Networking

Workspace ONE for Android

Android applications can be integrated with the Workspace ONE® platform networking capabilities, by using its mobile software development kit.

Workspace ONE for Android networking capabilities include the following, for example:

- Tunneling of application data.
- NTLM and Basic Authentication.
- Certificate-based authentication.

This document is a collection of content from the earlier developer guide that hasn't yet been moved to the new integration guide set.

Table of Contents

Introduction

To integrate your Android application with Workspace ONE networking, integrate the AWNetworkLibrary. Instructions are given in this document.

See the new integration guide set for

- an introduction to SDK integration in general.
- details of the base integration tasks for the Framework integration level, which must be done before the tasks in this document.

An overview that includes links to the new integration guides is available

- in Markdown format, in the repository that also holds the sample code: https://github.com/euc-releases/workspace-ONE-SDK-integration-samples/blob/main/IntegrationGuideForAndroid/Guides/01Overview/WorkspaceONE_Android_IntegrationOverview.md
- in Portable Document Format (PDF), on the Omnissa website: https://developer.omnissa.com/ws1-sdk-for-android/guides/WorkspaceONE_Android_IntegrationOverview.pdf

Set Up Gradle and Initialize the AWNetworkLibrary

To integrate the AWNetworkLibrary, add the necessary dependencies to the Gradle project and initialize the AWNetworkLibrary.

Procedure

1. Set up Gradle. In the application build.gradle file, in the dependencies block, add references to the required libraries. For example:

```
packagingOptions {
    exclude 'META-INF/INDEX.LIST'
    exclude 'META-INF/io.netty.versions.properties'
}

repositories {
    maven {
        url = uri("https://maven.pkg.github.com/euc-releases/Android-WorkspaceONE-SDK/")
        credentials {
            /*
            Update gradle.properties of the project with following entries
            github.user=GITHUB_USER_NAME
            github.token=GITHUB_PERSONAL_ACCESS_TOKEN
            */
            username = project.findProperty("github.user") ?: System.getenv("USERNAME")
            password = project.findProperty("github.token") ?: System.getenv("TOKEN")
        }
    }
    maven {
        url = uri("https://maven.pkg.github.com/euc-releases/ws1-intelligencesdk-sdk-android/")
        credentials {
            /*
            Update gradle.properties of the project with following entries
            github.user=GITHUB_USER_NAME
            github.token=GITHUB_PERSONAL_ACCESS_TOKEN
            */
            username = project.findProperty("github.user") ?: System.getenv("USERNAME")
            password = project.findProperty("github.token") ?: System.getenv("TOKEN")
        }
    }
}

dependencies {
    ...
    // In addition to AWFramework entries add the listed library
    implementation "com.airwatch.android:awnetworklibrary:25.02.3"
}
```

2. Initialize the AWNetworkLibrary.
3. Set GatewaySplashActivity as your main launching activity instead of SDKSplashActivity.

```
    <activity
        android:name="com.airwatch.gateway.ui.GatewaySplashActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

4. Extend GatewayBaseActivity at the activity level to support network features like tunneling and integrated authentication in addition to AWFramework features.

Use the AWNetworkLibrary

The AWNetworkLibrary library provides network capabilities that include the following.

- Tunneling of application data.
- NTLM and Basic Authentication.
- Certificate-based authentication.

To make use of the AWNetworkLibrary in your application, change any code that uses native networking classes to use wrapper classes from the library instead. The following table lists original classes and their replacements.

Original Class	Wrapper Class
WebView	AWWebView
WebViewClient	AWWebViewClient
URLConnection	AWURLConnection
OkHttpClient	AWOkHttpClient

Note for OkHttp users

If you use the OkHttp library in your application, you can create a wrapped instance of the OkHttp client with code like the following.

```
OkHttpClient awOkClient = AWOkHttpClient.copyWithDefaults(
    context, client, new X509TrustManager(){
        public java.security.cert.X509Certificate[] getAcceptedIssuers() {
            return new X509Certificate[0];
        }

        public void checkClientTrusted(X509Certificate[] certs, String authType) {
        }

        public void checkServerTrusted(X509Certificate[] certs, String authType) {
        }
    }
);
```

From version 21.5.1 of the Workspace ONE software development kit for Android:

- The X509TrustManager parameter will be used to secure HTTPS connections. In the above snippet, a default implementation is instantiated for the purpose.
- The `context` should be an Activity so that an enrollment credential dialog can be shown when authentication fails. Failure could occur after an Active Directory password change, for example.

Support for Tunneling in Android 10 (Q)

The implementation of tunneling changed in the Workspace ONE SDK for Android to support Android 10 (Q). To use tunneling with your SDK-built apps successfully, follow the listed troubleshooting tips and use the AWSDK-provided wrapper classes.

Loss of Support for `/proc/net`

Prior to 19.6 version, the AWNetworkLibrary used app UIDs retrieved from the /proc/net filesystem to authenticate and send specific app traffic through an internal SDK proxy. This proxy then sent this specific traffic to the Omnissa Tunnel - Proxy server.

Due to restrictions introduced in Android Q, the AWNetworkLibrary can no longer use the /proc/net filesystem to authenticate app traffic through the SDK internal proxy.

For background, see the relevant pages on the Android developer website, for example: https://developer.android.com/.../changes#proc-net-filesystem

(Some PDF viewers incorrectly escape the hash anchor marker in the above link. If that happens, edit the link in the browser address bar.)

What Does the SDK Use to Authenticate Traffic?

The Workspace ONE SDK for Android uses AWSDK-provided wrapper classes to authenticate traffic.

The wrapper classes include authentication tokens used for each request for traffic through the SDK internal proxy. Apps must use these wrapper classes to support Android Q.

The SDK sample application provides example usages for common HTTP clients like URLConnection, OKHttpClient, Volley, Retrofit, and Picasso. Refer to `ProxyTestActivity` and `AWClientExampleActivity` for examples on how to use these wrapper classes.

Troubleshooting Tips

- Before version 19.6 of the Workspace ONE SDK for Android

  Apps using SDK versions older than 19.6 fail with a “Connection reset” or “Tunnel connection failed” error on Android 10 (Q) devices. Upgrade your apps to Workspace ONE SDK for Android v19.6 or later and use the AW wrapper classes.

- Version 19.6 of the Workspace ONE SDK for Android and later

  Apps using SDK versions 19.6 and later, but not using the AW wrapper classes fail with a “407 Proxy Authentication Required” error. Set your apps to use the AW wrapper classes to resolve this problem.

Tunneling Support with Tunnel

To tunnel application traffic with and the Workspace ONE SDK, you do not need the provided HTTP clients. Configure the SDK default settings for AirWatch App Tunnel in the Workspace ONE UEM console and the SDK tunnels traffic according to this configuration.

Prerequisites

- You must have a valid Tunnel deployment. Refer [Quick Start to Deploying Tunnel on Android] (https://docs.omnissa.com/bundle/Workspace_ONE_TunnelV2406/page/TunnelQuickStartDeployAndroid.html) for details.
- Set Up Gradle and Initialize the AWNetworkLibrary.

Procedure

1. In the Workspace ONE UEM console, navigate to **Groups & Settings > All Settings > Apps > Settings and Policies > Security Policies > AirWatch App Tunnel**
2. Select **Enabled** and then select **Omnissa Tunnel** for the **App Tunnel Mode**.

Integrated Authentication

The integrated authentication (IA) capability of the AWNetworkLibrary can be used to authenticate to a web server using any of the following.

- NTLM authentication.
- Basic authentication.
- Certificate-based authentication (CBA).

Sharing and Downloading Certificates with SSO Enabled

The Workspace ONE SDK for Android will share and download IA certificates if the certificates aren't available from other SDK apps in the cluster.

Certificate sharing is controlled by the Workspace ONE unified endpoint manager (UEM) console. In the console, in the SDK settings, if single sign-on (SSO) is enabled, then certificates will be shared.

Integrated Authentication Programming Interfaces

There are different IA programming interfaces, for use with WebView and for use with different HTTP clients.

Note: You can find sample code that uses the IA programming interface in the sample application, in the IntegratedAuthActivity.java file.

Programming Interface for WebView

Support all types of IA in a WebView as follows.

1. Create an AWWebViewClient subclass in your app code and customize the methods as required.
2. Set an instance of your AWWebViewClient subclass as the WebViewClient for your WebView.

The above applies to NTLM, Basic Authentication, and CBA in a WebView.

Programming Interfaces for NTLM in HTTP Clients

Support NTLM IA for different HTTP clients as follows.

HTTP Client	Integration
Apache HttpClient	Register an NTLM AuthScheme and add an AWAuthInterceptor request interceptor.
UrlConnection	No change, use NTLMHttpUrlConnection.
OkHttpClient	Set an AWOkHttpAuthenticator instance as an authenticator.

Programming Interfaces for Basic Authentication in HTTP Clients

Support Basic Authentication IA for different HTTP clients as follows.

HTTP Client	Integration
Apache HttpClient	Not supported.
UrlConnection	No change, use BasicAuthHttpUrlConnection.
OkHttpClient	Set an AWOkHttpAuthenticator instance as an authenticator.

Programming Interfaces for Certificate-Based Authentication in HTTP Clients

Support CBA IA for different HTTP clients by using AWCertAuthUtil to construct an SSLContext instance with the required certificates for authentication.

For example, you would support CBA IA with Apache HttpClient as follows.

1. Get a list of KeyManager objects by calling the AWCertAuthUtil getCertAuthKeyManagers() method.

2. Construct an SSLContext instance from the KeyManager list.
3. Get an SSLSocketFactory instance from the SSLContext.
4. Use the SSLSocketFactory with the HttpClient.

To support CBA IA with UrlConnection or OkHttpClient make changes corresponding to the above but in the specific programming interface of the HTTP client.

SCEP Support to Retrieve Certificates for Integrated Authentication

The Workspace ONE SDK supports the SCEP protocol, with limitations, to retrieve certificates for integrated authentication.

To use SCEP certificates for your SDK-built application, ensure integrated authentication is enabled and that SCEP is configured in the console as a certificate authority.

Supported SAN Information Types

The SDK fully supports the listed Subject Alternative Names (SAN) information types in certificate attributes.

- dNSName
- ntPrincipalName

  Note: When you configure this information type, it displays as an entry nested under the otherName attribute. Although otherName is not supported, ntPrincipalName is supported even as a nested entry of otherName.

- rfc822Name
- uniformResourceIdentifier

Supported with Correct Format

The Workspace ONE SDK supports the listed SAN information types but you must use the correct format or the SDK ignores them.

- iPAddress
- registeredID

Not Supported

The Workspace ONE SDK does not support the listed SAN information types. If you configure them, the SCEP process fails.

- Custom
- directoryName
- ediPartyName
- GUID
- otherName
- x400Address

APIs for a Pending Status from the SCEP Certificate Authority

Use the SCEPCertificateFetcher method to modify SCEP certificate fetches to poll the SCEP server and to refetch certificates when the server returns a pending status.

When using SCEP, some configurations set the SCEP certificate authority not to issue the certificate until the request is approved. In this scenario, the authority returns a pending status to the SDK.

The Workspace ONE SDK for Android fetches SCEP certificates when you enable integrated authentication and have SCEP configured as a certificate authority. However, if you want your application to handle the listed scenarios, you must modify code.

- You want the application to handle a pending status result for a certification fetch.
- You want the application to know the result of the fetch and to act based on the result.

Ensure the Certificate Authority server handles retry requests. The SDK retries the fetch request based on the parameters in the modified code or using the default behavior (retries every 5 milliseconds for 10 tries). If the server is not configured to handle retry requests caused by the pending status, the fetch never finishes.

To handle the pending result and to poll the server to refetch, modify the SCEPCertificateFetcher method.

Procedure

1. Set the retry interval and maximum retry count.

```
/**
 * Sets the maximum number of retry attempts. Once this is elapsed, the pending status is
 * cleared, polling stops and the next fetch results in a new SCEP request for a new
 * certificate.
 */
void setMaxRetryCount(int maxRetryCount);
/**
 * Sets the retry interval between fetch attempts (in seconds).
 */
void setRetryIntervalSeconds(int retryIntervalSeconds);
```

2. Pause and resume the polling mechanism. Use the Activity Lifecycle methods so that the polling mechanism does not run when the application is not in the foreground.

```
/**
 * Triggers/resumes scheduled polling for fetching "Pending" certificates. To be  called
 * when the application is brought to foreground(in any Activity's onResume()).
 */
void triggerPolling();
/**
 * Pause polling for SCEP pending certificates. To be called when the application is
 * sent to background (in any Activity's onPause()).
 */
boolean pausePolling();
/**
 * Returns true if a SCEP certificate fetch is pending. This will be reset to false when the
 * SCEP certificate polling results in a success or failure or if maximum number of attempts
 * is exceeded.
 */
boolean isSCEPCertificatePending();
```

3. Register listeners to identify the SCEP certificate fetch result.

```
/**
 * Register an implementation of {@link SCEPCertificateFetchListener} to listen for fetch
 * results. Ensures that one instance is added only once. Unregister the listeners using
 * {@link #unregisterFetchListener(SCEPCertificateFetchListener)} when callbacks are no longer
 * required in order to prevent memory leaks of the listener implementation.
 */
void registerFetchListener(SCEPCertificateFetchListener statusListener);

/**
 * Unregister the registered listener when callbacks are no longer necessary. This ensures
 * that the listeners are not leaked.
 */
void unregisterFetchListener(SCEPCertificateFetchListener statusListener);
```

4. Obtain an instance of the SCEPCertificateFetcher by calling:
   `SCEPContext.getInstance().getSCEPCertificateFetcher()`

   Example of an Activity to Poll and Listen for SCEP Fetch Results

```
public class IntegratedAuthActivity extends AppBaseActivity implements View.OnClickListener{

    private static final String TAG = "IntegratedAuthActivity";

    private static final String SCEP_MAX_COUNT_KEY = "max_count";
    private static final String SCEP_RETRY_INTERVAL_KEY = "retry_interval";

    private SCEPCertificateFetchListener certStatusListener =
    new SCEPCertificateFetchListener() {
        @Override
        public void onResult(CertificateFetchResult certificateFetchResult) {
            handleResult(certificateFetchResult);
        }
    };

    private SCEPCertificateFetcher certificateFetcher =
SCEPContext.getInstance().getSCEPCertificateFetcher();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ntlm_and_basic_auth);
        certificateFetcher.registerFetchListener(certStatusListener);

        final SharedPreferences pref = PreferenceManager.getDefaultSharedPreferences(this);

        //These values are fetched from SharedPreferences.
        int maxCount = pref.getInt(SCEP_MAX_COUNT_KEY,
SCEPCertificateFetcher.DEFAULT_MAX_RETRY_COUNT);
        int retryInterval = pref.getInt(SCEP_RETRY_INTERVAL_KEY,
SCEPCertificateFetcher.DEFAULT_RETRY_INTERVAL_SECONDS);

        certificateFetcher.setMaxRetryCount(maxCount);
        certificateFetcher.setRetryIntervalSeconds(retryInterval);
    }

    @Override
    protected void onResume() {
        super.onResume();

        if(certificateFetcher.isSCEPCertificatePending()){
            certificateFetcher.triggerPolling();
        }
    }

    @Override
    protected void onPause() {
        super.onPause();

        pauseSCEPCertificatePolling();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();

        certificateFetcher.unregisterFetchListener(certStatusListener);
    }

    private void handleResult(CertificateFetchResult result) {
        switch (result.getStatus()){
            case SUCCESS:
                showSnackbar("SCEP certificate fetch succeeded");
                break;
            case FAILURE:
                String errorString = getErrorString(result.getErrorCode());
                showSnackbar("SCEP certificate fetch failed. " + errorString);
                break;
            case PENDING:
                String messageString = getErrorString(result.getErrorCode());
                PendingRetryDataModel retryDataModel = result.getPendingRetryDataModel();
                String retryMessage = "Attempts remaining: " +
retryDataModel.getRetryAttemptsRemaining() +
                        ". Time remaining for next retry: " +
retryDataModel.getTimeRemainingForNextRetryAttempt();
                showSnackbar(messageString + " " + retryMessage);
                break;
        }
    }
}
```

Document Information

Revision History

The following table shows the revision history of this document.

Date	Revision
30jul2020	First publication, for 20.7 SDK for Android.
09sep2020	Updated for 20.8 SDK for Android.
11oct2020	Updated for 20.9 SDK for Android.
17jun2021	Update for 21.5.1 SDK for Android.
17jul2021	Update for 21.6 SDK for Android.
20sep2021	Update for 21.8 SDK for Android.
26oct2021	Update for 21.10 SDK for Android.
09Dec2021	Update for 21.11 SDK for Android.
26Jan2022	Update for 22.1 SDK for Android.
28Feb2022	Update for 22.2 SDK for Android.
04Apr2022	Update for 22.3 SDK for Android.
29Apr2022	Update for 22.4 SDK for Android.
06Jun2022	Update for 22.5 SDK for Android.
05Jul2022	Update for 22.6 SDK for Android.
23Aug2022	Update for 22.8 SDK for Android.
04Nov2022	Update for 22.10 SDK for Android.
13Dec2022	Update for 22.11 SDK for Android.
25Jan2023	Update for 23.01 SDK for Android.
15Mar2023	Update for 23.03 SDK for Android.
27Apr2023	Update for 23.04 SDK for Android.
06Jun2023	Update for 23.06 SDK for Android.
24Jul2023	Update for 23.07 SDK for Android.
07Sep2023	Update for 23.09 SDK for Android.
25Oct2023	Update for 23.10 SDK for Android.
18Dec2023	Update for 23.12 SDK for Android.
15May2024	Update for 24.04 SDK for Android.
05Jul2024	Update for 24.06 SDK for Android.
28Aug2024	Update for 24.07 SDK for Android.
18Feb2025	Brand Revision.
04Mar2025	Documentation Update for Android.
04Aug2025	Update for 25.02.3 SDK for Android.

Legal