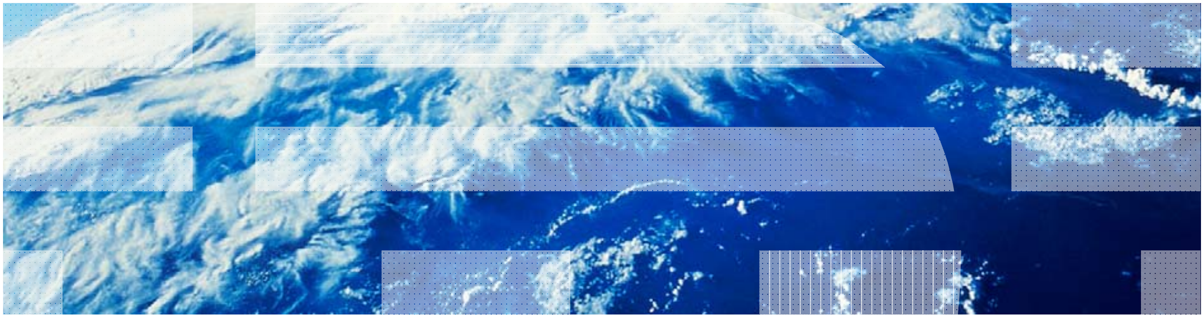
IBM Worklight V6.0.0
Getting Started

Integrating with SiteMinder



12 July 2013

Trademarks

About IBM®

See http://www.ibm.com/ibm/us/en/

Agenda

- Background
- Creating the challenge handler
- Creating the native page handler
- Creating the native page handler (Android)
- Creating the native page handler (iOS)
- Examining the result
- Troubleshooting
- Exercise

Background

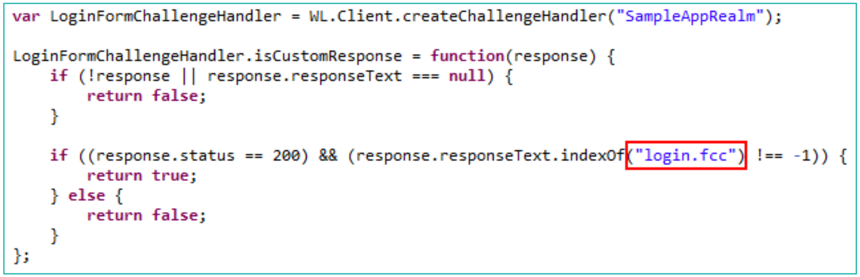
- As with other form-based authentication methods, SiteMinder requires a custom challenge handler to detect when authentication is required.
- A *native page handler* is required for this module because it displays the SiteMinder authentication form to the user directly.
- This native page handler is required to propagate SiteMinder cookies to your IBM® Worklight® application.
- Though this module targets SiteMinder specifically, the concepts that it describes can apply to other form-based authentication methods.

Agenda

- Background
- Creating the challenge handler
- Creating the native page handler
- Creating the native page handler (Android)
- Creating the native page handler (iOS)
- Examining the result
- Troubleshooting
- Exercise

Creating the challenge handler

- Start the implementation of the challenge handler with the following code, which is used to detect the SiteMinder login page:



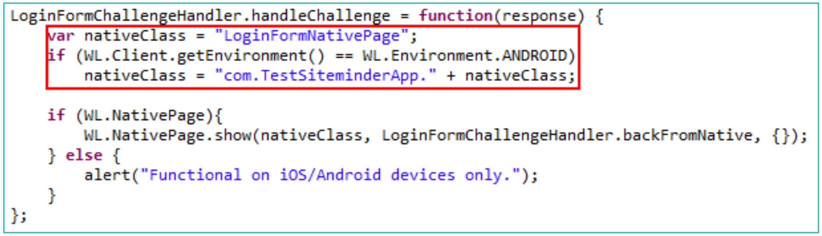
```
var LoginFormChallengeHandler = WL.Client.createChallengeHandler("SampleAppRealm");

LoginFormChallengeHandler.isCustomResponse = function(response) {
    if (!response || response.responseText === null) {
        return false;
    }

    if ((response.status == 200) && (response.responseText.indexOf("login.fcc") !== -1)) {
        return true;
    } else {
        return false;
    }
};
```

Creating the challenge handler

- This challenge handler makes some assumptions about your SiteMinder configuration:
  - **PreservePostData** is set to **yes**. This value is the default one.
  - The template file that is being used is named **login.fcc**.
- When SiteMinder is configured with **PreservePostData** set to **yes**:
  - The login page is not displayed immediately.
  - Instead, a temporary page is displayed to collect POST parameters that are then redirected to the login page.
  - The challenge handler then searches for some text in this temporary page.
    - Generally, a string to search for is the name of the FCC template file. In this case: **login.fcc**

Creating the challenge handler

- Continue implementation of the challenge handler with the following code, which is used to start an appropriate native page handler:

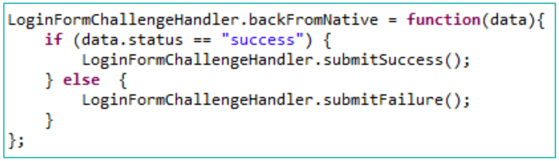


```
LoginFormChallengeHandler.handleChallenge = function(response) {
    var nativeClass = "LoginFormNativePage";
    if (WL.Client.getEnvironment() == WL.Environment.ANDROID)
        nativeClass = "com.TestSiteminderApp." + nativeClass;

    if (WL.NativePage){
        WL.NativePage.show(nativeClass, LoginFormChallengeHandler.backFromNative, {});
    } else {
        alert("Functional on iOS/Android devices only.");
    }
};
```

- The native page handler is native code that starts a browser instance that handles the remote authentication form.
- For this module, the class names that are used are **LoginFormNativePage** for iOS and **com.TestSiteMinderApp.LoginFormNativePage** for Android.
  – Your values might differ.

Creating the challenge handler

- Complete the implementation of the challenge handler with the following code, used to detect a successful login from the native page handler:



```
LoginFormChallengeHandler.backFromNative = function(data){
    if (data.status == "success") {
        LoginFormChallengeHandler.submitSuccess();
    } else  {
        LoginFormChallengeHandler.submitFailure();
    }
};
```

Agenda

- Background
- Creating the challenge handler
- Creating the native page handler
- Creating the native page handler (Android)
- Creating the native page handler (iOS)
- Examining the result
- Troubleshooting
- Exercise

Creating the native page handler

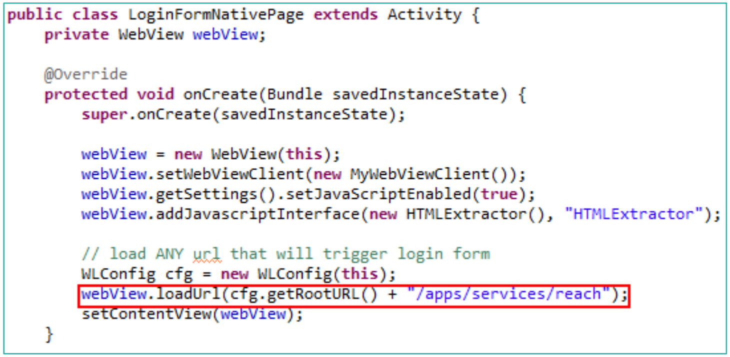
- The primary task of the native page handler is to load a SiteMinder remote authentication form.
- You cannot reuse the remote authentication form that caused this native page handler to be created.
- Therefore, an arbitrary URL is loaded that returns a SiteMinder remote authentication form.
- For the purposes of this module, assume that the Worklight Server itself is protected by SiteMinder.
- Therefore, the URL that is accessed is: **/apps/services/reach**
  - This URL exists within the context of the Worklight Server and is always available.

Agenda

- Background
- Creating the challenge handler
- Creating the native page handler
- Creating the native page handler (Android)
- Creating the native page handler (iOS)
- Examining the result
- Troubleshooting
- Exercise

Creating the native page handler (Android, 1 of 4)

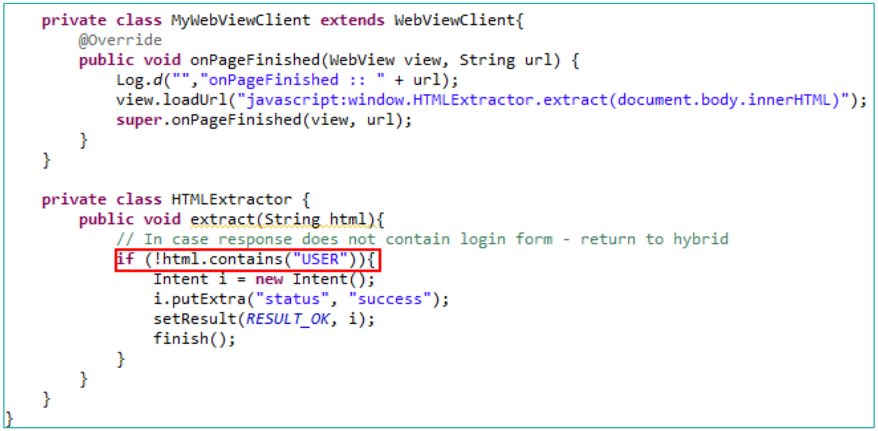
- Start the implementation of the native page handler by creating the class and its **onCreate()** method:



```
public class LoginFormNativePage extends Activity {
    private WebView webView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        webView = new WebView(this);
        webView.setWebViewClient(new MyWebViewClient());
        webView.getSettings().setJavaScriptEnabled(true);
        webView.addJavascriptInterface(new HTMLExtractor(), "HTMLExtractor");

        // load ANY url that will trigger login form
        WLConfig cfg = new WLConfig(this);
        webView.loadUrl(cfg.getRootURL() + "/apps/services/reach");
        setContentView(webView);
    }
```

Creating the native page handler (Android, 2 of 4)

- Complete the implementation of the native page handler with the following code:



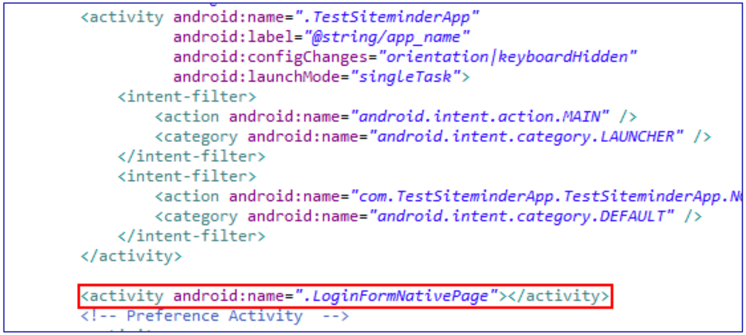
```
    private class MyWebViewClient extends WebViewClient{
        @Override
        public void onPageFinished(WebView view, String url) {
            Log.d("","onPageFinished :: " + url);
            view.loadUrl("javascript:window.HTMLExtractor.extract(document.body.innerHTML)");
            super.onPageFinished(view, url);
        }
    }

    private class HTMLExtractor {
        public void extract(String html){
            // In case response does not contain login form - return to hybrid
            if (!html.contains("USER")){
                Intent i = new Intent();
                i.putExtra("status", "success");
                setResult(RESULT_OK, i);
                finish();
            }
        }
    }
}
```

Creating the native page handler (Android, 3 of 4)

- Register the **HTMLExtractor.extract()** method as a JavaScript interface that you run after the page for the native page handler is loaded.
- In the **HTMLExtractor.extract()** method, verify that you are dealing with a SiteMinder login form.
  - Here, consider that you are dealing with a SiteMinder login form if the page contains a field that is called **USER**, as most non-customized SiteMinder login forms do.
- Do this check in the **HTMLExtractor.extract()** method to ensure that the native page handler does not run before a SiteMinder login form is presented.

Creating the native page handler (Android, 4 of 4)

- Finally, because the native page handler is an Android activity, add the appropriate definition in your application **AndroidManifest.xml** file:



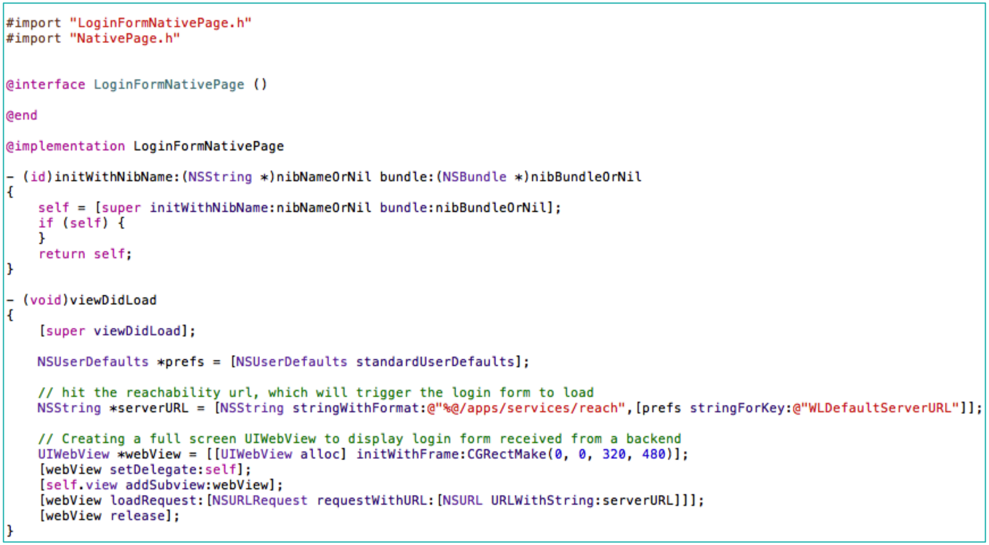
```
<activity android:name=".TestSiteminderApp"
          android:label="@string/app_name"
          android:configChanges="orientation|keyboardHidden"
          android:launchMode="singleTask">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.TestSiteminderApp.TestSiteminderApp.N
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

<activity android:name=".LoginFormNativePage"></activity>
<!-- Preference Activity  -->
```

Agenda

- Background
- Creating the challenge handler
- Creating the native page handler
- Creating the native page handler (Android)
- Creating the native page handler (iOS)
- Examining the result
- Exercise

Creating the native page handler (iOS, 1 of 3)

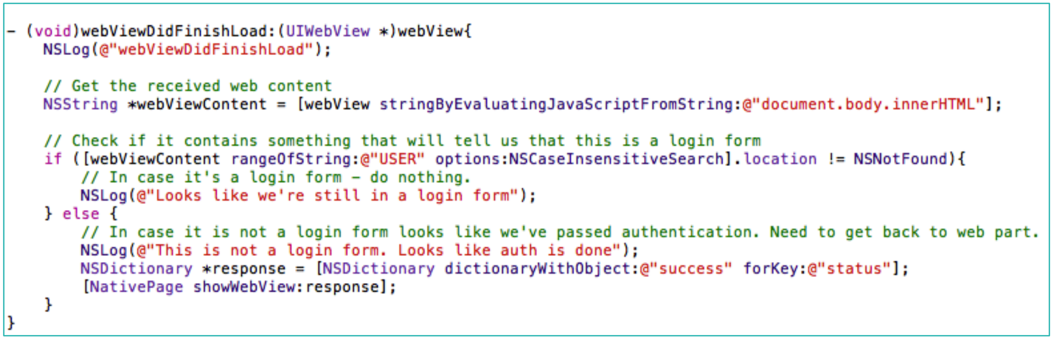
- Start the implementation of the native page handler by creating the **LoginFormNativePage.m** file and its **viewDidLoad** method:



```
#import "LoginFormNativePage.h"
#import "NativePage.h"


@interface LoginFormNativePage ()

@end

@implementation LoginFormNativePage

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];

    // hit the reachability url, which will trigger the login form to load
    NSString *serverURL = [NSString stringWithFormat:@"%@/apps/services/reach",[prefs stringForKey:@"WLDefaultServerURL"]];

    // Creating a full screen UIWebView to display login form received from a backend
    UIWebView *webView = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];
    [webView setDelegate:self];
    [self.view addSubview:webView];
    [webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:serverURL]]];
    [webView release];
}
```

Creating the native page handler (iOS, 2 of 3)

- Complete the implementation of the native page handler with the following code:



```
- (void)webViewDidFinishLoad:(UIWebView *)webView{
    NSLog(@"webViewDidFinishLoad");

    // Get the received web content
    NSString *webViewContent = [webView stringByEvaluatingJavaScriptFromString:@"document.body.innerHTML"];

    // Check if it contains something that will tell us that this is a login form
    if ([webViewContent rangeOfString:@"USER" options:NSCaseInsensitiveSearch].location != NSNotFound){
        // In case it's a login form - do nothing.
        NSLog(@"Looks like we're still in a login form");
    } else {
        // In case it is not a login form looks like we've passed authentication. Need to get back to web part.
        NSLog(@"This is not a login form. Looks like auth is done");
        NSDictionary *response = [NSDictionary dictionaryWithObject:@"success" forKey:@"status"];
        [NativePage showWebView:response];
    }
}
```

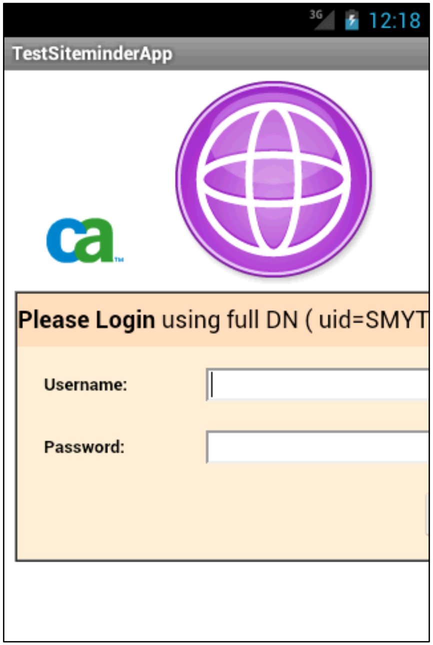
Creating the native page handler (iOS, 3 of 3)

- In the **webViewDidFinishLoad** method, verify that you are dealing with a SiteMinder login form.
  - Here, consider that you are dealing with a SiteMinder login form if the page contains a field that is called **USER**, as most non-customized SiteMinder login forms do.
- Do this check in the **webViewDidFinishLoad** method to ensure that the native page handler does not run before a SiteMinder login form is presented.

Agenda

- Background
- Creating the challenge handler
- Creating the native page handler (Android)
- Creating the native page handler (iOS)
- Examining the result
- Troubleshooting
- Exercise

Examining the result



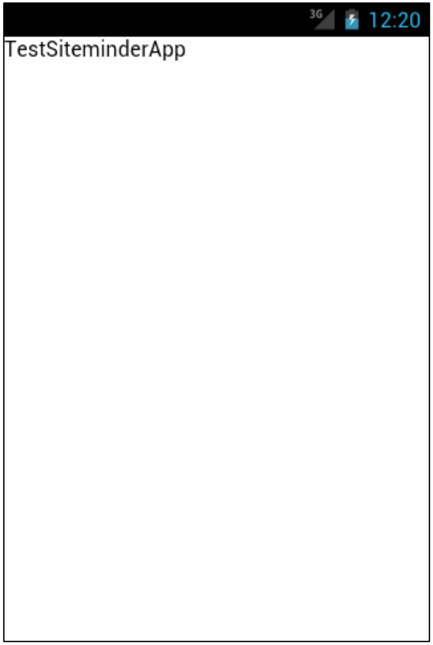
12:18

TestSiteminderApp

Please Login using full DN (uid=SMYT

Username:

Password:



12:20

TestSiteminderApp

Agenda

- Background
- Creating the challenge handler
- Creating the native page handler (Android)
- Creating the native page handler (iOS)
- Examining the result
- Troubleshooting
- Exercise

Troubleshooting

- Because SiteMinder can be customized, you might also customize this scenario.
- Focus on the following two areas when troubleshooting:
  - Was the login form properly detected?
    - Check the content of **/apps/services/reach** to make sure that you are receiving the login form as expected.
    - Ensure that the returned content contains the string that the challenge handler and the native page handler are looking for. You might have to look at the network response directly because, from the browser, you might not be able to identify a redirect for post data.
  - Did authentication complete successfully?
    - Check the network traffic to see whether cookies were set on the response.
    - Add logging to make sure that **submitSuccess** is being called locally.

Agenda

- Background
- Creating the challenge handler
- Creating the native page handler (Android)
- Creating the native page handler (iOS)
- Examining the result
- Troubleshooting
- Exercise

Exercise

- Implement the remote form-based authentication that is described in this module.
- The sample for this training module can be found in the Getting Started page of the IBM Worklight documentation website at http://www.ibm.com/mobile-docs

Notices

Support and comments

- For the entire IBM Worklight documentation set, training material and online forums where you can post questions, see the IBM website at:
  - http://www.ibm.com/mobile-docs
- **Support**
  - Software Subscription and Support (also referred to as Software Maintenance) is included with licenses purchased through Passport Advantage and Passport Advantage Express. For additional information about the International Passport Advantage Agreement and the IBM International Passport Advantage Express Agreement, visit the Passport Advantage website at:
    - http://www.ibm.com/software/passportadvantage
  - If you have a Software Subscription and Support in effect, IBM provides you assistance for your routine, short duration installation and usage (how-to) questions, and code-related questions. For additional details, consult your IBM Software Support Handbook at:
    - http://www.ibm.com/support/handbook
- **Comments**
  - We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.
  - For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.
  - When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state.
  - Thank you for your support.
  - Submit your comments in the IBM Worklight Developer Edition support community at:
    - https://www.ibm.com/developerworks/mobile/worklight/connect.html
  - If you would like a response from IBM, please provide the following information:
    - Name
    - Address
    - Company or Organization
    - Phone No.
    - Email address

Thank You