Lesson 4 – Insecure Communication

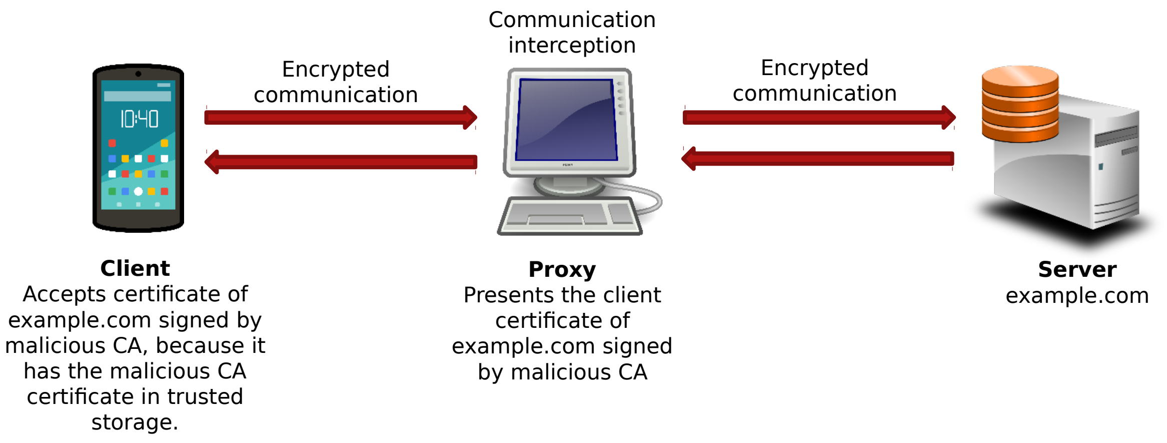
Introduction

This lesson focuses on finding and exploiting security issues in Android applications that are caused by insecure communication. Communication is an important attack surface of mobile applications security since mobile devices are often used in many places and connected to different, possibly malicious wireless networks.

The basic way to secure HTTP connection is the use of SSL/TLS protocol. If implemented correctly it guarantees confidentiality and integrity of communication and authenticates the server. Some implementations of SSL/TLS can also authenticate the user using the client certificates, but it is not a commonly used feature.

The authentication relies on the server certificate signed by the trusted Certificate Authority. The Public Key Infrastructure is used for this and the root CA certificates are embedded in the device software. The certificate that the server represents itself with must be in the chain of trust that starts with one of the root CA certificates trusted by the device in order for the authentication to be performed successfully.

To analyze the application we need to inspect and intercept the traffic. This can be done by performing a Man in the Middle attack that puts the intercepting proxy in between of the client – server communication. In order to do this you need to put the root CA generated by intercepting proxy in the device trusted certificates storage. By doing this, the device will trust the certificates generated by the intercepting proxy for visited sites and it will allow for the connection to be established. Installing of root CA to analyze the traffic is commonly used by AV software and some of the firewalls.


Communication interception
Encrypted communication
Encrypted communication
10:40
Client
Accepts certificate of example.com signed by malicious CA, because it has the malicious CA certificate in trusted storage.
Proxy
Presents the client certificate of example.com signed by malicious CA
Server
example.com

Some of the mobile applications are protected against such attacks by using Certificate Pinning[1] – additional checks for the server's certificate. It protects against malicious entity that has inserted its certificate into device trusted storage but it also provides protection against Reverse Engineering of the application since the mechanism has to be bypassed in order to inspect application traffic.

Getting started

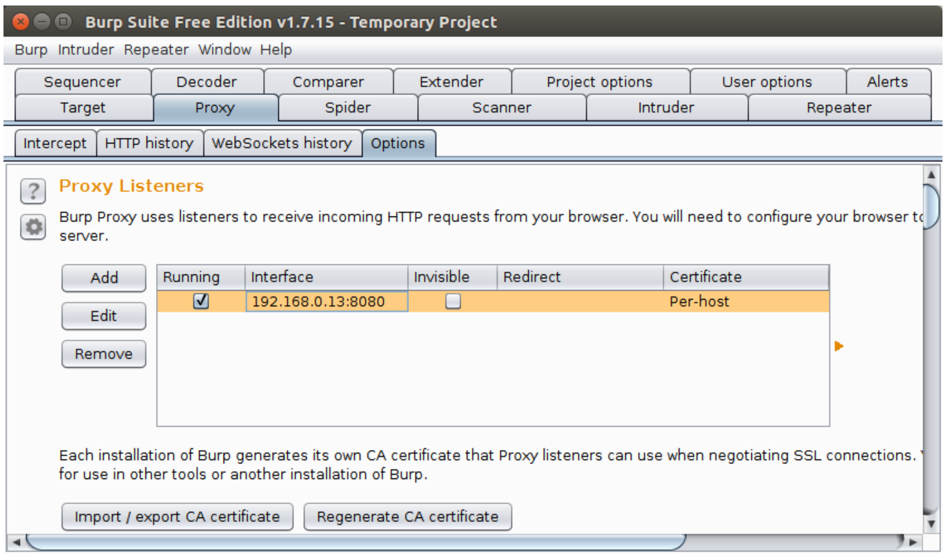
You will need:

- a device or emulator with Android , no USB debugging or root access needed,
- computer with intercepting proxy (Burp Suite[2], OWASP ZAP[3]) installed.

1 https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning
2 https://portswigger.net/burp/download.html
3 https://github.com/zaproxy/zaproxy/wiki/Downloads

There is free version of Burp Suite available online. It has limited functionality, but it's enough for basic traffic inspection.

In order for the communication interception to work with physical device, the device and computer must be on the same network. Start your proxy and select your computer IP address (or 127.0.0.1 for emulator) from the drop-down menu and put in a port number:


Burp Suite Free Edition v1.7.15 - Temporary Project
Burp Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User options Alerts
Target Proxy Spider Scanner Intruder Repeater
Intercept HTTP history WebSockets history Options
Add
Edit
Remove
Import / export CA certificate
Regenerate CA certificate

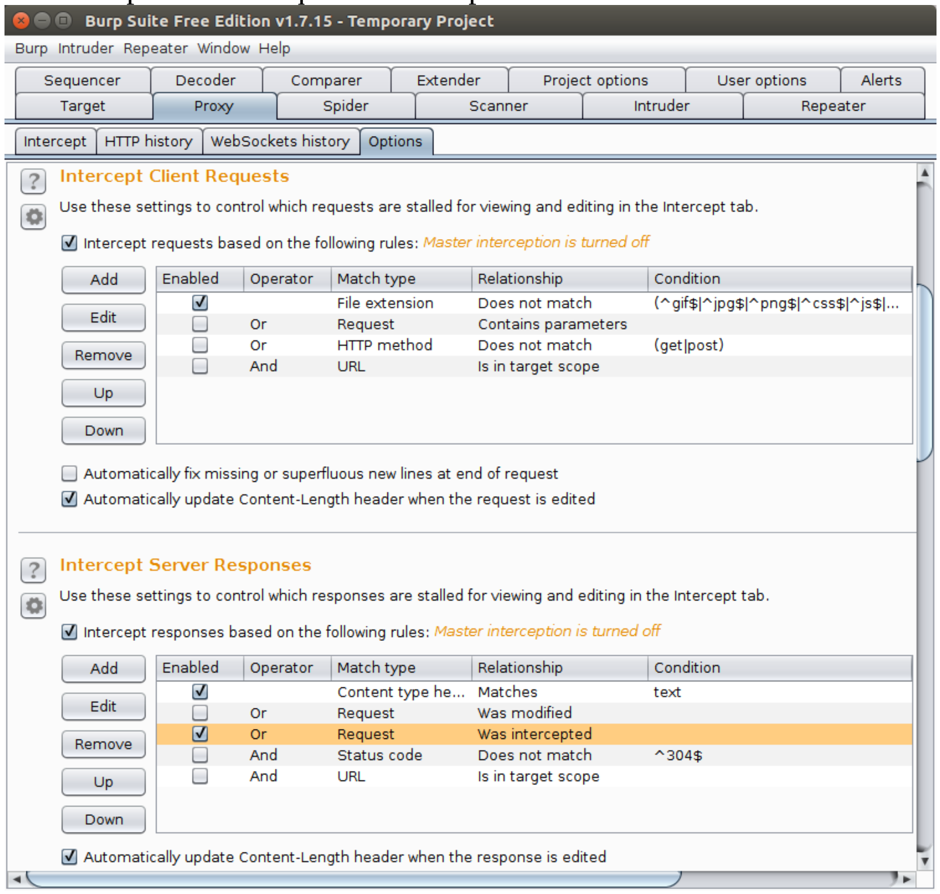
Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser t
server.

Running	Interface	Invisible	Redirect	Certificate
☑	192.168.0.13:8080	☐		Per-host

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating SSL connections.
for use in other tools or another installation of Burp.

Set the proxy to intercept both on requests and responses:


Burp Suite Free Edition v1.7.15 - Temporary Project
Burp Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User options Alerts
Target Proxy Spider Scanner Intruder Repeater
Intercept HTTP history WebSockets history Options
Add
Edit
Remove
Up
Down

Intercept Client Requests

Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

☑ Intercept requests based on the following rules: *Master interception is turned off*

Enabled	Operator	Match type	Relationship	Condition
☑		File extension	Does not match	(^gif$\|^jpg$|^png$\|^css$|^js$|...
☐	Or	Request	Contains parameters	
☐	Or	HTTP method	Does not match	(get|post)
☐	And	URL	Is in target scope	

☐ Automatically fix missing or superfluous new lines at end of request
☑ Automatically update Content-Length header when the request is edited

Intercept Server Responses

Use these settings to control which responses are stalled for viewing and editing in the Intercept tab.

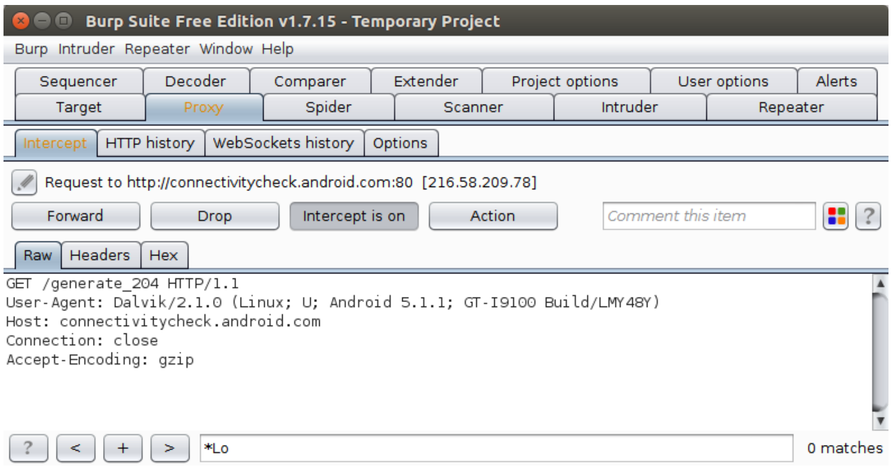
☑ Intercept responses based on the following rules: *Master interception is turned off*

Enabled	Operator	Match type	Relationship	Condition
☑		Content type he...	Matches	text
☐	Or	Request	Was modified	
☑	Or	Request	Was intercepted	
☐	And	Status code	Does not match	^304$
☐	And	URL	Is in target scope	

☑ Automatically update Content-Length header when the response is edited

Go to intercept tab and turn Intercept on.

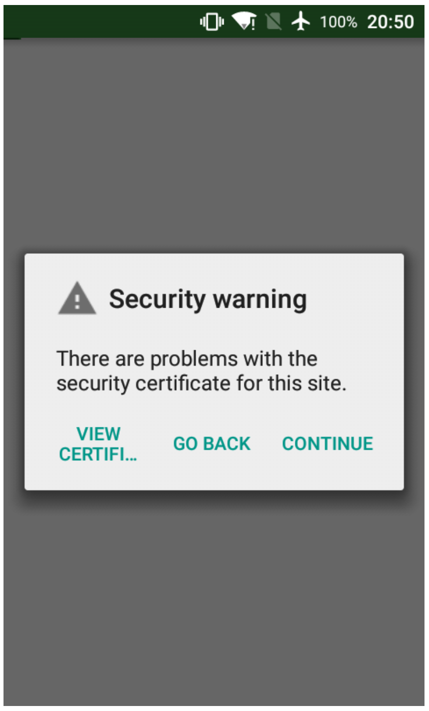
Now, go to the WiFi settings on your Android device. Long press the WiFi network you are connected to and tap Modify. Set the same values for proxy IP address and port number as you did in Burp. After setting this, look at the Burp Intercept tab:


Burp Suite Free Edition v1.7.15 - Temporary Project
Burp Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User options Alerts
Target Proxy Spider Scanner Intruder Repeater
Intercept HTTP history WebSockets history Options
Request to http://connectivitycheck.android.com:80 [216.58.209.78]
Forward Drop Intercept is on Action
Comment this item
Raw Headers Hex
*Lo
0 matches

```
GET /generate_204 HTTP/1.1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 5.1.1; GT-I9100 Build/LMY48Y)
Host: connectivitycheck.android.com
Connection: close
Accept-Encoding: gzip
```

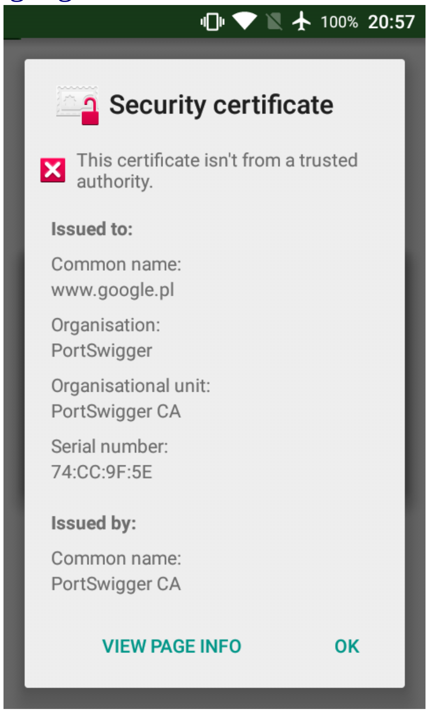
You should see a packet being intercepted. It will probably be the packet sent to http://connectivitycheck.android.com in order to check the connectivity to the Internet. Press Forward the packet. The next packet showing up on the screen will be the response to this request. You can turn the interception off, the history of the traffic will still be available in HTTP history tab until you exit Burp.

Now start the web browser and try to open https://www.google.com:



Security warning

There are problems with the security certificate for this site.



Security certificate

This certificate isn't from a trusted authority.

Issued to:

Common name: www.google.pl

Organisation: PortSwigger

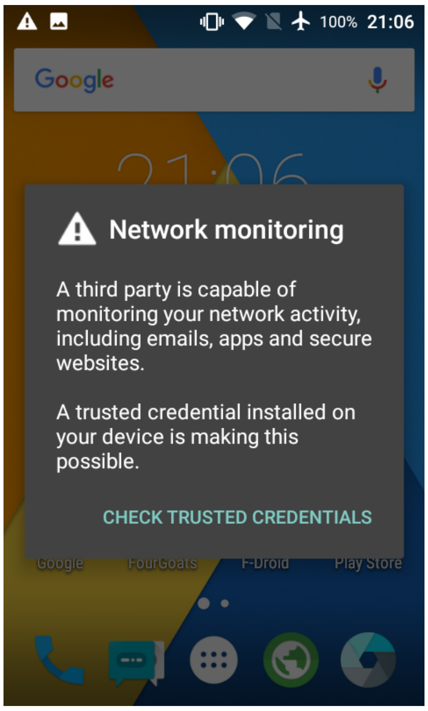
Organisational unit: PortSwigger CA

Serial number: 74:CC:9F:5E

Issued by:

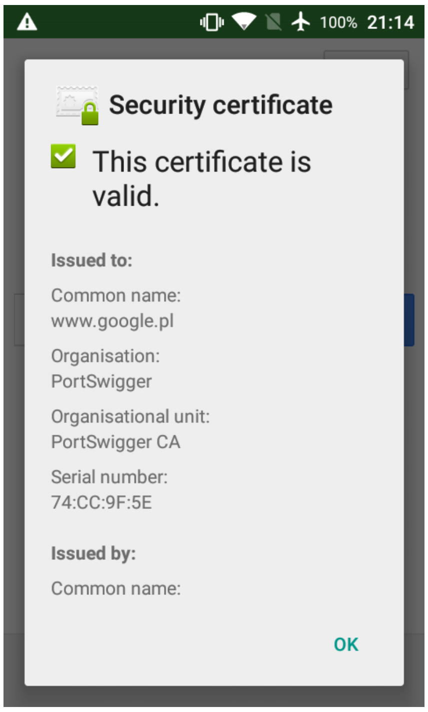
Common name: PortSwigger CA

You did not install the CA certificate generated by Burp, so the certificate is not considered trusted. In order to fix it, visit http://burp and click CA Certificate. Go to Android Settings - > Security → Install from SD card and tap on the downloaded certificate to install it. If it does not work, you need to change the file extensions from *.der to *.cer. After the certificate installation a notification appears. This is a security feature that is in Android since KitKat version. It warns user about the dangers of having third party certificate installed.
Now try to visit google webpage again. It loads correctly. Find an option in your browser to view the certificate.


Network monitoring

A third party is capable of monitoring your network activity, including emails, apps and secure websites.

A trusted credential installed on your device is making this possible.


Security certificate

This certificate is valid.

Issued to:

Common name:
www.google.pl

Organisation:
PortSwigger

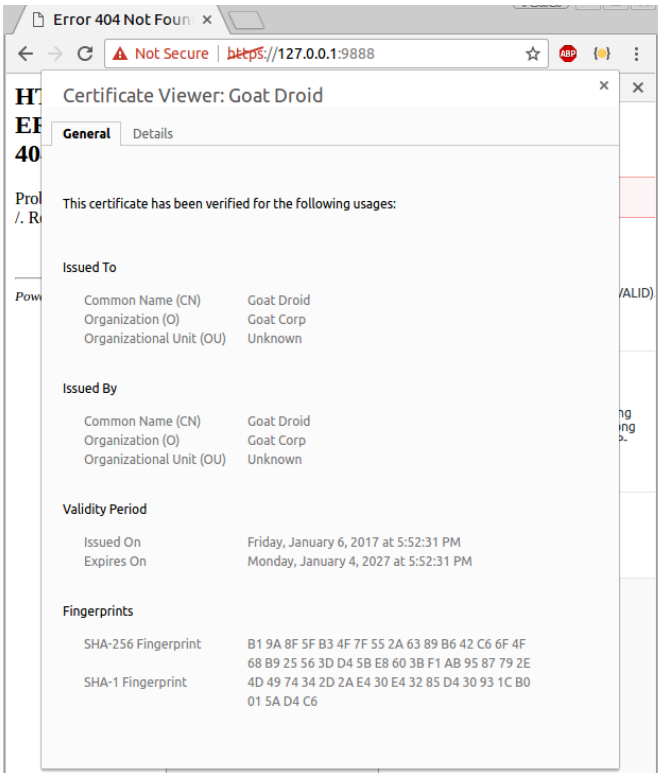
Organisational unit:
PortSwigger CA

Serial number:
74:CC:9F:5E

Issued by:

Common name:

The web browser usually allow you to accept the risk and open a website with incorrect certificate. It is because some of the websites may not have correctly configured SSL/TLS. However a mobile application with good implemented SSL/TLS should refuse to make a connection in such situation. Incorrectly developed application may accept the certificate or even fall back to unencrypted HTTP communication.
After viewing the certificate in the browser you can see that google.com presents a certificate signed by PortSwigger CA – this is the Burp CA installed on the device. Every time you connect to a server Burp creates a certificate for the given server, signs it by the Burp CA and presents it to the client.



Certificate Viewer: Goat Droid

This certificate has been verified for the following usages:

Issued To

Common Name (CN)	Goat Droid
Organization (O)	Goat Corp
Organizational Unit (OU)	Unknown

Issued By

Common Name (CN)	Goat Droid
Organization (O)	Goat Corp
Organizational Unit (OU)	Unknown

Validity Period

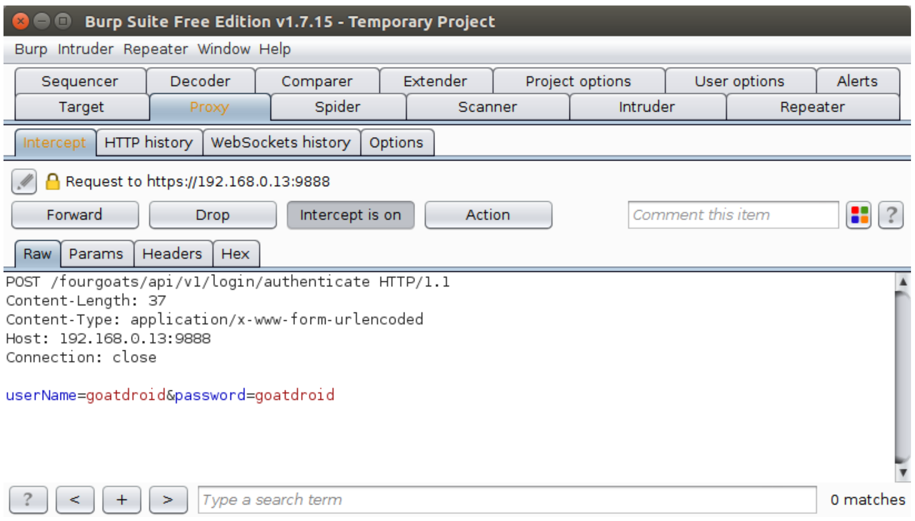
Issued On	Friday, January 6, 2017 at 5:52:31 PM
Expires On	Monday, January 4, 2027 at 5:52:31 PM

Fingerprints

SHA-256 Fingerprint	B1 9A 8F 5F B3 4F 7F 55 2A 63 89 B6 42 C6 6F 4F 68 B9 25 56 3D D4 5B E8 60 3B F1 AB 95 87 79 2E
SHA-1 Fingerprint	4D 49 74 34 2D 2A E4 30 E4 32 85 D4 30 93 1C B0 01 5A D4 C6

Finding vulnerabilities

Start with removal of the Burp certificate from Android device storage. We want to check how the application behaves, when the incorrect certificate is presented. Configure the environment in the way described above and start the application. Put Intercept on in Burp. Input the username and password and log in.


Burp Suite Free Edition v1.7.15 - Temporary Project
Burp Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User options Alerts
Target Proxy Spider Scanner Intruder Repeater
Intercept HTTP history WebSockets history Options
Request to https://192.168.0.13:9888
Forward Drop Intercept is on Action
Comment this item
Raw Params Headers Hex
Type a search term
0 matches

```
POST /fourgoats/api/v1/login/authenticate HTTP/1.1
Content-Length: 37
Content-Type: application/x-www-form-urlencoded
Host: 192.168.0.13:9888
Connection: close

userName=goatdroid&password=goatdroid
```

Burp managed to decrypt the credentials sent to the server. It means that the application doesn't handle correctly the SSL/TLS and it was possible to intercept the trafic even without the Burp CA certificate installed.

This is because of the code in class CustomSSLSocketFactory which implements trust for all certificates. It was implemented because the certificate that server uses is self-signed:

The code of CustomSSLSocketFactory class:

```
//Inspired    from    here:    http://stackoverflow.com/questions/2642777/trusting-all-
certificates-using-httpclient-over-https
public class CustomSSLSocketFactory extends SSLSocketFactory {
	SSLContext sslContext = SSLContext.getInstance("TLS");

	public CustomSSLSocketFactory(KeyStore truststore)
			throws NoSuchAlgorithmException, KeyManagementException,
			KeyStoreException, UnrecoverableKeyException {
		super(truststore);

		TrustManager tm = new X509TrustManager() {

			...

			@Override
			public void checkServerTrusted(
					java.security.cert.X509Certificate[]   chain,   String
authType)
					throws java.security.cert.CertificateException {
				// TODO Auto-generated method stub

			}
		};

		sslContext.init(null, new TrustManager[] { tm }, null);
	}

	...
	public static HttpClient getNewHttpClient() {
		try {
			KeyStore trustStore = KeyStore.getInstance(KeyStore
					.getDefaultType());
			trustStore.load(null, null);
			SSLSocketFactory sf = new CustomSSLSocketFactory(trustStore);
			sf.setHostnameVerifier(SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
			HttpParams params = new BasicHttpParams();
			SchemeRegistry registry = new SchemeRegistry();
			registry.register(new Scheme("http", PlainSocketFactory
					.getSocketFactory(), 80));
			registry.register(new Scheme("https", sf, 443));

			ClientConnectionManager ccm = new ThreadSafeClientConnManager(
					params, registry);

			return new DefaultHttpClient(ccm, params);
		} catch (Exception e) {
			return new DefaultHttpClient();
		}
	}
}
```

The method that checks if the server is trusted is empty and does not check anything. The hostname verifier is set to trust all hostnames.

Fixing the vulnerability

The solution that trusts all certificates was used because the server certificate is self-signed. This kind of vulnerabilities often occur when the server certificate in test environment is self-signed and the code is not changed before deploying to production environment.

There would not be any need to write custom SSLSocketFactory, if the server certificate would be signed by a known and trusted CA. In such case the standard Android SSL Socket Factory would work correctly and throw an exception in case of invalid server certificate.

The certificates signed by known CA used to be expensive and difficult to get, but that is not the case anymore. Let’s Encrypt[4] is an non-profit organization that issues free certificates in an automated way.

Using a well known CA is not always possible, especially in intranet only applications. In this case the best solution would be to generate certificate for your own CA and make the users install this CA certificate on their device. This is the approach commonly used inside organizations which sign the certificates of internal application servers with its own CA and distribute the CA certificate with MDM (Mobile Device Managment) system.

You can use OpenSSL to generate the CA certificate, server certificate and to sign the server certificate. Start with generating private key of the CA and encrypt it with DES3:

```
openssl genrsa -des3 -out rootCA.key 2048
```

This file has to be protected with a strong password and stored in a safe place. An attacker who would gain the access to the CA private key could use it to capture HTTPS traffic on devices that trust this CA certificate.
Then self-sign this certificate. You can change the validity of period for this certificate by modifying the days number:

```
openssl req -x509 -new -nodes -key CA.key -sha256 -days 1024 -out CA.pem
```

4 https://letsencrypt.org/

Now you have the PEM file with the CA certificate that needs to be installed on the devices.

Generate the private key for the server certificate:
```
openssl genrsa -des3 -out server.key 2048
```

Generate the certificate signing request (csr file), so the CA can sign it. You will be asked about the Common Name. This is a very important question since the validity of this certificate will be decided based upon the correct value of this certificate field. It should be a static IP address or ideally a FQDN (Fully Qualified Domain Name) of the server.
```
openssl req -new -key server.key -out server.csr
```

Now you can sign the server's certificate with the CA private key:
```
openssl x509 -req -in server.csr -CA CA.pem -CAkey CA.key -CAcreateserial -out
server.crt -days 500 -sha256
```

Now you need to put both private key (server.key) and certificate (server.crt) in one PKCS12 file:
```
openssl pkcs12 -inkey server.key -in server.crt -export -out server.pkcs12
```

Import the PKCS12 file to Java Key Store – a format used by our Jetty server. For export password use "goatdroid" a password used by a previous keystore:
```
keytool -importkeystore -srckeystore server.pkcs12 -srcstoretype PKCS12
-destkeystore keystore
```

Replace the current keystore file with a newly generated one. And start the server. Everything should work now.