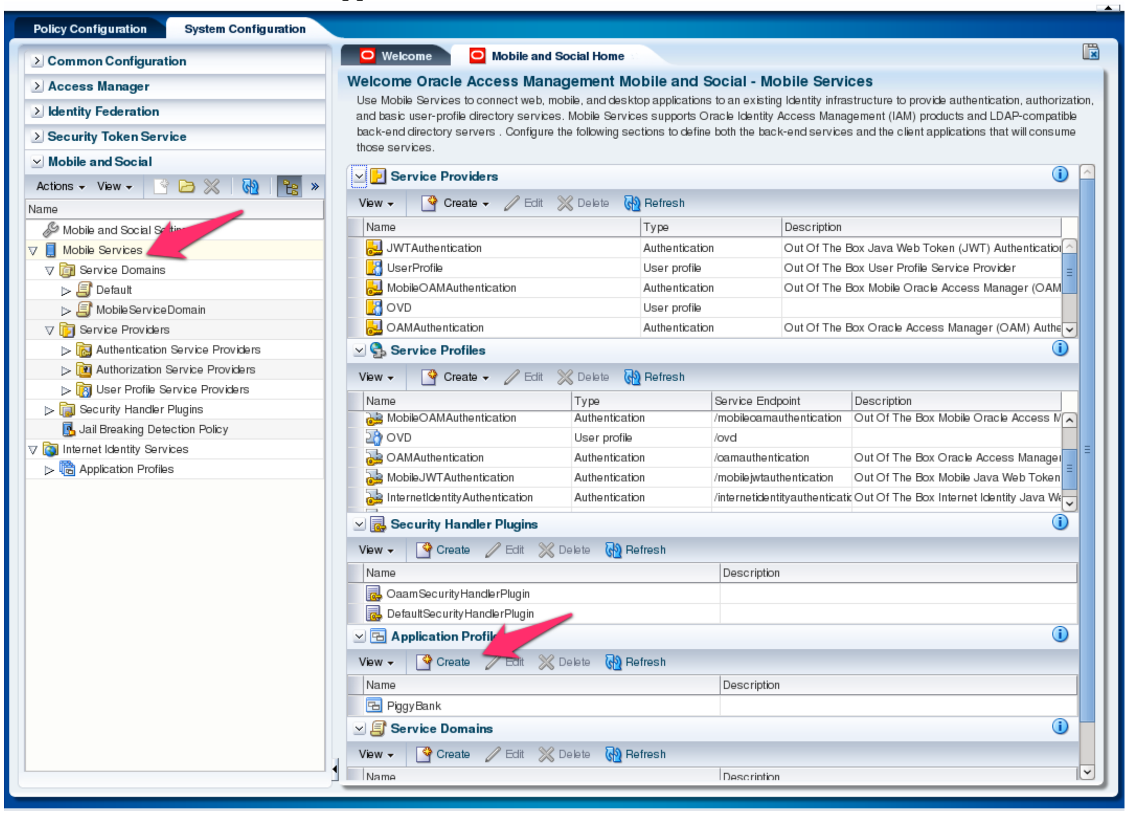
Most, but not all of the needed settings are located within the highlighted section of the administrative console screenshot above.

Creating the Mobile Application Profile

There are a number of steps required to configure the interaction between the mobile application and the Mobile and Social Services.

The first of these steps is to tell Mobile and Social about the mobile application. To do that we need to create a Mobile Application Profile; this does two things: first it tells Mobile and Social about the application and its configuration, and second it defines the configuration information that will be sent to the Mobile and Social SDK embedded in the application.

To create the Application Profile click on Mobile Services on the left hand navigation panel and then click the Create button under Application Profiles:



Welcome Oracle Access Management Mobile and Social - Mobile Services

Use Mobile Services to connect web, mobile, and desktop applications to an existing Identity infrastructure to provide authentication, authorization, and basic user-profile directory services. Mobile Services supports Oracle Identity Access Management (IAM) products and LDAP-compatible back-end directory servers . Configure the following sections to define both the back-end services and the client applications that will consume those services.

Service Providers

Name	Type	Description
JWTAuthentication	Authentication	Out Of The Box Java Web Token (JWT) Authenticatio
UserProfile	User profile	Out Of The Box User Profile Service Provider
MobileOAMAuthentication	Authentication	Out Of The Box Mobile Oracle Access Manager (OAM
OVD	User profile	
OAMAuthentication	Authentication	Out Of The Box Oracle Access Manager (OAM) Authe

Service Profiles

Name	Type	Service Endpoint	Description
MobileOAMAuthentication	Authentication	/mobileoamauthentication	Out Of The Box Mobile Oracle Access M
OVD	User profile	/ovd	
OAMAuthentication	Authentication	/oamauthentication	Out Of The Box Oracle Access Manage
MobileJWTAuthentication	Authentication	/mobilejwtauthentication	Out Of The Box Mobile Java Web Token
InternetIdentityAuthentication	Authentication	/internetidentityauthenticati	Out Of The Box Internet Identity Java We

Security Handler Plugins

Name	Description
OaamSecurityHandlerPlugin	
DefaultSecurityHandlerPlugin	

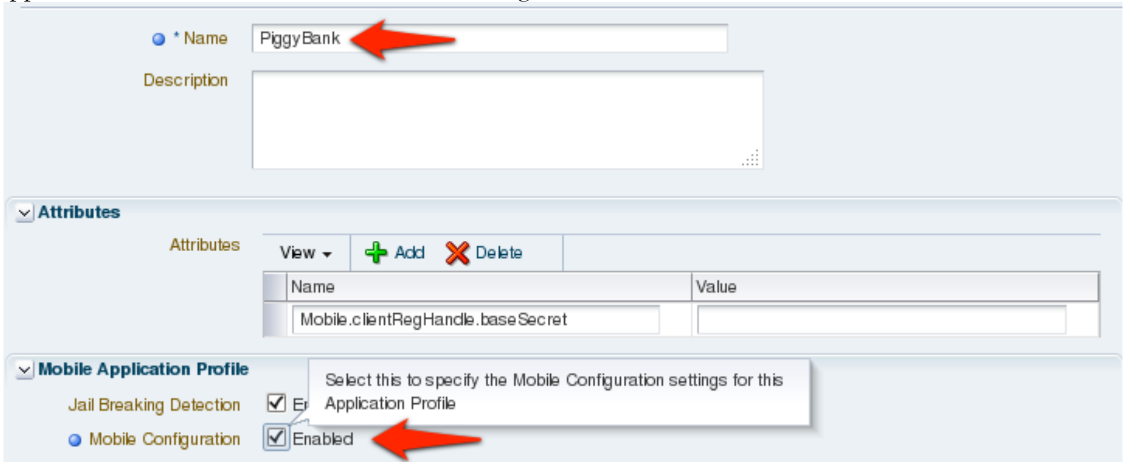
Application Profil

Name	Description
PiggyBank	

Service Domains

Name	Description

A new tab will appear with the new Application Profile. Enter a name and, since this is a mobile application, check the box next to Mobile Configuration:



* Name	PiggyBank
Description	

Attributes

Name	Value
Mobile.clientRegHandle.baseSecret	

Mobile Application Profile

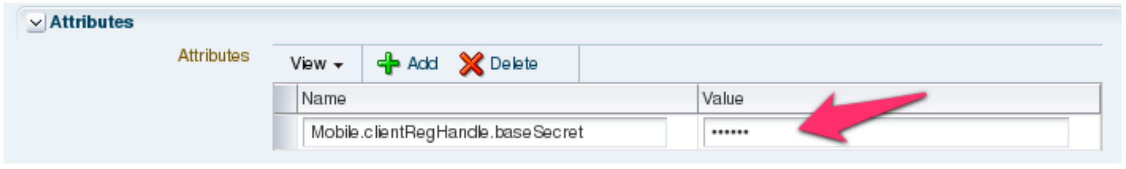
Jail Breaking Detection	☑ En
Mobile Configuration	☑ Enabled

Select this to specify the Mobile Configuration settings for this Application Profile

Please note: it is best to check the Mobile Configuration checkbox at this point (i.e. before clicking the Create button to save the Application Profile). Clicking this checkbox on a newly created Application Profile behaves slightly differently than checking it on an existing Application Profile for various reasons. For now it is best to simply check that box before proceeding.

After setting the Application Profile as a Mobile Configuration a number of other options will automatically appear and default settings will be populated. For now the only settings we need to adjust are (1) the "Attribute" named "baseSecret" and the (2) "Apple iOS Bundle ID".

The baseSecret is near the top of the configuration pane:

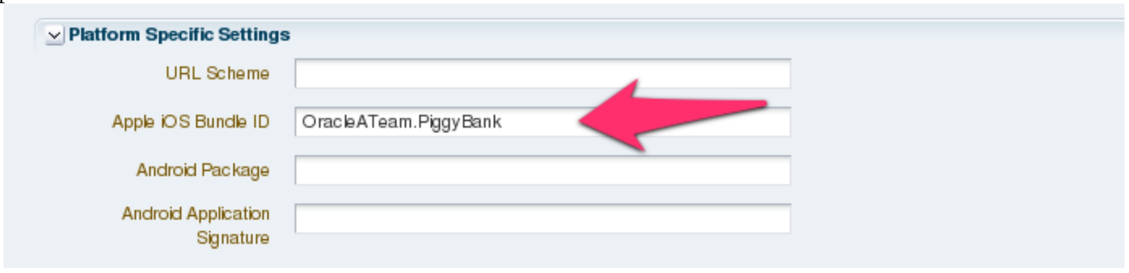


Name	Value
Mobile.clientRegHandle.baseSecret	••••••

You can set this value to any value you like, but you should take note as it will be used when creating the mobile application. The M&S documentation describes this setting as:

- `Mobile.clientRegHandle.baseSecret` is a mandatory attribute used by the server as a private secret to sign each Client Registration Handle for this application.

The iOS Bundle is located within the Platform Specific Settings near the bottom of the configuration pane:



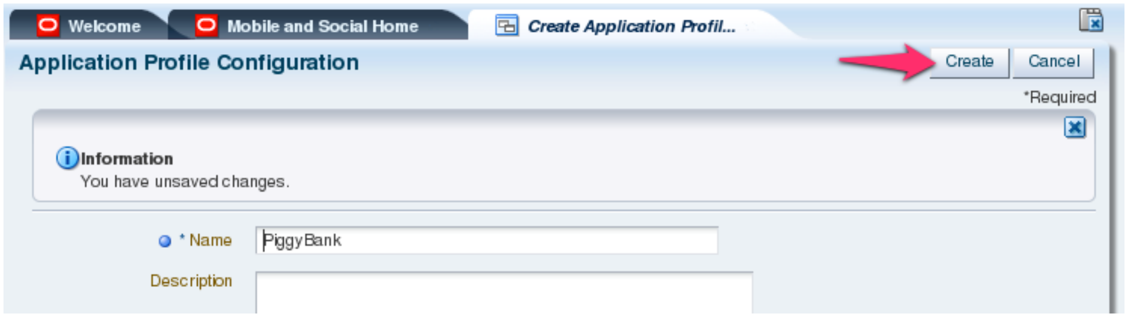
URL Scheme	
Apple iOS Bundle ID	OracleATeam.PiggyBank
Android Package	
Android Application Signature	

The iOS Bundle needs to match the "Bundle Identifier" within the XCode project, which is detailed in a later section. As with the baseSecret the value you enter here will be needed in a later step so it is important to take note of the value you select.

- Apple iOS Bundle ID - Type the unique Bundle ID that is configured in the mobile client application. Each iOS mobile application has a unique Bundle ID.

If you are unfamiliar with XCode and Application Bundles for now simply enter your company name, a dot, and a short name for your application. As shown in the image above we used "OracleATeam.PiggyBank".

Finally, click the Create button at the top of the pane to save the new Application Profile:



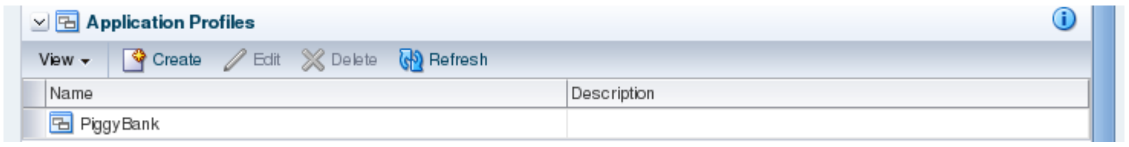
Application Profile Configuration

*Required

Information
You have unsaved changes.

* Name: PiggyBank
Description:

The Application Profile should now appear within the Application Profiles section of the Mobile and Social configuration:



Name	Description
PiggyBank	

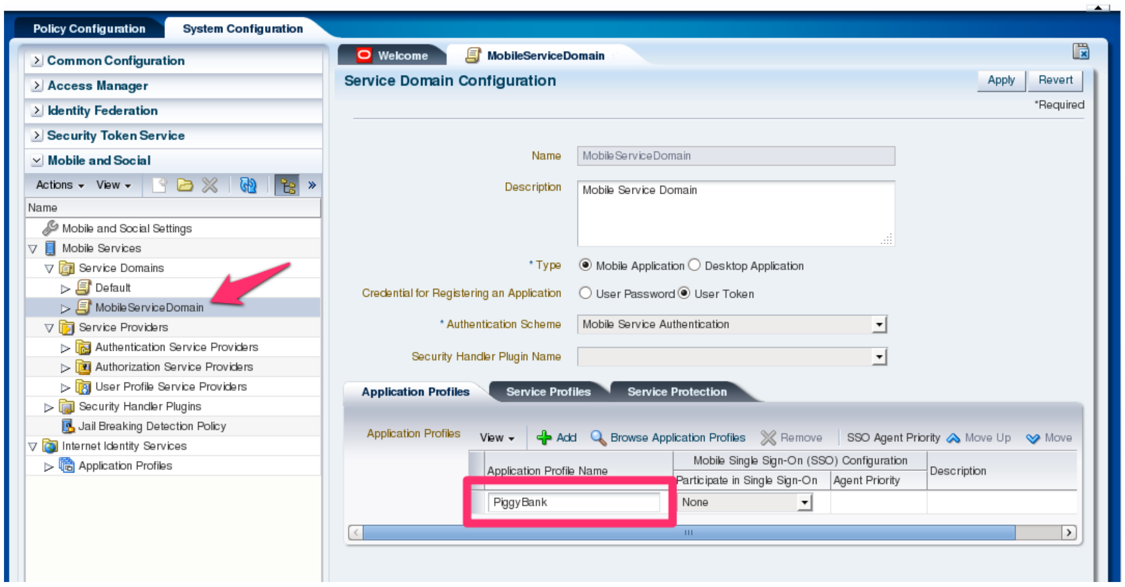
The Application Profile does two things – it stores the configuration information that the Mobile and Social SDK embedded in the mobile App will need to interact with the Mobile and Social Services and it describes the mobile App to Mobile and Social so that M&S can provide that information to the SDK.

Add the Application Profile to a Service Domain

A Service Domain simply groups one or more mobile applications, described by Application Profiles we created above, and applies a common configuration to those Application Profiles.

When you install Mobile and Social two Service Domains are automatically created. Those Service Domains are intended as “defaults” and contain settings that are appropriate for the most common types of applications customers create. The existing Service Domain named “MobileServiceDomain” contains a configuration that is appropriate for this mobile banking App.

Open the MobileServiceDomain and add the “PiggyBank” Application Profile as shown below:



Service Domain Configuration

*Required

Name	MobileServiceDomain
Description	Mobile Service Domain
* Type	[✓ Mobile Application] Desktop Application
Credential for Registering an Application	User Password [✓ User Token]
* Authentication Scheme	Mobile Service Authentication
Security Handler Plugin Name	

Application Profiles

Application Profile Name	Mobile Single Sign-On (SSO) Configuration		Description
	Participate in Single Sign-On	Agent Priority	
PiggyBank	None		

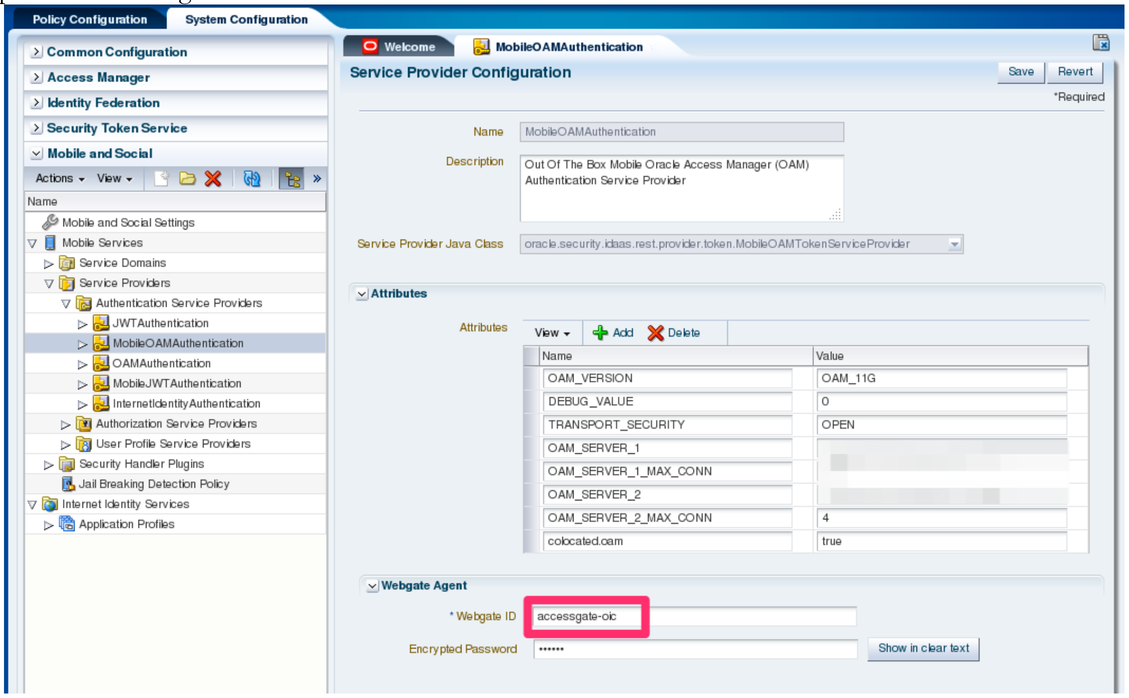
The remaining settings are likely appropriate for most first Mobile and Social-enabled applications.

OAM Configuration

As described above when the M&S SDK sends a username and password to the M&S Services, the M&S Services in turn send those credentials to the OAM Server to be validated. Mobile and Social sends those credentials to the core OAM server via an OAP connection. As with any OAP connection, the client (in this case M&S) needs to have an agent identity and a protected resource. When you installed OAM an agent identity and the necessary resource was automatically created and preconfigured. You should not need to make any changes to this configuration but it is useful to understand what is going on inside should you need to troubleshoot.

The Application Profile described above uses the “MobileOAMAuthentication” Authentication Service Provider. An Authentication Service Provider is simply a means for OAM to verify user credentials. As discussed above this Authentication Service Provider uses OAP to talk to OAM, which in turn checks the user’s username and password against the configured LDAP directory.

That provider contains the configuration information needed by M&S to communicate with the core OAM server. If you examine that Authentication Service Provider you will see the Webgate ID is preset as “accessgate-oic”:



Service Provider Configuration

Name	MobileOAMAuthentication
Description	Out Of The Box Mobile Oracle Access Manager (OAM) Authentication Service Provider
Service Provider Java Class	oracle.security.idaas.rest.provider.token.MobileOAMTokenServiceProvider

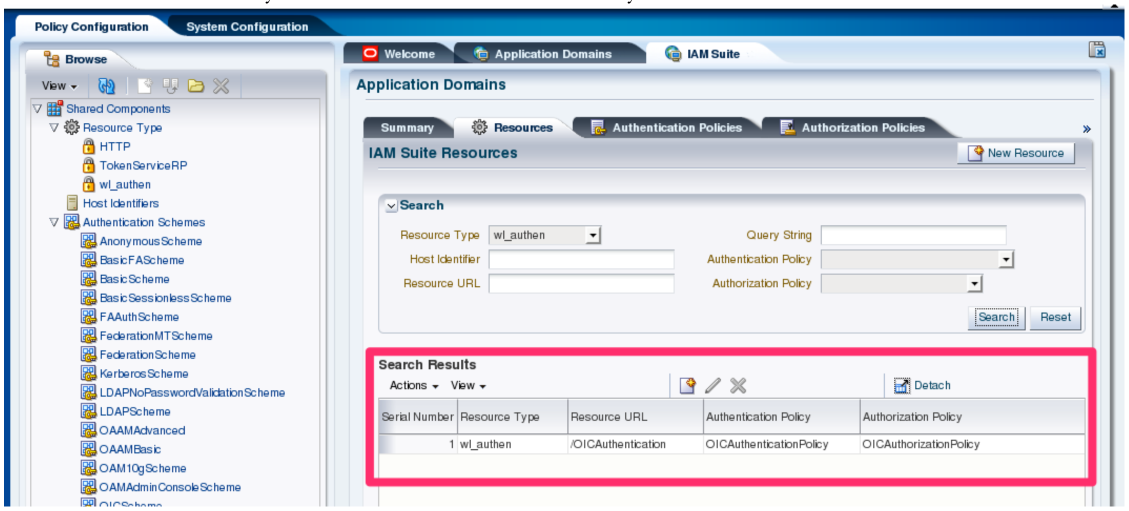
Name	Value
OAM_VERSION	OAM_11G
DEBUG_VALUE	0
TRANSPORT_SECURITY	OPEN
OAM_SERVER_1	[redacted]
OAM_SERVER_1_MAX_CONN	[redacted]
OAM_SERVER_2	[redacted]
OAM_SERVER_2_MAX_CONN	4
colocated.oam	true

* Webgate ID	accessgate-oic
Encrypted Password	••••••

You should not need to change this value.

Mobile and Social connects to the OAM server with this Webgate ID and sends the username and password in an authentication request for the resource /OICAuthentication.

If you open the IAM Suite domain and search for that resource you will see that it uses the OICAuthenticationPolicy and the OICAuthorizationPolicy:



Application Domains

IAM Suite Resources

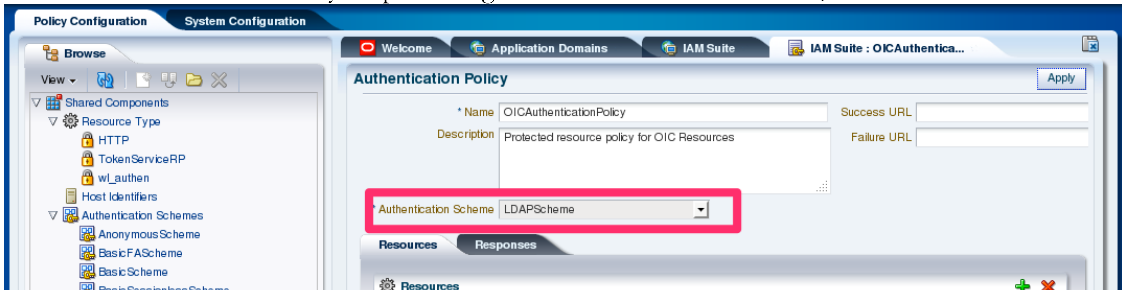
Search

Resource Type	wl_authen	Query String	
Host Identifier		Authentication Policy	
Resource URL		Authorization Policy	

Search Results

Serial Number	Resource Type	Resource URL	Authentication Policy	Authorization Policy
1	wl_authen	/OICAuthentication	OICAuthenticationPolicy	OICAuthorizationPolicy

The “OICAuthenticationPolicy” is pre-configured to use the LDAPScheme, as shown below:



Authentication Policy

* Name: OICAuthenticationPolicy
Description: Protected resource policy for OIC Resources
Success URL:
Failure URL:
* Authentication Scheme: LDAPScheme

The Authentication Scheme is primarily used as any Authentication Scheme would be for any other resource – it defines the sort of credentials that are required and how the OAM server validates those credentials.

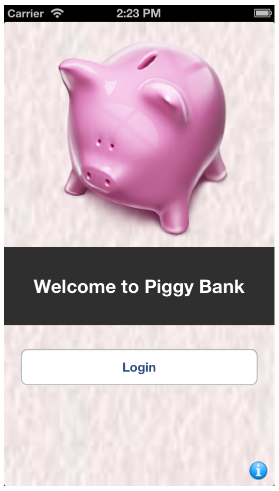
What all this means is that once you configure the LDAPScheme for regular web resources protected by conventional WebGates you should not need to do anything more; Mobile and Social will use that same configuration to authenticate users. It also means that if you want to configure Mobile and Social to authenticate a different user population you can simply adjust the Authentication Scheme in the OICAuthenticationPolicy.

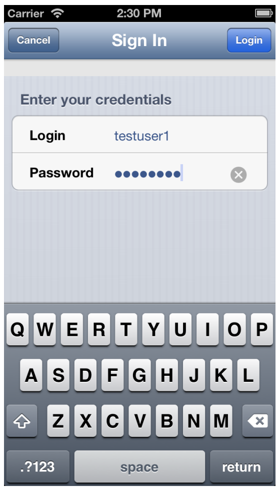
For the purposes of this paper we will assume that OAM has been properly configured to authenticate users.

The Piggy Bank “App”

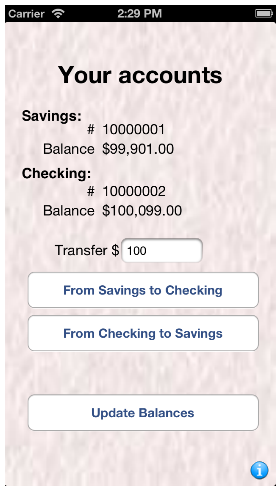
The OAM Mobile and Social SDK is distributed as a set of native libraries for each of the supported platforms. The Piggy Bank application we created was written for iOS but could just as easily be written for any other mobile platform supported by the SDK.

The Piggy Bank Application is a simple “three screen” application. The first screen has a single button – “Login” which, when tapped causes the second screen to appear. The second screen is the login screen on which a user enters their username and password. Once authenticated the user is shown the third screen, which displays the user’s bank accounts and allows them to transfer money between accounts:





Login testuser1
Password ••••••••



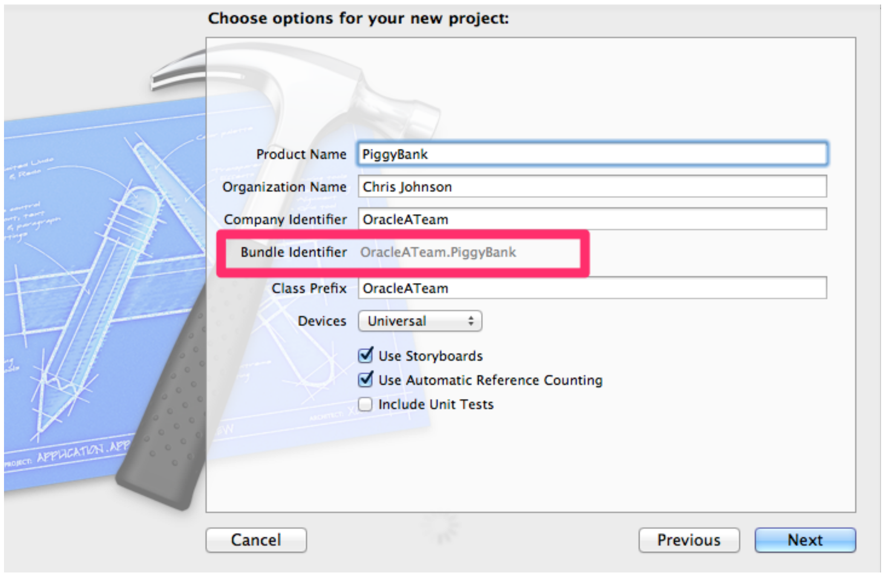
Savings:
# 10000001
Balance $99,901.00
Checking:
# 10000002
Balance $100,099.00
Transfer $ 100

The first screen is part of the core application logic and was created with XCode's Interface Builder. The second screen is the default login screen provided with the Mobile and Social SDK; the screen shots above show the default Login screen but it is trivial to replace the login screen with another of your own design (information on how to customize this screen is included later in this paper). The third screen, like the first, is part of the core application and was created with Interface Builder.

Creating the Application

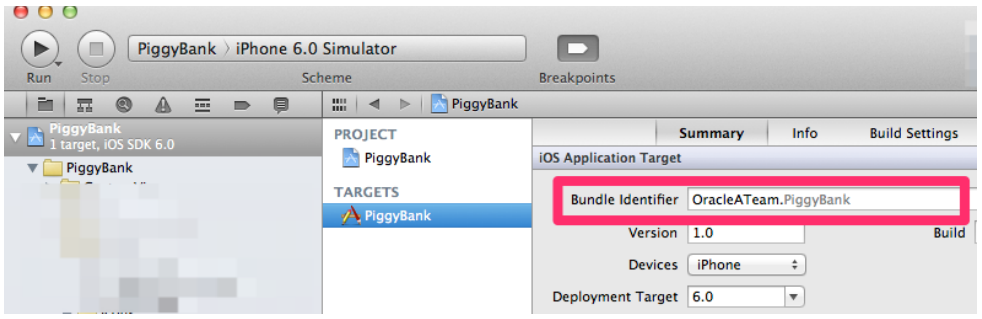
When creating the application in XCode, Apple's development environment for iOS, the first important step is to ensure that the Bundle Identifier matches the value previously entered when creating the Mobile Application Profile in Mobile and Social. In the case of PiggyBank that string is "OracleATeam.PiggyBank".

XCode creates the bundle identifier by concatenating two strings – the Company Identifier and the Product Name, as shown below:



Product Name: PiggyBank
Organization Name: Chris Johnson
Company Identifier: OracleATeam
Bundle Identifier: OracleATeam.PiggyBank
Class Prefix: OracleATeam
Devices: Universal
[✓] Use Storyboards
[✓] Use Automatic Reference Counting
[ ] Include Unit Tests

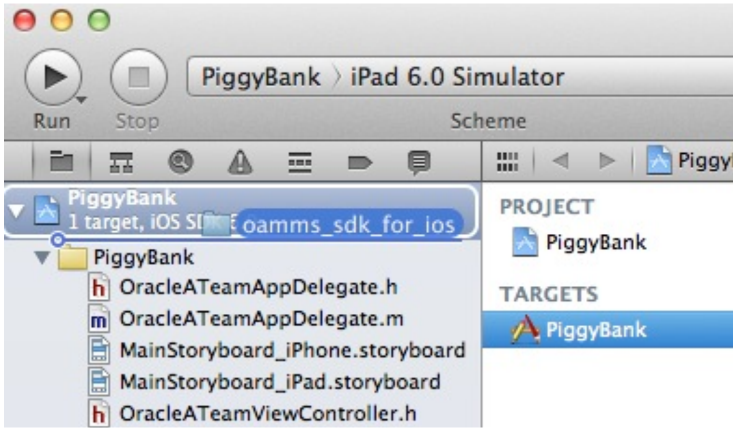
Once you have created the application you can change the Bundle Identifier if needed, but not the Application Name:



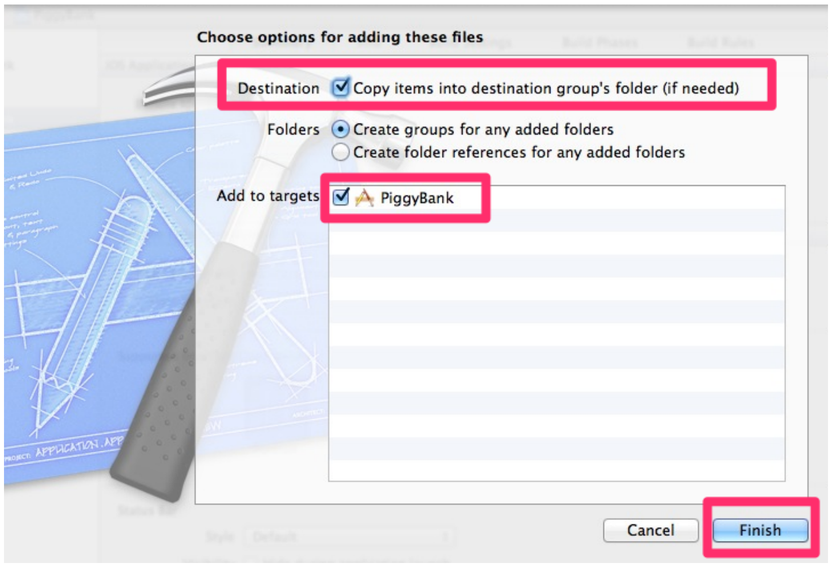
Bundle Identifier: OracleATeam.PiggyBank
Version: 1.0
Build
Devices: iPhone
Deployment Target: 6.0

Adding the Mobile and Social SDK

To add the Mobile and Social SDK to the Project simply drag the folder from Finder to the Project in Xcode:



In the presented dialog, check the “Copy items…” checkbox and the checkbox next to the PiggyBank target. Then click Finish.


Choose options for adding these files

Destination ✓ Copy items into destination group's folder (if needed)

Folders ● Create groups for any added folders
○ Create folder references for any added folders

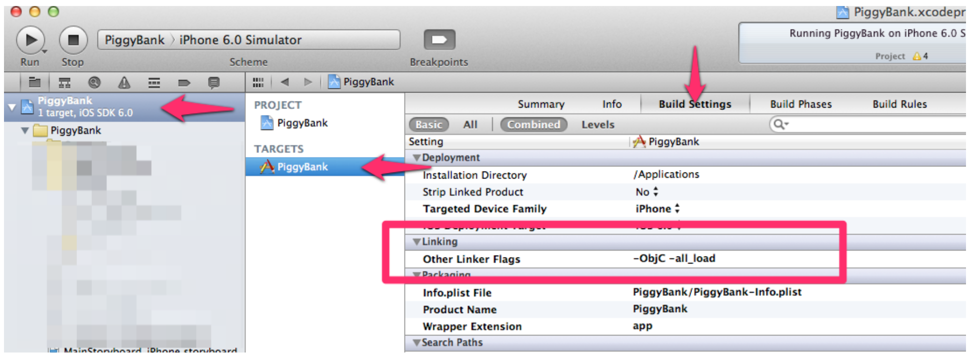
Add to targets ✓ PiggyBank

The first checkbox causes Xcode to copy the files into the project. The second checkbox causes Xcode to add those files into the build so that when you compile the project they are included in the App executable.

The final, and most often forgotten, step is to add two linker flags under the Build Settings. The SDK documentation includes this text block:

> **Important:**
> Before linking your project, add as a single line both the -ObjC and -all_load linker flags to your project. Without these flags your application will crash with a "selector not recognized" runtime exception.
> Because libIDMMobileSDK extends pre-existing classes with categories, the linker does not know how to associate the object code of the core class implementation with the category implementation. This prevents objects created in the resulting application from responding to a selector that is defined in the category.
> For background information and steps that describe how to add flags to your project, see the following page:
> http://developer.apple.com/library/mac/#qa/qa1490/

To add those flags first click on the Project, then click the Target, then click the "Build Settings" tab and find the "Other Linker Flags" row. Edit it and add -ObjC -all_load to any existing values. In simple or new applications there may not be any settings yet there. The screen capture below shows the settings for PiggyBank after they have been properly updated:



Setting	PiggyBank
▼Deployment	
Installation Directory	/Applications
Strip Linked Product	No
Targeted Device Family	iPhone
▼Linking	
Other Linker Flags	-ObjC -all_load
▼Packaging	
Info.plist File	PiggyBank/PiggyBank-Info.plist
Product Name	PiggyBank
Wrapper Extension	app
▼Search Paths	

Using the Mobile and Social SDK

The Mobile and Social SDK is quite easy to use. The PiggyBank app uses three main services of the SDK – one to initialize the SDK, one to "login" to OAM, and one to "logout" from OAM. Each of the calls takes a number of parameters and performs its work asynchronously. Once the operation is complete the SDK calls back to your application to notify it that the action has completed. For the PiggyBank application I placed all of my code that interacts with the M&S SDK in a single class to make it easier to see and understand, but you are free to include the logic wherever you like.

It should be noted that the Mobile and Social SDK includes a number of other services that are not needed for this very basic application. A complete list of services is available in the documentation.

Initialization

To initialize the SDK you simply 'alloc' an instance of the OMMobileSecurityService as shown below and call initWithURL:

```
NSURL * url = @"http://login.piggybank.oracleateam.com:8000";
NSString * appName = @"PiggyBank";
NSString * oicDomain = @"MobileServiceDomain";

OMMobileSecurityService *mobileServices = [[OMMobileSecurityService alloc]
                                             initWithURL:url
                                             appName:appName
                                             domain:oicDomain
                                             delegate:self];
```

The delegate is the class that the M&S SDK will call back to when the SDK has been initialized or if the initialization fails. The delegate should have a didReceiveApplicationProfile method as shown below:

```
- (void)didReceiveApplicationProfile:(NSDictionary *)applicationProfile
                               error:(NSError *)error
{
  NSLog(@"didReceiveApplicationProfile");
}
```

If the SDK was able to reach the server and retrieve the profile, the profile information will be returned in applicationProfile. Application developers using the SDK are not required or expected to do anything with the profile information; it is available for advanced use cases. The "error" will be populated only in cases where the server can't be reached or there was some problem retrieving the profile.

"Login"

Once the Mobile and Social SDK has been initialized the App is free to call the SDK to initiate the login process. To do that the App calls the startAuthenticationProcess method:

```
NSError *err = nil;
err = [self.mss startAuthenticationProcess:nil
                   presenterViewController:self];
```

The startAuthenticationProcess initiates a login process that continues asynchronously. The "err" return code returned is used only to indicate a fatal error starting that process and should not be encountered in normal cases. The presenterViewController is the delegate notified of either a success or failure of the authentication.

When the method is called the login screen (prompting for username and password) will appear. After the user submits their credentials the SDK will send them to the M&S Server and wait for a response. When the response is received the SDK will process it and then notify the App by calling the delegate passed in as presenterViewController.

The SDK notifies the delegate by calling this method:

```
- (void)didFinishAuthentication: (OMAuthenticationContext *)context
                          error: (NSError *)error
{
  NSLog(@"didFinishAuthentication");
}
```

Once the user is "logged in" the iOS app should be able to access OAM-protected resources.

Making REST calls

While simply verifying a user's username and password is interesting and potentially useful, most mobile Applications do much more than that. Mobile Apps usually make calls to servers on the Internet or Intranet to retrieve information or execute transactions. And the most common way Apps interact with services is to make REST calls over HTTP.

In iOS Apps developers create an HTTP request (an NSURLRequest object) and then call the sendAsynchronousRequest method of NSURLConnection. The code would look *something* like this:

```
NSURLRequest*request =[NSURLRequest requestWithURL:url];
NSOperationQueue*queue =[[NSOperationQueue alloc] init];

[NSURLConnection sendAsynchronousRequest:request
                                   queue:queue
 completionHandler:^(NSURLResponse*response,NSData*data,NSError*error)
     {
     ...
     }
     ];
```

The caller is then responsible for parsing the resulting return data in the completion handler.

The OAM Mobile and Social SDK offers a slightly simpler interface, which automatically manages inserting the correct OAM session identification information in HTTP request. This interface is a method in OMRestRequest called executeRESTRequestAsynchronously and is called as shown below:

```
NSMutableURLRequest * request = [NSMutableURLRequest
    requestWithURL:url]];

OMRESTRequest * omr = [OMRESTRequest alloc];
[omr initWithMobileSecurityService:mss
                          delegate:self];

[omr executeRESTRequestAsynchronously: request
                    convertDataToJSON:FALSE];
```

Note: There is a synchronous version of the same method (called executeRESTRequest) but the Asynchronous method is recommended by Oracle and preferred by most developers.

When the method is called the SDK returns to the application immediately, the REST request is sent asynchronously, and once the result is received the delegate is notified. The SDK notifies the delegate that the request is complete by calling the didFinishExecutingRESTRequest method. The method signature is shown below along with an example of how to process the resulting data:

```
- (void) didFinishExecutingRESTRequest:(OMRESTRequest *) RESTRequest
                                  data:(id) data
                           urlResponse:(NSURLResponse *) urlResponse
                                 error:(NSError *) error
                           asyncHandle:(OMAsyncOpHandle *) handle
{
    if (error != nil)
    {
        NSLog(@"Error: %@", [error localizedDescription]);
        NSLog(@"Error: %d - %@", error.code, error.description );
        NSLog(@"Response data: %@", data );
        [self.restNotificationDelegate notifyRESTError];
    }
    else
    {
        // Convert the response data to a string.
        NSString *responseString = [[NSString alloc] initWithData:data
encoding:NSUTF8StringEncoding];
        // View the data returned - should be ready for parsing.
        NSLog(@"HTTP Response string: %@", responseString);
}
```

In the sample method implementation above the HTTP response body is simply logged via NSLog.

In the example above convertDataToJSON is set to FALSE – which matches the behavior of the previous example using NSURLConnection. Since most REST interfaces used by mobile apps use JSON for their requests and responses the Mobile and Social SDK also simplifies application code by including an option to parse the returned string as a JSON representation and turn it into an NSDictionary*. In that case "data" would contain an NSDictionary *.

Creating the REST Services

The REST services used by the PiggyBank "App" are quite simple. As discussed above only two services were created – one returns a list of accounts and their balances, the other allows the user to transfer money between accounts. The REST services could be written in any language and can run in any web server, but for the purposes of this PoC they were run inside a WebLogic container as a very simple J2EE web application. The google-gson library was used to transform from HTTP requests to Java classes and from Java classes to JSON format for the response.

REST Call Examples

The "get balances" call is the simpler of the two requests but the more complicated response. A nearly complete HTTP request/response pair is shown below:

```
GET /PiggyBankServices/balances HTTP/1.1
Host: services.piggybank.oracleateam.com:8000
Accept-Encoding: gzip, deflate
Accept: application/json
...


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

{
"Accounts":[
  {"accountNumber":"10000001","accountType":"SAVINGS","accountBalance":100000.0},
  {"accountNumber":"10000002","accountType":"CHECKING","accountBalance":100000.0}
]
}
```

In the interest of space some extraneous HTTP headers were removed from the request and response.

The HTTP request is simply an HTTP GET to /PiggyBankServices/balances. The response contains a JSON data representation of an array of accounts, each of which has an account number, type and balance. As you can see in the above example the user has two accounts – a savings account and a checking account, each with a unique "accountNumber", and each with an "accountBalance" of 100000.0.

The "transfer" service is a slightly more complicated request since JSON data is sent. A sample is shown below:

```
POST /PiggyBankServices/transfer HTTP/1.1
Host: services.piggybank.oracleateam.com:8000
Content-Type: application/json
Accept: application/json
...

{"fromAccount":"10000001","toAccount":"10000002","amount":"100"}

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

{"transferResult":"true"}
```

In this case the HTTP request is a POST to the transfer service's URL. The JSON data in the request contains three items – the "from" and "to" accounts and an amount. The service processes the request and returns back either "true" or "false" to the client in the HTTP response.

Implementing the REST Services

The REST services were implemented as two very simple Servlets – one for "get balances" and one for "transfer". As described above the Servlets are running inside a WebLogic server instance and are protected by a WebGate. When the mobile App makes an HTTP request to the Servlet the request passes through a web server with an OAM WebGate. The WebGate and the M&S SDK negotiate passing the user identity securely and then the HTTP request is passed along to WebLogic. The OAM Identity Asserter inside WebLogic then establishes the JAAS Subject and Principals based on the user identity included in the HTTP request by the WebGate. The Servlet can then easily retrieve the username by simply calling getRemoteUser or by examining the JAAS Subject. Both Servlets rely on the google-gson library to parse and generate JSON formatted data.

The "Get Balances" Servlet

The core logic of the "get balances" Servlet code is as quite simple – the Servlet retrieves the username from WebLogic, gets the list of accounts for the user, and then converts that to JSON with a single call to google-gson. The (slightly simplified) code is shown below:

```
public class Balances extends HttpServlet {
...

    public void service(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException
    {
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();

        String userName = request.getRemoteUser();

        BankUser bankUser = BankUsers.getUser(userName);
        List accounts = bankUser.getAccounts();

        Map<String,Object> responseData = new HashMap<String,Object>();
        responseData.put("Accounts", accounts);

        Gson gson = new Gson();
        String json = gson.toJson(responseData);

        out.println( json );
        out.close();
    }
}
```

Note: The above code is fully functional; the only changes from the actual code were the removal of error detection and sanity checks.

The “Transfer” Servlet

Similarly the “transfer” service code is quite simple – it parses the incoming request from JSON into a Java class and then processes the transfer request. That code is shown below:

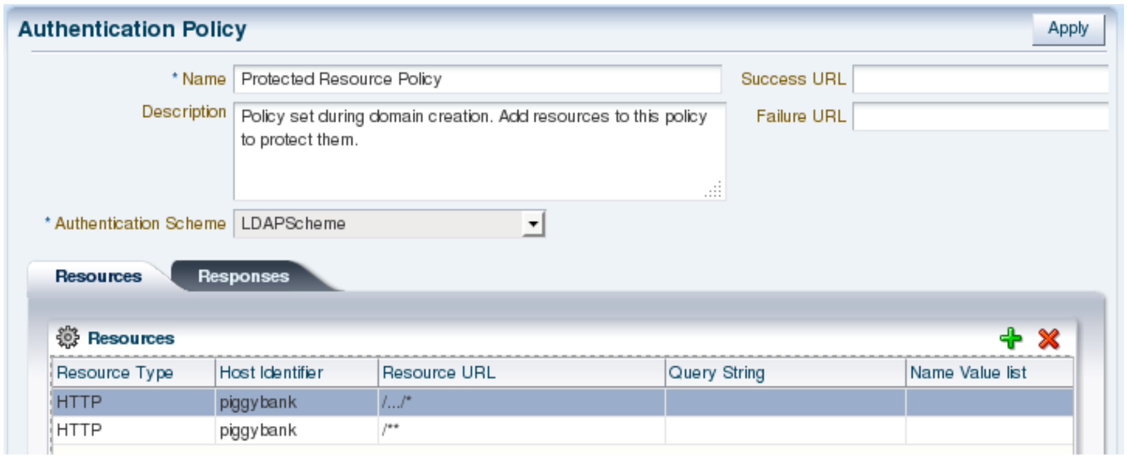
```
public class Transfer extends HttpServlet {
...

    public void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException
    {
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();

        String result = "false";
        String reason = null;

        try {
            Reader reader = new InputStreamReader(request.getInputStream(),
                                                  "UTF-8");
            JsonReader jsonReader = new JsonReader(reader);

            TransferRequest transfer = gson.fromJson(jsonReader,
                                                     TransferRequest.class);

            // sanity checks removed
            ...

            // actual transfer logic
            ...

            result = "true";
        }
        catch ( AccountNotFoundException anfe ) {
            reason = anfe.getMessage();
        }
        catch ( InsufficientBalanceException ibe ) {
            reason = ibe.getMessage();
        }

        Map<String,Object> responseData = new HashMap<String,Object>();
        responseData.put("transferResult", result);

        if ( null != reason ) {
            responseData.put( "Reason", reason );
        }

        Gson gson = new Gson();
        String json = gson.toJson(responseData);

        out.println( json );
        out.close();
    }
}
```

In the above code if the transfer succeeds the transferResult is set to true. If a failure occurs an exception will be thrown and caught, in which case the transferResult will be false **and** a Reason will be added to the response indicating the cause of the failure. In either case the result is transformed to JSON by calling gson.toJson() and then sent as text in the HTTP Response by out.println().

Protecting the REST Services with OAM

Once the REST services have been deployed they need to be protected with OAM both for actual security from rogue attackers and so that the user identity is populated. The REST services are simply Servlets and so protecting them with OAM is the same as any other web resource protected with a WebGate – simply create the resources and choose an appropriate Authentication Policy. The default policies created when you register a new WebGate via "oamreg" are perfectly adequate.

This PoC environment used the /** and /…/* wildcard resources and the LDAPScheme as shown below:



Authentication Policy

* Name	Protected Resource Policy
Description	Policy set during domain creation. Add resources to this policy to protect them.
* Authentication Scheme	LDAPScheme
Success URL	
Failure URL	

Resource Type	Host Identifier	Resource URL	Query String	Name Value list
HTTP	piggybank	/…/*		
HTTP	piggybank	/**		

With these policies the REST services are protected by Oracle Access Manager and the user identity acquired by the App is automatically propagated to the bank services.

Extending the PoC Use Cases

The basic system described above is a complete, if minimal, implementation needed to demonstrate the capabilities of the Mobile and Social Services and the associated client-side SDK. The basic system created for this PoC and white paper can be extended in a number of interesting ways using the services offered by the Oracle IAM platform.

Customizing the Login Screen

The default login screen provided by the Mobile and Social SDK is intentionally generic and the Mobile and Social SDK was built with the assumption that customers would want to customize the login screen. Customization is not limited to simply changing words or images - customers have complete control of the login screen. You can include your own images, wording, branding, and have complete control of the layout; in other words you can do anything you can do with any other iOS UIView.

Customizing the Login screen requires two steps – writing the Objective C code to implement the Login screen and then adjusting the call to "startAuthenticationProcess" to pass an instance of the new screen to the SDK.

Implementing the Objective C Code

Complete documentation of how to create and use a custom login screen, and an example of how to do so are included in the product documentation. At a high level the process begins with the crteation of a new class implementing the OMAuthView interface (defined in OMAuthView.h). The OMAuthView interface is in turn a UITableView with a few extra items. When you implement your custom OMAuthView you have complete control – add whatever controls, text, images and other UI components you need. The only absolute requirement is that your OMAuthView include a method that returns the username and password in an NSDictionary. For example:

```
- (NSDictionary *)retrieveAuthData
{
    [self.authData setValue:self.usernameField.text forKey:OM_USERNAME];
    [self.authData setValue:self.passwordField.text forKey:OM_PASSWORD];
    return self.authData;
}
```

If you decide to use your own Login and Cancel buttons rather than those automatically provided by the OMAuthView you will need to implement a number of other methods. The documentation and OMAuthView.h file discusses this in detail.

Using the Custom View for Login

Once the OMAuthView code has been written, the call to startAuthenticationProcess needs to be adjusted to use the new view, by using the authView field with the authnRequest, which is passed to startAuthenticationProcess. In the sample code above we passed nil for this parameter. To use the new custom OMAuthView simply allocate your custom OMAuthView and set that property to an instance of that view; the Mobile and Social SDK will then use your custom view instead of the default one. The code below shows how to do this:

```
OMAuthenticationRequest * authnRequest = nil;
authnRequest = [[OMAuthenticationRequest alloc] init];

PiggyBankAuthView *myLoginView = [[PiggyBankAuthView alloc] init];
authnRequest.authView = myLoginView;

[self.mss startAuthenticationProcess:authnRequest
            presenterViewController:presenter];
```

Device Fingerprinting and Jailbreak Detection

The Mobile and Social Server includes the ability to acquire a "device fingerprint" and/or detect whether the device has been "Jailbroken" (for iOS devices) or "rooted" (for Android devices). With this feature enabled, device information including the GPS coordinates, WLAN MAC address and others are collected by the SDK and passed to the Mobile and Social Server at login time. This information is then passed to OAAM and can be used to deny authentication or for a number of other purposes (see below). One example use of this feature might be for an employer to prohibit their employees from using the employer's custom Apps on a jailbroken iPhone for security reasons.

Strong Authentication

In addition to simply denying access as discussed above the integration between Mobile and Social and Oracle Adaptive Access Manager (OAAM) can be used to perform stronger authentication. With this feature enabled users can be challenged for a One Time Password (OTP) delivered by email or SMS or for answers to their Knowledge-based authentication (KBA) challenge questions stored in OAAM.

When enabled the user is prompted to login as normal and then, if the OAAM policies require further authentication, the user is shown another screen with the challenge from OAAM. As with the login screen this challenge screen is completely customizable; the authnRequest passed to startAuthenticationProcess has a kbaView field. Customers can create that custom view and then pass it into the method in the same was as the authnView. The "KBA" screen is used for both KBA and OTP challenges – in the latter case the challenge question is simply a fixed string asking for the one time password.

Introducing Transaction Risk Scoring

In an internet-facing environment, determining the risk associated with a particular transaction may be important. For example if a user were to use the PiggyBank App to transfer a large amount of money from one account to another while in a physical location they have never visited before, the bank might consider that action more risky. Generally banks would not want to block the transaction but risky transactions might warrant some other action to confirm the user's identity before actually executing the transaction.

Oracle Adaptive Access Manager (OAAM) is designed to use multiple bits of context information to help determine the risk level of a particular transaction, inform an application of the risk level, and advise the application to take some action to reduce the risk. In the example above OAAM might instruct the application to perform an OTP or KBA to confirm the user's identity before allowing the transfer.

If the PiggyBank application were a real electronic banking application, adding calls from the REST/JSON services to OAAM might be an important part of the bank's approach to mitigating risk both for the bank and for customers.

Using Oracle Enterprise Gateway

Oracle Enterprise Gateway (OEG) is a service gateway designed to be securely deployed in the DMZ as a way to safely expose business services to the Internet. OEG allows administrators to expose existing or new services without writing code or altering existing services.

In the PiggyBank PoC use case we could have replaced the Oracle HTTP Server plus WebGate and the custom REST services with OEG. Using OEG in place of OHS would have allowed us to expose REST services to the mobile apps and internally call already existing SOAP services on the bank's internal network. This change would also have allowed the gateway to easily call out to OAAM, OES or to other systems before calling those SOAP services, and could do so without additional code.

Conclusion

While the PiggyBank application is quite simple, it illustrates the power and capabilities of the Oracle Identity and Access Management platform including Oracle Access Manager, Oracle Adaptive Access Manager and some of the Mobile and Social Services. By using the OAM Mobile and Social SDK a fully functional mobile e-Banking application was created and secured in a very short time, without the need to install and configure any additional software and without the need to write complex code to secure the mobile App and its communication to the services it uses.

A customer with an existing security infrastructure based on Oracle Access Manager and Adaptive Access Manager can easily deploy Oracle Mobile and Social to extend the same security capabilities to mobile applications. By using the Mobile and Social SDK customers can seamlessly integrate security into their native Apps on popular mobile platforms including iOS and Android.

Oracle OAM Mobile and Social supports a number of other capabilities including authentication via Social sites, REST services for access to user profiles, Single Sign-On across mobile apps and more. For more information about the other features of the Mobile and Social product please refer to the Mobile and Social page on Oracle.com at http://www.oracle.com/technetwork/middleware/id-mgmt/overview/oamms-1696162.html

The Oracle Identity and Access management platform utilizes the High availability and scalability features of WebLogic server and the Identity and Access Management platform and thus any deployment of Mobile and Social needs no additional infrastructure to provide enterprise application requirements like scalability, reliability and high availability.

ORACLE®

Oracle is committed to developing practices and products that help protect the environment

Oracle Access Manager Mobile and Social, A Case Study – Piggy Bank
February 2013
Author: Christopher Johnson

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Hardware and Software, Engineered to Work Together