Павел @LrdSpr
Технический консультант по Dynamics CRM

22 апреля 2014 в 21:58

Авторизация с помощью клиентских SSL сертификатов в IOS и Androi

Разработка под Android*, Разработка под iOS*

Протокол безопасной передачи данных SSL (Secure Sockets Layer) помимо обеспечения безопасной передачи данных так же позволяет реали авторизацию клиентов при помощи клиентских SSL сертификатов. Данная статья является практическим руководством по реализации данно авторизации в мобильных приложениях на IOS и Android.

Процесс организации работы сервера обеспечивающего такой вид авторизации в статье не рассматривается, однако в конце приведены ссыл данной тематике.

Процесс авторизации выглядит следующим образом. При переходе клиента в закрытую область сервер запрашивает у клиента сертификат, е проверка прошла успешно то клиент получает доступ к закрытому контенту в ином случае клиент может получить ошибку "**No required SSL certificate was sent**".

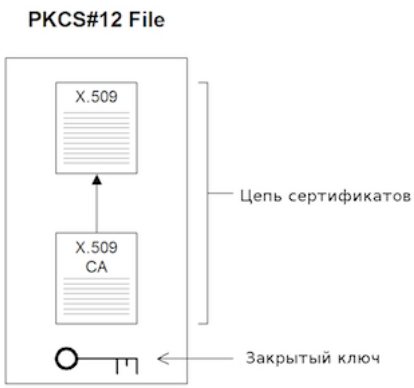
Для организации подключения мы сгенирировали клиентский сертификат, а так же создали запрос на подписание сертификата в результате получили файл client.csr. Далее мы отправили данный файл поставщику услуг и получили наш подписанный клиентский сертификат необход для аутентификации на удаленном сервере.

Тестирование подключения может быть осуществлено при помощи утилиты curl.

curl cert client.crt key client.key k someserive.com

Однако стоит заметить, что в последняя версия curl 7.30.0 в OS X сломана и не может быть использована для организации тестирования (http://curl.haxx.se/mail/archive-2013-10/0036.html).

Для передачи клиентского сертификата мы будем использовать файл в формате PKCS#12. В файлах PKCS#12 хранятся одновременно и закр ключ, и сертификат (разумеется в зашифрованном виде). Примерная организация PKCS#12 файла показана на рисунке.



Сконвертировать Ваш client.crt в файл формата PKCS#12 можно при помощи следующей команды:

openssl pkcs12 export in client.crt inkey client.key out client.p12

После того как мы получили файл в формате PKCS#12 можно переходить к разработке и тестированию нашего мобильного приложения. Нач IOS.

1. Реализуем IOS версию приложения

Необходимо подключить к Вашему проекту Security.Framework
Для осуществления запроса нам необходимо извлечеть из PKCS#12 цифровой сертификат и ассоциированный с ним приватный ключ (SecIdentityRef). Наличие данного объекта позволит нам получить соответствующий NSURLCredential.
Итак реализуем функцию extractIdentityAndTrust.

```
OSStatus extractIdentityAndTrust(CFDataRef inP12data, SecIdentityRef *identity)
{
    OSStatus securityError = errSecSuccess;

    CFStringRef password = CFSTR("");
    const void *keys[] = { kSecImportExportPassphrase };
```

```
    const void *values[] = { password };

    CFDictionaryRef options = CFDictionaryCreate(NULL, keys, values, 1, NULL, NULL);

    CFArrayRef items = CFArrayCreate(NULL, 0, 0, NULL);
    securityError = SecPKCS12Import(inP12data, options, &items);

    if (securityError == 0) {
        CFDictionaryRef myIdentityAndTrust = CFArrayGetValueAtIndex(items, 0);
        const void *tempIdentity = NULL;
        tempIdentity = CFDictionaryGetValue(myIdentityAndTrust, kSecImportItemIdentity);
        *identity = (SecIdentityRef)tempIdentity;
    }

    if (options) {
        CFRelease(options);
    }

    return securityError;
}
```

Производим извлечение при помощи функции SecPKCS12Import, незабываем указать пароль к серитификату.
Далее реализуем делегат canAuthenticateAgainstProtectionSpace, вызов данного делегата позволяет нам определить свойства сервера, а имен протокол, механизм авторизации. У нас реализация этого делегата будет простой, укажем, что обрабатываем любой способ аутентификации представленный сервером.

```
- (BOOL)connection:(NSURLConnection *)connection canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace *)protectionSpace
{
    return YES;
}
```

Обработаем возможные ошибки:

```
- (void)connection:(NSURLConnection*) connection didFailWithError:(NSError *)error
{
    NSLog(@"Did recieve error: %@", [error localizedDescription]);
    NSLog(@"%@", [error userInfo]);
}
```

Теперь перейдем к реализации непосредственно механизма аутентификации. Реализуем делегат didRecieveAuthentificationChallenge:

```
- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
{
    NSLog(@"Authentication challenge");

    // load cert
    NSString *path = [[NSBundle mainBundle] pathForResource:@"keystore" ofType:@"p12"];
    NSData *p12data = [NSData dataWithContentsOfFile:path];
    CFDataRef inP12data = (__bridge CFDataRef)p12data;

    SecIdentityRef myIdentity;
    OSStatus status = extractIdentityAndTrust(inP12data, &myIdentity);

    SecCertificateRef myCertificate;
    SecIdentityCopyCertificate(myIdentity, &myCertificate);
    const void *certs[] = { myCertificate };
    CFArrayRef certsArray = CFArrayCreate(NULL, certs, 1, NULL);

    NSURLCredential *credential = [NSURLCredential credentialWithIdentity:myIdentity certificates:(__bridge NSArray*)certsArray persistence: NS
edentialPersistenceForSession];

    [[challenge sender] useCredential:credential forAuthenticationChallenge:challenge];
}
```

Загружаем наш сертификат, извлекаем из него нужные нам данные, создаем NSURLCredential, передаем нужную информацию, сохраняем дан аутентификации только в контексте текущей сессии.

Ну и для полноты картины приведу код подготавливающий NSURLConnection:

```
NSString *key = @"test";

    NSError *jsonSerializationError = nil;
    NSMutableDictionary *projectDictionary = [NSMutableDictionary dictionaryWithCapacity:1];
    [projectDictionary setObject:key forKey:@"test"];

    NSData *jsonData = [NSJSONSerialization dataWithJSONObject:projectDictionary options:nil error:&jsonSerializationError];

    NSURL *requestUrl = [[NSURL alloc] initWithString:@"https://your_service"];
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:requestUrl cachePolicy:NSURLRequestReloadIgnoringCacheData timeoutInterv
0.0];
    [request setHTTPMethod:@"POST"];
    [request setValue:@"UTF-8" forHTTPHeaderField:@"content-charset"];
    [request setValue:@"application/json" forHTTPHeaderField:@"Accept"];
    [request setValue:@"application/json" forHTTPHeaderField:@"Content-Type"];
    [request setValue:[NSString stringWithFormat:@"%d", [jsonData length]] forHTTPHeaderField:@"Content-Length"];
    [request setHTTPBody: jsonData];
    NSURLConnection *connection = [[NSURLConnection alloc] initWithRequest:request delegate:self];
    [connection start];
```

Реализацию делегата didReceiveData приводить не буду.

2. Реализуем Android версию приложения

Начну сразу с кода:

```
KeyStore keystore = KeyStore.getInstance("PKCS12");

keystore.load(getResources().openRawResource(R.raw.keystore), "".toCharArray());

SSLSocketFactory sslSocketFactory = new AdditionalKeyStoresSSLSocketFactory(keystore);

HttpParams params = new BasicHttpParams();
HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
HttpProtocolParams.setContentCharset(params, HTTP.UTF_8);
HttpProtocolParams.setUseExpectContinue(params, true);

final SchemeRegistry registry = new SchemeRegistry();
registry.register(new Scheme("http", PlainSocketFactory.getSocketFactory(), 80));
registry.register(new Scheme("https", sslSocketFactory, 3123));

ThreadSafeClientConnManager manager = new ThreadSafeClientConnManager(params, registry);
DefaultHttpClient httpclient = new DefaultHttpClient(manager, params);

HttpPost httpPostRequest = new HttpPost("https://your_service");

// datas – array which contains data to send to server
StringEntity se = new StringEntity(datas[0].toString(), HTTP.UTF_8);

// Set HTTP parameters
httpPostRequest.setEntity(se);
httpPostRequest.setHeader("Accept", "application/json");
httpPostRequest.setHeader("Content-Type", "application/json");

HttpResponse response = httpclient.execute(httpPostRequest);
```

Получаем экземпляр соответствующего KeyStore в нашем случае это (PKCS12), загружаем из ресурсов наш сертификат, вторым аргументом указываем пароль. Далее создаем экземпляр SSLSocketFactory, использую собственную реализацию SSLSocketFactory, позволяющую инициализировать SSL контекст с использованием нашего сертификата. Код фабрики приведен чуть ниже. Далее конфигурируем параметры подключения, регистрируем нашу фабрику, указываем порт на который будем посылать запрос, формируем соответсвующий POST и выполня запрос.

Код фабрики:

```
import java.io.IOException;
import java.net.Socket;
```

```
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.Arrays;

import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;
import javax.net.ssl.X509TrustManager;

import org.apache.http.conn.ssl.SSLSocketFactory;

/**
 * Allows you to trust certificates from additional KeyStores in addition to
 * the default KeyStore
 */
public class AdditionalKeyStoresSSLSocketFactory extends SSLSocketFactory {
    protected SSLContext sslContext = SSLContext.getInstance("TLS");

    public AdditionalKeyStoresSSLSocketFactory(KeyStore keyStore) throws NoSuchAlgorithmException, KeyManagementException, KeyStoreException, U
nrecoverableKeyException {
        super(null, null, null, null, null, null);
        KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
                kmf.init(keyStore, "".toCharArray());
        sslContext.init(kmf.getKeyManagers(), new TrustManager[]{new ClientKeyStoresTrustManager(keyStore)}, new SecureRandom());
    }

    @Override
    public Socket createSocket(Socket socket, String host, int port, boolean autoClose) throws IOException {
        return sslContext.getSocketFactory().createSocket(socket, host, port, autoClose);
    }

    @Override
    public Socket createSocket() throws IOException {
        return sslContext.getSocketFactory().createSocket();
    }

    /**
     * Based on http://download.oracle.com/javase/1.5.0/docs/guide/security/jsse/JSSERefGuide.html#X509TrustManager
     */
    public static class ClientKeyStoresTrustManager implements X509TrustManager {

        protected ArrayList<X509TrustManager> x509TrustManagers = new ArrayList<X509TrustManager>();

        protected ClientKeyStoresTrustManager(KeyStore... additionalkeyStores) {
            final ArrayList<TrustManagerFactory> factories = new ArrayList<TrustManagerFactory>();

            try {
                // The default Trustmanager with default keystore
                final TrustManagerFactory original = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
                original.init((KeyStore) null);
                factories.add(original);

                for ( KeyStore keyStore : additionalkeyStores ) {
                    final TrustManagerFactory additionalCerts = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
                    additionalCerts.init(keyStore);
                    factories.add(additionalCerts);
                }

            } catch (Exception e) {
                throw new RuntimeException(e);
            }

            /*
```

```
             * Iterate over the returned trustmanagers, and hold on
             * to any that are X509TrustManagers
             */
            for (TrustManagerFactory tmf : factories)
                for ( TrustManager tm : tmf.getTrustManagers() )
                    if (tm instanceof X509TrustManager)
                        x509TrustManagers.add( (X509TrustManager) tm );

            if ( x509TrustManagers.size() == 0 )
                throw new RuntimeException("Couldn't find any X509TrustManagers");

        }

        public void checkClientTrusted(X509Certificate[] chain, String authType) throws CertificateException {
            for ( X509TrustManager tm : x509TrustManagers ) {
                try {
                    tm.checkClientTrusted(chain, authType);
                    return;
                } catch ( CertificateException e ) {

                }
            }
            throw new CertificateException();
        }

        /*
         * Loop over the trustmanagers until we find one that accepts our server
         */
        public void checkServerTrusted(X509Certificate[] chain, String authType) throws CertificateException {
            for ( X509TrustManager tm : x509TrustManagers ) {
                try {
                    tm.checkServerTrusted(chain, authType);
                    return;
                } catch ( CertificateException e ) {

                }
            }
            throw new CertificateException();
        }

        public X509Certificate[] getAcceptedIssuers() {
            final ArrayList<X509Certificate> list = new ArrayList<X509Certificate>();
            for ( X509TrustManager tm : x509TrustManagers )
                list.addAll(Arrays.asList(tm.getAcceptedIssuers()));
            return list.toArray(new X509Certificate[list.size()]);
        }
    }

}
```

Заключение.

Мы рассмотрели как производить авторизацию по SSL с использованием клиентского сертификата.

Полезная информация:
Certificate, Key, and Trust Services Tasks for iOS
Описание PKCS12
Creating .NET web service with client certificate authentication
Certificate Authentication in asp.net
Java 2-way TLS/SSL (Client Certificates) and PKCS12 vs JKS KeyStores

Спасибо за внимание!

ssl, p12, iOS, android

ПОХОЖИЕ ПУБЛИКАЦИИ

9 февраля 2015 в 08:59
Анонс конференции Mobius 2015: доклады по iOS, Android и Mobile Security
+16 6k 44 0

21 марта 2014 в 16:05
Билеты на AppSummit — iOS, Android, Windows и бизнес на приложениях
+24 3,7k 7 22

27 января 2014 в 11:56
Реализация L2TP/IPsec VPN сервера стандартными средствами Windows 7/8 для подключения Windows/iOS/Android систем к внутренне сети
+14 147k 271 7

САМОЕ ЧИТАЕМОЕ Разраб

Сутки | Неделя | Месяц

Сортировка пузырьком в коде Qualcomm
+71 18,2k 36 77

VAX — инструмент для визуального программирования, или как написать SQL мышкой
+41 11,4k 139 43

Пусть интернет прогнётся под нас
+17 10,5k 70 30

Особенности национальной SMS-авторизации
+41 10,7k 33 24

Критическая уязвимость в multisig кошельке Parity, хакерами выведен $31 миллион в ethereum (обновлено)
+30 17,6k 29 47

Комментарии (8)

НЛО прилетело и опубликовало эту надпись здесь

LrdSpr 22 апреля 2014 в 23:01

К сожалению со SCEP не сталкивался, ответить не могу. Может кто-то из коллег подскажет.

НЛО прилетело и опубликовало эту надпись здесь

LrdSpr 22 апреля 2014 в 23:32

По следующей ссылке коллеги рассматривают вопрос схожий с Вашим: stackoverflow.com/questions/5323686/ios-pre-install-ssl-certificate-in-keychain-programmatic
поводу SCEP, наткнулся на клиентскую библиотеку, которая возможно пригодится: github.com/microsec/MscSCEP

НЛО прилетело и опубликовало эту надпись здесь

Ghedeon 23 апреля 2014 в 11:15 *(комментарий был изменён)*

Есть ли какие-то идеи, как обезопасить себя от того, что есть «400 сравнительно честных способов» утянуть ваш сертификат из приложения?

AAM 23 апреля 2014 в 13:23

Данный недостаток перекрывает все достоинства описанного метода. Неужели кто-то действительно хранит сертификат/ключ в ресурсах приложения (счита
открытом виде)?

НЛО прилетело и опубликовало эту надпись здесь

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

ИНТЕРЕСНЫЕ ПУБЛИКАЦИИ

Возвращение арматурной легенды: наушники Etymotic ER4SR, ER4XR GT

+5 417 0 0

Британские спутниковые снимки 2: Как все было на самом деле

+24 1,5k 8 2

Децентрализованные цифровые валюты. Часть 1. Биткойн GT

+10 1,2k 15 2

Мониторинг работы производства веб-студии

+14 1,1k 11 0

Необходимость регулирования интернета вещей

+12 2,8k 4 12