SSL pinning para prevenir un ataque man-in-the-middle (MITM) en la aplicación Android / iOS

Muchos de ustedes probablemente han oído hablar de un ataque de hombre en el medio y se preguntaron qué tan difícil sería un ataque como ese. Para aquellos de ustedes que nunca han oído hablar de uno, es simplemente donde nosotros, el hacker, nos colocamos entre la víctima y el servidor y enviamos y recibimos toda la comunicación entre los dos.

Debe ser totalmente transparente tanto para el cliente como para el servidor sin sospechar que están conectados a nada ni a nadie más que a quienes esperan. Esto nos permite ver y leer toda la comunicación (contraseñas, información confidencial, etc.), así como modificarla si es necesario.

El arte de oler(Sniffing)

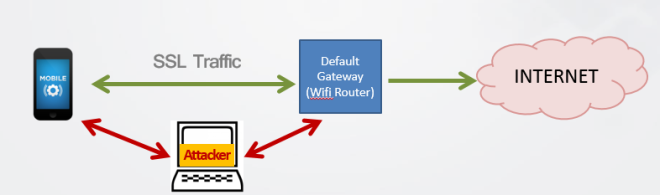
Antes de embarcarnos en un ataque MitM, debemos abordar algunos conceptos. Primero, olfatear es el acto de captar todo el tráfico que pasa por la comunicación por cable o inalámbrica. Hay una serie de herramientas que le permitirán hacer esto. El más famoso es Wireshark, pero también tcpdump, dsniff y algunos otros.

Entrar en modo promiscuo

Para ver y captar tráfico que no sea el suyo, primero debe poner su NIC o adaptador inalámbrico en modo promiscuo (llamado modo monitor en inalámbrico), lo que significa que captará TODO el tráfico, no solo el destinado a su MAC / Dirección IP. En redes inalámbricas y cableadas con concentradores, esto se puede lograr con relativa facilidad. En un entorno conmutado, debemos ser un poco más creativos.

Spoofing de ARP para un ataque MitM

Lo que haremos aquí es usar ARP spoofing para colocarnos entre dos máquinas, haciendo que el cliente crea que somos el servidor y el servidor crea que somos el cliente. Con esto, podemos enviar todo el tráfico a través de nuestra computadora y rastrear cada paquete que vaya en cualquier dirección.



Un ataque man-in-the-middle es aquel en el que el atacante intercepta y transmite en secreto mensajes entre dos partes que creen que se están comunicando directamente entre sí. Ataque utilizado para interceptar el tráfico entre el dispositivo de la víctima y la puerta de enlace El atacante debe estar en la misma red que la víctima El atacante puede usar ARP spoofing El atacante puede usar Kali Linux para este ataque (Attackers Machine) El atacante puede rastrear el tráfico usando SSLstrip y Ettercap en la máquina del atacante.

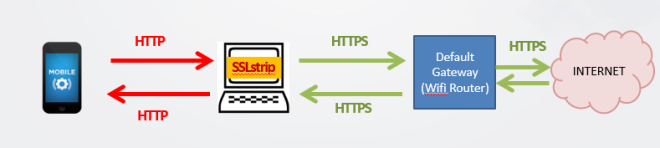
SSLStrip

SSLstrip es un tipo de ataque MITM que obliga al navegador de la víctima a comunicarse con un adversario en texto sin formato a través de HTTP.

El adversario se comunica con el servidor a través de SSL regular (HTTPS)

El servidor no conoce la diferencia

Para hacer esto, SSLstrip está "quitando" https: //URLs y convirtiéndolas en http: // URLs



Esperanza anterior Introducción a MiTM nos ayudará a avanzar y buscar posibles soluciones para proteger nuestra aplicación de MiTM Attack.

Analicemos el concepto general para prevenir el ataque MiTM en aplicaciones móviles SSL Pinning.

¿Qué es la Pinning SSL?

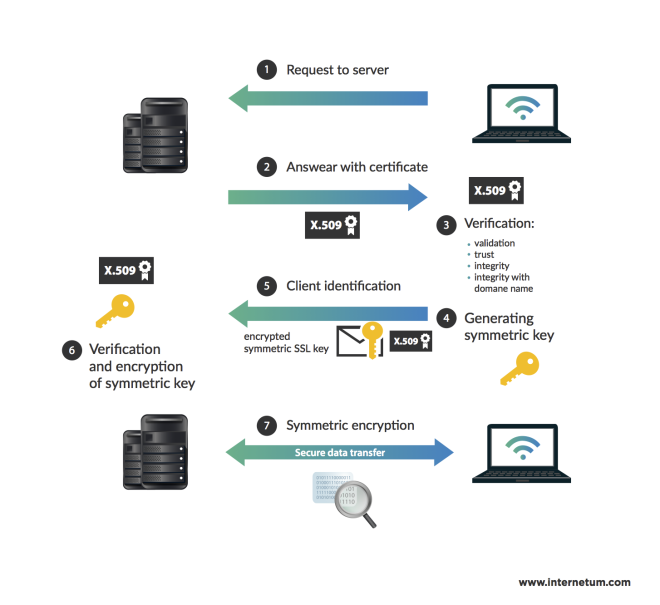
De forma predeterminada, al realizar una conexión SSL, un cliente comprueba que el certificado del servidor:

tiene una cadena de confianza verificable a un certificado de confianza (raíz) coincide con el nombre de host solicitado

Lo que no hace es comprobar que es tu certificado, el que subiste a tu servidor.

¿Cómo funciona HTTP Client con conexiones SSL y certificados SSL?

Vamos a entender cómo funcionará un cliente HTTP (clientes HTTP de aplicaciones de navegador / Android o iOS) con conexiones SSL y certificados SSL.



1. Http Client lo hará y solicitará al servidor una URL respectiva, es decir, https://wordpress.com.
2. El servidor responderá con un Certificado (Certificado X.509).
3. El Cliente Verificará el Certificado que lo hace El Certificado es emitido por una CA válida (Autoridad de Certificación) o no, si SÍ, habrá intercambio de clave simétrica en NO, habrá una excepción de reconocimiento (error) y la conexión se terminará.

¿Cómo verifico el certificado de mi servidor?

Puede utilizar el cliente OpenSSL para diversas tareas relacionadas con SSL:

para comprobar el certificado del servidor: openssl s\_client -showcerts -connect wordpress.com:443

para descargar el Certificado: openssl s\_client -showcerts -connect wordpress.com:443 | openssl x509 -outform PEM> wordpress\_certfile.pem

SSL pinning para evitar un ataque man-in-the-middle (MITM) en la aplicación Android / iOS



Implementar SSL pinning

Antes de escribir cualquier código, hay algunas decisiones que debe tomar desde el certificado que usa para fijar lo que hace cuando las cosas van mal. Así que comencemos hablando de los certificados.

Para abordar estas desventajas, los servidores generalmente se configuran con certificados de emisores conocidos llamados Autoridades de certificación (CA). La plataforma de host generalmente contiene una lista de CA conocidas en las que confía. A partir de Android 4.2 (Jelly Bean), Android contiene actualmente más de 100 CA que se actualizan en cada versión.

Similar a un servidor, una CA tiene un certificado y una clave privada. Al emitir un certificado para un servidor, la CA firma el certificado del servidor con su clave privada. Luego, el cliente puede verificar que el servidor tiene un certificado emitido por una CA conocida en la plataforma. Sin embargo, al resolver algunos problemas, el uso de CA presenta otro. Debido a que la CA emite certificados para muchos servidores, aún necesita alguna forma de asegurarse de que está hablando con el servidor que desea. Para solucionar esto, el certificado emitido por la CA identifica al servidor con un nombre específico como gmail.com o un conjunto de hosts con comodines como \* .google.com. El siguiente ejemplo hará que estos conceptos sean un poco más concretos. En el siguiente fragmento de una línea de comandos, el comando s\_client de la herramienta openssl examina la información del certificado del servidor de Wikipedia. Especifica el puerto 443 porque es el predeterminado para HTTPS. El comando envía la salida de openssl s\_client a openssl x509, que formatea la información sobre certificados de acuerdo con el estándar X.509. Específicamente, el comando solicita el asunto, que contiene la información del nombre del servidor, y el emisor, que identifica la CA.

|  |
| --- |
| *$ openssl s\_client -connect wikipedia.org:443 | openssl x509 -noout -subject -issuer*  ***subject=*** */serialNumber=sOrr2rKpMVP70Z6E9BT5reY008SJEdYv/C=US/O=\*.wikipedia.org/OU=GT03314600/OU=See www.rapidssl.com/resources/cps (c)11/OU=Domain Control Validated - RapidSSL(R)/****CN=\*.wikipedia.org***  ***issuer=*** */C=US/O=GeoTrust, Inc./CN=****RapidSSL CA*** |

Puede ver que el certificado fue emitido para servidores que coinciden con \* .wikipedia.org por RapidSSL CA.

Los certificados raíz vienen preinstalados en los dispositivos Android con alrededor de 150 incluidos en Android N. Puede verificar qué hay en su propio dispositivo yendo a Configuración> Seguridad> Credenciales confiables. Se asume que ninguna de estas CA raíz o las miles de CA intermedias en las que confían estos certificados raíz emitirán certificados hoja para nombres de dominio que no deberían. Si no me cree, lea acerca de las CA DigiNotar, GlobalSign y Comodo. Además de todo esto, el dispositivo del usuario podría verse comprometido con un certificado falso instalado a través de la ingeniería social. La fijación de SSL, también conocida como fijación de clave pública, es un intento de resolver estos problemas, asegurando que la cadena de certificados utilizada sea la que espera su aplicación al verificar que aparece una clave pública o un certificado en la cadena. Certificado intermedio.

Al anclar contra el certificado intermedio, confía en que la autoridad certificadora intermedia no emitirá incorrectamente un certificado para su servidor (s). Esto también tiene la ventaja de que siempre que se ciña al mismo proveedor de certificados, cualquier cambio en sus certificados hoja funcionará sin tener que actualizar su aplicación. Certificado raíz. Al anclar el certificado raíz, confía en la autoridad del certificado raíz, así como en los intermediarios en los que confían para no emitir certificados incorrectamente. A menudo, las autoridades raíz e intermedias son la misma empresa, en cuyo caso no hay mucha diferencia en la cantidad de personas en las que confía; sin embargo, no siempre es así. No es necesario que se fije en un solo certificado de la cadena. De hecho, la recomendación general es fijar varios niveles para disminuir las posibilidades de bloquear su aplicación a expensas de confiar en más emisores de certificados. En mi opinión, parecería prudente fijar en los niveles intermedio y foliar para dar un equilibrio sensato.

¿Certificado o fijación de clave pública? En la documentación de entrenamiento de Android sobre cómo anclar sus pines de ejemplo contra el certificado. Sin embargo, a menudo es mejor anclar contra la clave pública, o más específicamente el SubjectPublicKeyInfo (SPKI).

"Algunos sitios web rotan sus certificados mensualmente y, por lo tanto, fijarlos en un certificado significaría una aplicación que nadie puede usar después de un mes o realizar actualizaciones frecuentes". “Normalmente, aunque la clave pública dentro de estos certificados rotados permanece igual. Al fijar contra la clave, está reduciendo las posibilidades de bloquear su aplicación al limitar los valores que están marcados " Con la mayoría de las API de red, puede optar por anclar contra el certificado o el SPKI, aunque normalmente es mucho más fácil de anclar utilizando el SPKI ya que la mayoría de las API proporcionan un mecanismo integrado, como el CertificatePinner de OkHttp. Con HttpUrlConnection no hay mucha diferencia en el costo de desarrollo, así que para mantener la coherencia, he mostrado el código utilizando SPKI.

Compromiso de manejo

Si los pines almacenados en su aplicación no coinciden con los devueltos por la conexión HTTPS, ¿cómo informa al usuario? Fracasar duro. Detenga la aplicación para que no establezca la conexión. Este es el más seguro y fácil de implementar, pero presenta la posibilidad de problemas de negación de servicio y experiencia del usuario autoinducidos cuando no se pueden establecer conexiones.

Para implementar eso, podemos buscar en el enlace de GitHub de Moxie Marlinspike para AndroidPinning.

Cree un PinningSSLSocketFactory donde usamos PinningTrustManager para administrar la verificación de PKI.

package com.example.myapplication;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.os.Bundle;  
  
import com.android.volley.Request;  
import com.android.volley.RequestQueue;  
import com.android.volley.Response;  
import com.android.volley.VolleyError;  
import com.android.volley.toolbox.HurlStack;  
import com.android.volley.toolbox.JsonObjectRequest;  
import com.android.volley.toolbox.Volley;  
  
import org.json.JSONException;  
import org.json.JSONObject;  
  
import java.io.BufferedInputStream;  
import java.io.IOException;  
import java.io.InputStream;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.security.KeyManagementException;  
import java.security.KeyStore;  
import java.security.KeyStoreException;  
import java.security.NoSuchAlgorithmException;  
import java.security.cert.Certificate;  
import java.security.cert.CertificateException;  
import java.security.cert.CertificateExpiredException;  
import java.security.cert.CertificateFactory;  
import java.security.cert.CertificateNotYetValidException;  
import java.security.cert.X509Certificate;  
import java.util.HashMap;  
import java.util.Map;  
  
import javax.net.ssl.HostnameVerifier;  
import javax.net.ssl.HttpsURLConnection;  
import javax.net.ssl.SSLContext;  
import javax.net.ssl.SSLSession;  
import javax.net.ssl.SSLSocketFactory;  
import javax.net.ssl.TrustManager;  
import javax.net.ssl.TrustManagerFactory;  
import javax.net.ssl.X509TrustManager;  
  
public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
 //------------------------------------------------------------------------------------------  
 CertificateFactory cf = null;  
 try {  
 cf = CertificateFactory.*getInstance*("X.509");  
  
 // Generate the certificate using the certificate file under res/raw/cert.cer  
 InputStream caInput = new BufferedInputStream(getResources().openRawResource(R.raw.*cert\_agile*));  
 Certificate ca = cf.generateCertificate(caInput);  
 caInput.close();  
  
 // Create a KeyStore containing our trusted CAs  
 String keyStoreType = KeyStore.*getDefaultType*();  
 KeyStore trusted = KeyStore.*getInstance*(keyStoreType);  
 trusted.load(null, null);  
 trusted.setCertificateEntry("ca", ca);  
  
 // Create a TrustManager that trusts the CAs in our KeyStore  
 String tmfAlgorithm = TrustManagerFactory.*getDefaultAlgorithm*();  
 TrustManagerFactory tmf = TrustManagerFactory.*getInstance*(tmfAlgorithm);  
 tmf.init(trusted);  
  
 // Create an SSLContext that uses our TrustManager  
 SSLContext context = SSLContext.*getInstance*("TLS");  
 context.init(null, tmf.getTrustManagers(), null);  
  
 SSLSocketFactory sf = context.getSocketFactory();  
  
//------------------------------------------------------------------------------------------  
 RequestQueue queue = Volley.*newRequestQueue*(this, new HurlStack(null, sf));  
  
  
  
 Map<String, String> params = new HashMap<String, String>();  
 //params.put("d", "data test");  
 final String[] data = {""};  
 JSONObject jsonObj = new JSONObject(params);  
  
  
  
 JsonObjectRequest jsonObjRequest = new JsonObjectRequest(  
 Request.Method.*GET*,  
 "https://consultaruc.agilecorp.net.pe/api/v1/dni/41544650?token=abcxyz",  
 jsonObj, new Response.Listener<JSONObject>() {  
 @Override  
 public void onResponse(JSONObject response) {  
 data[0] = response.toString();  
 }  
 },  
 new Response.ErrorListener() {  
 @Override  
 public void onErrorResponse(VolleyError error) {  
 String err = error.toString();  
 }  
 });  
 queue.add(jsonObjRequest);  
  
 } catch (CertificateException | IOException | KeyStoreException | NoSuchAlgorithmException | KeyManagementException e) {  
 e.printStackTrace();  
 }  
  
  
  
  
  
  
  
  
  
 }  
  
  
  
  
  
  
}