DRD19. Properly verify server certificate on SSL/TLS

Android apps that use SSL/TLS protocols for secure communication should properly verify server certificates. The basic verification includes:

- verify that the subject (CN) of X.509 certificate and the URL matches
- verify that the certificate is signed by the trusted CA
- verify that the signature is correct
- verify that the certificate is not expired

Android SDK 4.0 and later offers packages to implement capabilities to establish network connections. For example, by using `java.net`, `javax.net`, an `droid.net` or `org.apache.http`, a developer can create server sockets or HTTP connection. `org.webkit` offers functions necessary to implement web browsing capabilities.

A developer has the freedom to customize their SSL implementation. The developer should properly use SSL as appropriate to the intent of the app and the environment the apps are used in. If the SSL is not correctly used, a user's sensitive data may leak via the vulnerable SSL communication channel.

Fahl et al [Fahl 2012] describes the following patterns of the insecure use of SSL:

- **Trusting All Certificates:** The developer implements the TrustManager interface so that it will trust all the server certificate (regardless of who signed it, what is the CN etc.)
- **Allowing All Hostnames:** The app does not verify if the certificate is issued for the URL the client is connecting to. For example, when a client connects to `example.com`, it will accept a server certificate issued for `some-other-domain.com`.
- **Mixed-Mode/No SSL:** A developer mixes secure and insecure connections in the same app or does not use SSL at all.

On Android, using `HttpURLConnection` is recommended for HTTP client implementation.

Noncompliant Code Example

The following code implements a custom `MySSLSocketFactory` class that inherits `javax.net.ssl.SSLContext`:

```
public class extends SSLSocketFactory {
    SSLContext sslContext;
public MySSLSocketFactory (KeyStore truststore) throws NoSuchAlgorithmException, KeyManagementException,
KeyStoreException, UnrecoverableKeyException {
    super(truststore);
    this.sslContext = SSLContext.getInstance("TLS");
    this.sslContext.init (null, new TrustManager[] {new X509TrustManager() {
            public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException
            {
            }
            public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException
            {
            }
            public X509Certificate[] getAcceptedIssuers() {
                return null;
            }
        }}, null);
}
    public Socket createSocket() throws IOException {
        return this.sslContext.getSocketFactory().createSocket();
    }
    public Socket createSocket(Socket socket, String host, int port, boolean autoClose) throws IOException
                                                                                                    ,
UnknownHostException {
        return this.sslContext.getSocketFactory().createSocket(socket, host, port, autoClose);
    }
}

 public static HttpClient getNewHttpClient() {
            DefaultHttpClient v6;
            try {
                KeyStore v5 = KeyStore.getInstance(KeyStore.getDefaultType());
                v5.load(null, null);
                MySSLSocketFactory mySSLScoket = new MySSLSocketFactory(v5);
                if(DefineRelease.sAllowAllSSL) {
                    ((SSLSocketFactory)mySSLScoket).setHostnameVerifier(SSLSocketFactory.
ALLOW_ALL_HOSTNAME_VERIFIER);
                }
                BasicHttpParams v2 = new BasicHttpParams();
                HttpConnectionParams.setConnectionTimeout(((HttpParams)v2), 30000);
                HttpConnectionParams.setSoTimeout(((HttpParams)v2), 30000);
                HttpProtocolParams.setVersion(((HttpParams)v2), HttpVersion.HTTP_1_1);
                HttpProtocolParams.setContentCharset(((HttpParams)v2), "UTF-8");
                SchemeRegistry v3 = new SchemeRegistry();
                v3.register(new Scheme("http", PlainSocketFactory.getSocketFactory(), 80));
                v3.register(new Scheme("https", ((SocketFactory)mySSLScoket), 443));
                v6 = new DefaultHttpClient(new ThreadSafeClientConnManager(((HttpParams)v2), v3), ((HttpParams)
v2));
            }
            catch(Exception v1) {
                v6 = new DefaultHttpClient();
            }
            return ((HttpClient)v6);
}
```

In the example above, `checkClientTrusted()`and `checkServerTrusted()` are overriden to make a blank implementation so that `SSLSocketFactory` does not verify the SSL certificate. The `MySSLSocketFactory` class is used to create an instance of `HttpClient` in another part of the application.

`sAllowAllSSL`, which is a static member of the `DefineRelease` class, is initialized to `true` in its static constructor. This will enable the use of `SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER`. As a result, host name verification that should take place when establishing an SSL connection is disabled and will lead to the same situation as all the certificate is trusted.

Compliant Solution

The compliant solution may vary, depending on the actual implementation. For examples of secure implementation such as using a self-signed server certificate, please refer to "Android Application Secure Design/Secure Coding Guidebook", Section 5.4 Communicate by HTTPS.

Risk Assessment

Not properly verifying the server certificate on SSL/TLS may allow apps to connect to an imposter site, while fooling the user into thinking that the user is connected to an intended site. One example of associated risks is that this could expose a user's sensitive data.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
DRD19-J	High	Probable	Medium	P12	L1

Automated Detection

It is possible to automatically detect whether an application uses one of the three Android SDK packages named for establishing network connections, and to check if any of the methods from those classes are overriden by the application. It is not feasible to automatically determine the intent of the app or the environment the apps are used in.

Related Vulnerabilities

- VU#582497 Multiple Android applications fail to properly validate SSL certificates
- JVN#39218538 Pizza Hut Japan Official Order App for Android has a problem whereby it fails to verify SSL server certificates.
- JVN#75084836 Yome Collection for Android has a problem with management of IMEI.
- JVN#68156832 Yafuoku! contains an issue where it fails to verify SSL server certificates.

Related Guidelines

Android Secure Design / Secure Coding Guidebook by JSSEC	5.4 Communicating via HTTPS

Bibliography

Fahl 2012	Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security