Securing Wireless Channels in the Mobile Space

Supplement for OWASP Presentation

Northern Virgina OWASP

February 7, 2013

iOS

```
-(IBAction)fetchButtonTapped:(id)sender
{
    [m_fetchedLabel setText:@""];

    NSString* requestString = @"https://www.random.org/integers/?
        num=16&min=0&max=255&col=16&base=16&format=plain&rnd=new";
    NSURL* requestUrl = [NSURL URLWithString:requestString];
    ASSERT(nil != requestUrl);
    if(!(nil != requestUrl))
    {
        [self displayError:@"Failed to create requestString."];
        return;
    }

    NSURLRequest* request = [NSURLRequest requestWithURL:requestUrl
                                             cachePolicy:NSURLRequestReloadIgnoringLocalCacheData
                                         timeoutInterval:10.0f];
    ASSERT(nil != request);
    if(!(nil != request))
    {
        [self displayError:@"Failed to create request."];
        return;
    }

    // Note that the delegate for the NSURLConnection is self, so delegate methods must be defined in this
        file
    NSURLConnection* connection = [[NSURLConnection alloc] initWithRequest:request delegate:self];
    ASSERT(nil != connection);
    if(!(nil != connection))
    {
        [self displayError:@"Failed to create connection."];
        return;
    }

    self.m_fetchedData = [[NSMutableString string] retain];
    ASSERT(nil != m_fetchedData);
}
```

```
-(BOOL)connection:(NSURLConnection *)connection canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace
    *)space
{
    return [[space authenticationMethod] isEqualToString: NSURLAuthenticationMethodServerTrust];
}

- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationChallenge:
    (NSURLAuthenticationChallenge *)challenge
{
    // Use the following to fetch the cert of interest. It will be in PEM format.
    // PEM format is (--BEGIN CERTIFICATE--, --END CERTIFICATE--).
    //     $ echo "Get HTTP/1.0" | openssl s_client -showcerts -connect www.random.org:443
    // Save the certifcate of interest to a file (for example, "random-org.pem").
    //   The certificate is the leaf, and should be located at certifcates[0] for
    //   non-ephemeral exchanges. Then, convert the certifcate to DER.
    //     $ openssl x509 -in "random-org.pem" -inform PEM -out "random-org.der" -outform DER

    if ([[[challenge protectionSpace] authenticationMethod] isEqualToString:
        NSURLAuthenticationMethodServerTrust])
    {
        do
        {
            SecTrustRef serverTrust = [[challenge protectionSpace] serverTrust];
            ASSERT(nil != serverTrust);
            if(!(nil != serverTrust)) break; /* failed */

            OSStatus status = SecTrustEvaluate(serverTrust, NULL);
            ASSERT(errSecSuccess == status);
            if(!(errSecSuccess == status)) break; /* failed */

            SecCertificateRef serverCertificate = SecTrustGetCertificateAtIndex(serverTrust, 0);
            ASSERT(nil != serverTrust);
            if(!(nil != serverTrust)) break;  /* failed */

            CFDataRef serverCertificateData = SecCertificateCopyData(serverCertificate);
            ASSERT(nil != serverCertificateData);
            if(!(nil != serverCertificateData)) break;  /* failed */

            [(id)serverCertificateData autorelease];
            const UInt8* const data = CFDataGetBytePtr(serverCertificateData);
            const CFIndex size = CFDataGetLength(serverCertificateData);
```

```
            ASSERT(nil != data);
            ASSERT(size > 0);
            if(!(nil != data) || !(size > 0)) break; /* failed */

            // (lldb) p data
            // (const UInt8 *) $0 = 0x1d8d3820
            // (lldb) p size
            // (CFIndex) $1 = 1772
            // (lldb) po serverCertificateData
            // $2 = 0x1d8d3800   <308206e8 308205d0 a0030201 02021074 b805ae19
            // e5ad4bed 4c3a20e6af02c930 0d06092a 864886f7 0d010105 05003081 ... >

            NSData* cert1 = [NSData dataWithBytes:data length:(NSUInteger)size];
            ASSERT(nil != cert1);
            if(!(nil != cert1)) break; /* failed */

            NSString *file = [[NSBundle mainBundle] pathForResource:@"random-org" ofType:@"der"];
            ASSERT(nil != file);
            if(!(nil != file)) break; /* failed */

            NSData* cert2 = [NSData dataWithContentsOfFile:file];
            ASSERT(nil != cert2);
            if(!(nil != cert2)) break; /* failed */

            const BOOL equal = [cert1 isEqualToData:cert2];
            ASSERT(NO != equal);
            if(!(NO != equal)) break; /* failed */

            // The only good exit point
            return [[challenge sender] useCredential: [NSURLCredential credentialForTrust: serverTrust]
                          forAuthenticationChallenge: challenge];

        } while (0);
    }

    // Bad dog
    return [[challenge sender] cancelAuthenticationChallenge: challenge];
}
```

Android

```
import java.io.InputStreamReader;
import java.io.StreamTokenizer;
import java.net.URL;

import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;

// http://android-developers.blogspot.com/2009/05/painless-threading.html
public class FetchSecretTask extends AsyncTask<Void, Void, Object> {

    @Override
    protected Object doInBackground(Void... params) {

        Object result = null;

        try {

            byte[] secret = null;

            TrustManager tm[] = { new PubKeyManager() };
            assert (null != tm);

            SSLContext context = SSLContext.getInstance("TLS");
            assert (null != context);
            context.init(null, tm, null);

            URL url = new URL( "https://www.random.org/integers/?" +
                "num=16&min=0&max=255&col=16&base=10&format=plain&rnd=new");
            assert (null != url);

            HttpsURLConnection connection = (HttpsURLConnection) url.openConnection();
            assert (null != connection);

            connection.setSSLSocketFactory(context.getSocketFactory());
            InputStreamReader instream = new InputStreamReader(connection.getInputStream());
            assert (null != instream);

            StreamTokenizer tokenizer = new StreamTokenizer(instream);
            assert (null != tokenizer);

            secret = new byte[16];
```

```
            assert (null != secret);

            int idx = 0, token;
            while (idx < secret.length) {
                token = tokenizer.nextToken();
                if (token == StreamTokenizer.TT_EOF)
                    break;
                if (token != StreamTokenizer.TT_NUMBER)
                    continue;

                secret[idx++] = (byte) tokenizer.nval;
            }

            // Prepare return value
            result = (Object) secret;

        } catch (Exception ex) {
        }

        return result;
    }
}
```

```
import java.math.BigInteger;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPublicKey;
import javax.net.ssl.X509TrustManager;

public final class PubKeyManager implements X509TrustManager {

    // DER encoded public key
    private static String PUB_KEY = "30820122300d06092a864886f70d0101" +
    "0105000382010f003082010a0282010100b35ea8adaf4cb6db86068a836f3c85" +
    "5a545b1f0cc8afb19e38213bac4d55c3f2f19df6dee82ead67f70a990131b6bc" +
    "ac1a9116acc883862f00593199df19ce027c8eaaae8e3121f7f329219464e657" +
    "2cbf66e8e229eac2992dd795c4f23df0fe72b6ceef457eba0b9029619e0395b8" +
    "609851849dd6214589a2ceba4f7a7dcceb7ab2a6b60c27c69317bd7ab2135f50" +
    "c6317e5dbfb9d1e55936e4109b7b911450c746fe0d5d07165b6b23ada7700b00" +
    "33238c858ad179a82459c4718019c111b4ef7be53e5972e06ca68a112406da38" +
    "cf60d2f4fda4d1cd52f1da9fd6104d91a34455cd7b328b02525320a35253147b" +
    "e0b7a5bc860966dc84f10d723ce7eed5430203010001";

    public void checkServerTrusted(X509Certificate[] chain, String authType) {

        assert (chain != null);
        if (chain == null) {
            throw new IllegalArgumentException( "checkServerTrusted: X509Certificate array is null");
        }

        assert (chain.length > 0);
        if (!(chain.length > 0)) {
            throw new IllegalArgumentException( "checkServerTrusted: X509Certificate is empty");
        }

        assert (null != authType && authType.equalsIgnoreCase("RSA"));
        if (!(null != authType && authType.equalsIgnoreCase("RSA"))) {
            throw new IllegalArgumentException(
                    "checkServerTrusted: AuthType is not RSA");
        }

        // Hack ahead: BigInteger and toString(). We know a DER encoded
        // Public Key starts with 0x30 (ASN.1 SEQUENCE and CONSTRUCTED),
        // so there is no leading 0x00 to drop.
        RSAPublicKey pubkey = (RSAPublicKey)chain[0].getPublicKey();
```

```
        String encoded = new BigInteger(1 /*positive*/, pubkey.getEncoded()).toString(16);

        // Test the resulting hash
        final boolean expected = PUB_KEY.equalsIgnoreCase(encoded);
        assert (expected);
        if (!expected) {
            throw new IllegalArgumentException(
                    "checkServerTrusted: watch out. Expected public key (hash): " +
                    PUB_KEY + ", got public key: " + encoded);
        }
    }

    public void checkClientTrusted(X509Certificate[] xcs, String string) {
    }

    public X509Certificate[] getAcceptedIssuers() {
        return null;
    }
}
```

.Net (C#)

```
using System;
using System.Net;
using System.Net.Security;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;

namespace PinPubKeyTest
{
  class Program
  {
    // Encoded public key
    private static String PUB_KEY = "30818902818100C4A06B7B52F8D17DC1CCB47362" +
      "C64AB799AAE19E245A7559E9CEEC7D8AA4DF07CB0B21FDFD763C63A313A668FE9D764E" +
      "D913C51A676788DB62AF624F422C2F112C1316922AA5D37823CD9F43D1FC54513D14B2" +
      "9E36991F08A042C42EAAEEE5FE8E2CB10167174A359CEBF6FACC2C9CA933AD403137EE" +
      "2C3F4CBED9460129C72B0203010001";

    public static void Main(string[] args)
    {
      ServicePointManager.ServerCertificateValidationCallback = PinPublicKey;
      WebRequest wr = WebRequest.Create("https://encrypted.google.com/");
      wr.GetResponse();
    }

    public static bool PinPublicKey(object sender, X509Certificate certificate, X509Chain chain,
        SslPolicyErrors sslPolicyErrors)
    {
      if (certificate == null || chain == null)
        return false;

      if (sslPolicyErrors != SslPolicyErrors.None)
        return false;

      // Verify against known public key within the certificate
      String pk = certificate.GetPublicKeyString();
      if (!pk.Equals(PUB_KEY))
        return false;

      return true;
    }
  }
}
```