SSL/TLS Certificate Pinning Bypass

`SSL/TLS certificate pinning` is a common technique used by developers to enhance the security of mobile applications.It ensures that the app communicates only with a specific, trusted server by verifying the server's certificate against a known copy embedded in the app. However, if improperly implemented or combined with other vulnerabilities, this security feature may create a false sense of protection. Understanding how to bypass SSL pinning is essential for penetration testers evaluating the security of an application's network communication.

SSL (Secure Sockets Layer) pinning—more accurately referred to as `TLS (Transport Layer Security)` pinning, as SSL is now deprecated—is a security measure designed to prevent man-in-the-middle (MITM) attacks. It involves `pinning` or hardcoding the certificate or public key of a known, trusted server into an application. This way, the app can verify the identity of the server it's connecting to, ensuring that the data is being sent to and received by the correct entity.

When an app communicates over HTTPS, it relies on a chain of trust established through Certificate Authorities (CAs). A CA is an entity that issues digital certificates, which are used verify the identity of participants in secure communications over the Internet. The CA issues certificates to entities after verifying their identity, and the app trusts the certificate if it is signed by a known CA. However, this trust model is not foolproof. If a CA is compromised or impersonated, an attacker mmight intercept and alter communications. Certificate pinning strengthens this trust model by having the app carry a predefined copy of the server's certificate or public key. When the app makes a network request, it compares the received server's certificate (or public key) with the one it has pinned. If they match, the app can trust the server. If not, the app can assume a potential MITM attack and abort the connection. Certificate pinning can be done by hardcoding the entire digital public certificate or only the public key (or a hash of the public key) of the server in the app. Both are considered effective methods to enhance the security of an application and prevent man-in-the-middle (MITM) attacks.

Despite its advantages, SSL/TLS pinning can still be bypassed using dynamic instrumentation tools like Frida. These allow attackers to modify the app's behavior at runtime, including bypassing security checks. In this example, we'll demonstrate how to bypass certificate pinning using an Android Virtual Device (AVD), although the method applies equally to physical devices. Begin by connecting to the AVD via ADB and installing the application:

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ adb connect
rl1k@htb[/htb]$ adb install myapp.apk

Performing Streamed Install
Success
```

Device Setup

Before launching the app, configure the hosts file on the AVD to resolve the target domain to the appropriate IP address. First, boot the AVD in writable mode. Assuming you've created a rooted AVD named `Pixel_3a_API_34` and that Android SDK is installed in your home directory, run the following:

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ ~/Android/sdk/emulator/emulator -avd Pixel_3a_API_34 -netdelay none -netspeed full -dns-server 8.8.8.8 -writal

INFO    | Android emulator version 33.1.23.0 (build_id 11150993) (CL:N/A)
INFO    | Found systemPath /Users/bertolis/Library/Android/sdk/system-images/android-34/google_apis/arm64-v8a/
INFO    | Storing crashdata in: , detection is enabled for process: 14952
INFO    | Duplicate loglines will be removed, if you wish to see each individual line launch with the -log-nofilter flag.
INFO    | Changing default hw.initialOrientation to portrait
INFO    | Increasing RAM size to 2048MB
WARNING | System image is writable
<SNIP>
```

Once the device has started, issue the following commands. Replace the IP address `192.168.5.183` with that of the remote server.

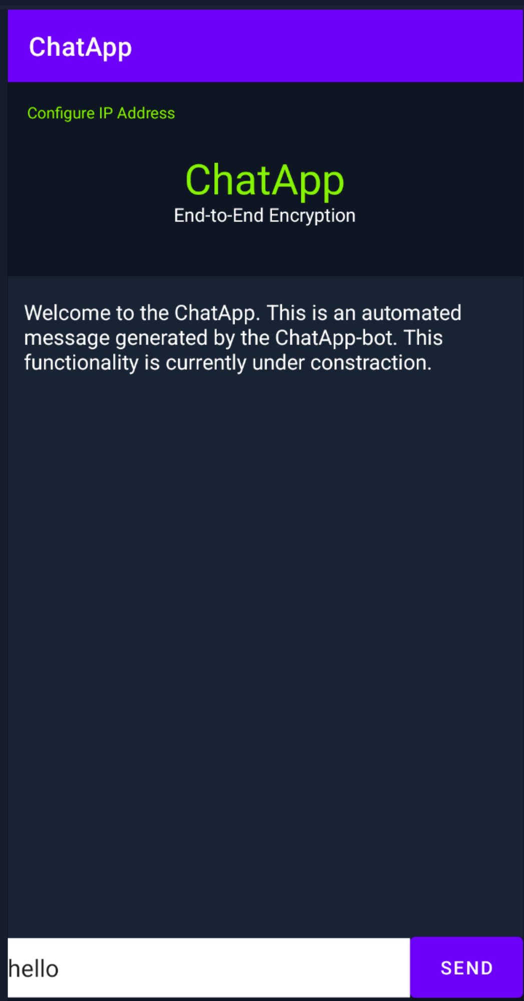
SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ adb root
rl1k@htb[/htb]$ adb remount
rl1k@htb[/htb]$ adb shell mount -o rw,remount /system
rl1k@htb[/htb]$ adb shell 'echo "192.168.5.183   www.chatapp.com" >> /system/etc/hosts'
rl1k@htb[/htb]$ adb shell mount -o ro,remount /system
rl1k@htb[/htb]$ adb shell reboot
```

This will remount the `/system` partition in writable mode, allowing us to edit files located within it. After editing the `hosts` file, the process remounts the partition in read-only mode and reboots the device. Once the domain has been added to the hosts file, any requests from the app to this domain will be redirected to the IP address 192.168.1.183. After the device restarts, we can run the application. Replace with the server's `PORT` while using the app if necessary.

Bypassing SSL Pinning

Running the application, we see that it's an app designed for chatting and sending encrypted messages.



Let's examine the application's source code using JADX, and find out if the communication with the remote server is secure.

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```

The `sendMessage()` method saves encrypted messages by calling `saveMessageToDatabase()`. It also invokes the method:

Code: java

```java
MainActivity.this.m126lambda$sendMessage$1$comhacktheboxchatappMainActivity(obj);
```

This in turn calls `sendToServer()`, whose return value is the message written to the database.

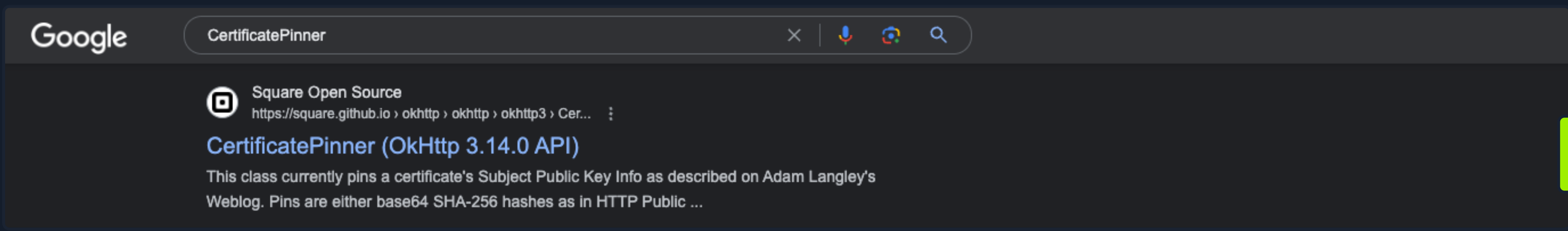

```java
    private String sendToServer(String str) {
        try {
            Response execute = new OkHttpClient.Builder().certificatePinner(new CertificatePinner.Builder().add("www.chatapp.com", certHash())
.build()).build().newCall(new Request.Builder().url("https://www.chatapp.com:8000/message").post(new FormBody.Builder().add(DatabaseHelper.
COLUMN_MESSAGE, str).build()).build()).execute();
            if (!execute.isSuccessful()) {
                throw new IOException("Unexpected code " + execute);
            }
            String optString = new JSONObject(execute.body().string()).optString("responseMessage", "No message found");
            if (execute != null) {
                execute.close();
            }
            return optString;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

The snippet above sends a POST request to `https://www.chatapp.com:8000/message` and includes the user's input, `str`, as a parameter. Notably, it also implements certificate pinning via the line:

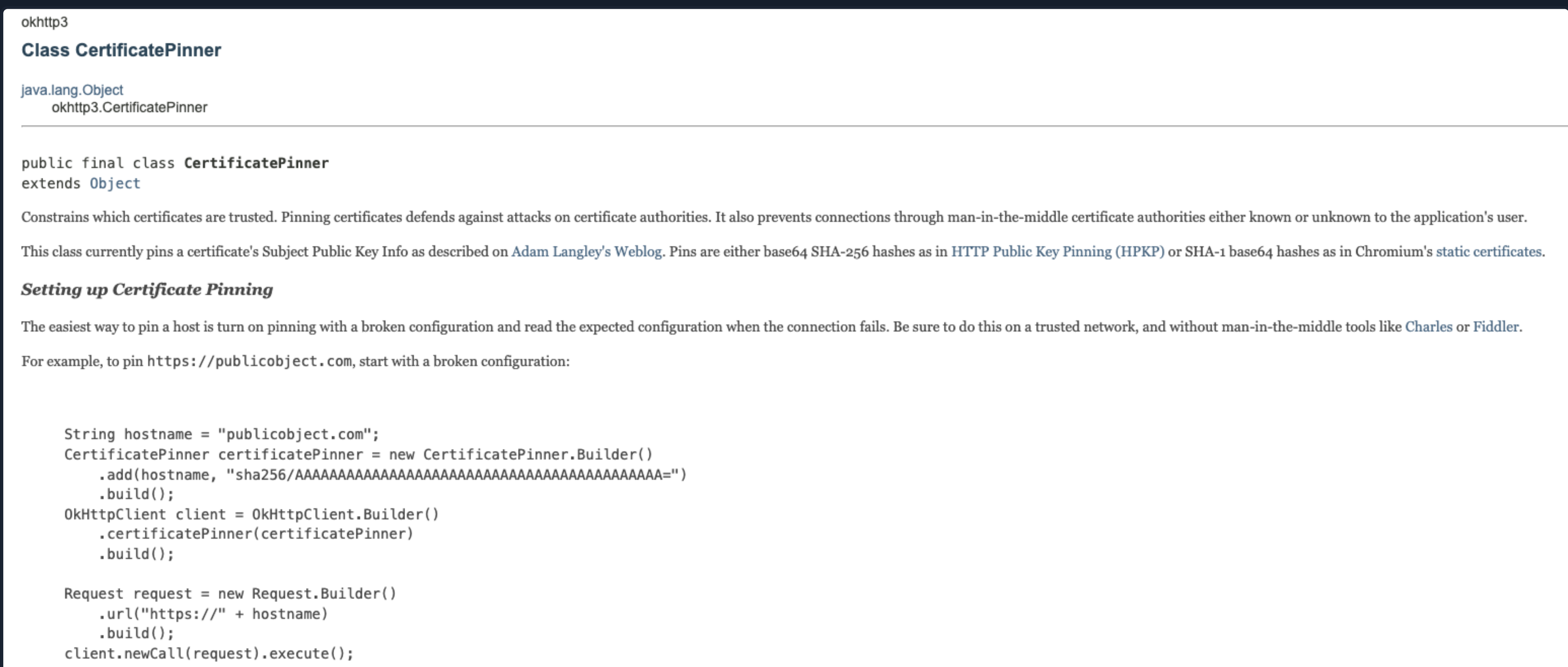
Code: java

```java
CertificatePinner.Builder().add("www.chatapp.com", certHash());
```

Searching for `CertificatePinner` leads to documentation for the `okhttp3` library:



Here, we find that `CertificatePinner` exists as a class within the `okhttp3` package.



okhttp3

Class CertificatePinner

java.lang.Object
    okhttp3.CertificatePinner

```
public final class CertificatePinner
extends Object
```

Constrains which certificates are trusted. Pinning certificates defends against attacks on certificate authorities. It also prevents connections through man-in-the-middle certificate authorities either known or unknown to the application's user.

This class currently pins a certificate's Subject Public Key Info as described on Adam Langley's Weblog. Pins are either base64 SHA-256 hashes as in HTTP Public Key Pinning (HPKP) or SHA-1 base64 hashes as in Chromium's static certificates.

Setting up Certificate Pinning

The easiest way to pin a host is turn on pinning with a broken configuration and read the expected configuration when the connection fails. Be sure to do this on a trusted network, and without man-in-the-middle tools like Charles or Fiddler.

For example, to pin `https://publicobject.com`, start with a broken configuration:

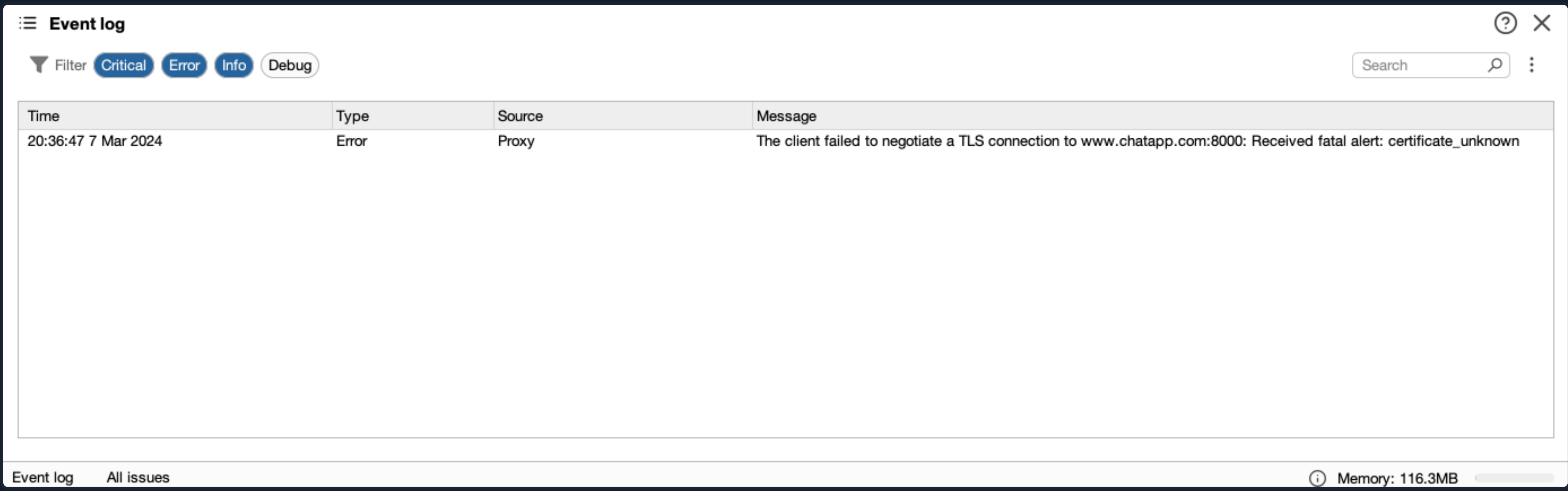
```java
    String hostname = "publicobject.com";
    CertificatePinner certificatePinner = new CertificatePinner.Builder()
        .add(hostname, "sha256/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=")
        .build();
    OkHttpClient client = OkHttpClient.Builder()
        .certificatePinner(certificatePinner)
        .build();

    Request request = new Request.Builder()
        .url("https://" + hostname)
        .build();
    client.newCall(request).execute();
```

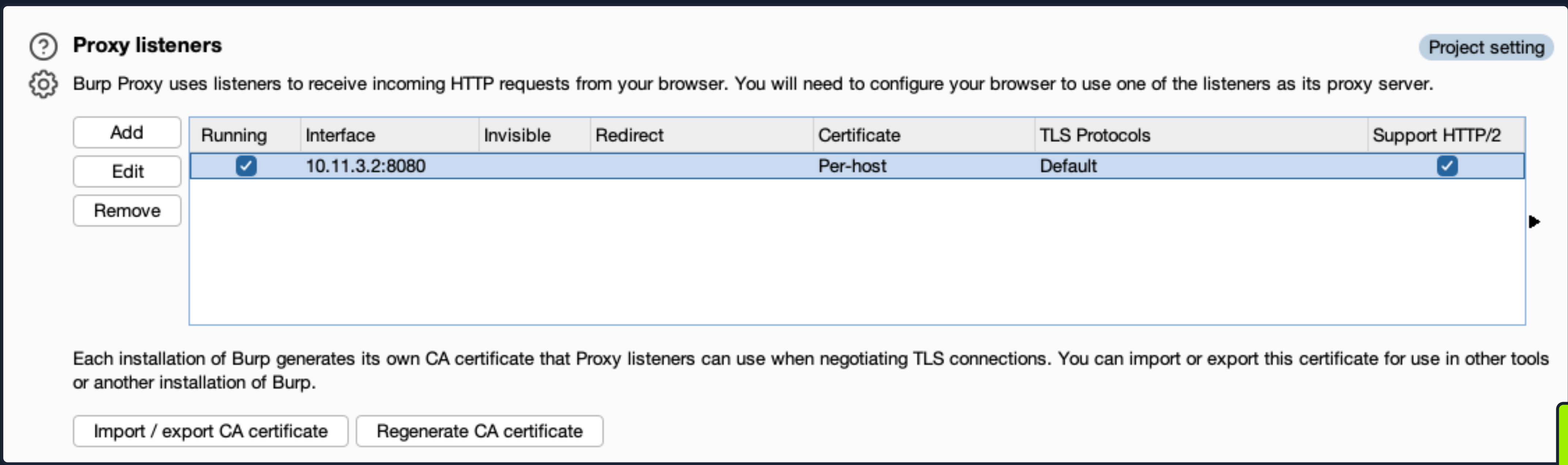
According to the documentation, `CertificatePinner` pins a certificate's Subject Public Key Info using either base64 SHA-256 (commonly used in HTTP Public Key Pinning) or SHA-1 hashes. The usage pattern `.add(hostname, "sha256/...")` matches the one found in the application. Based on this, the pinned host is `www.chatapp.com`, and the certificate hash is returned from the method `certHash()`. This confirms the app uses both HTTPS and certificate pinning to secure communication. Let's try to intercept this request using Burp Suite. Follow the same proxy setup steps outlined in the previous `Intercepting API Calls` section. Then, send a message (e.g., `hello`) from the app.

Although the request was issued, Burp's Intercept tab shows nothing. However, the Event Log shows activity:

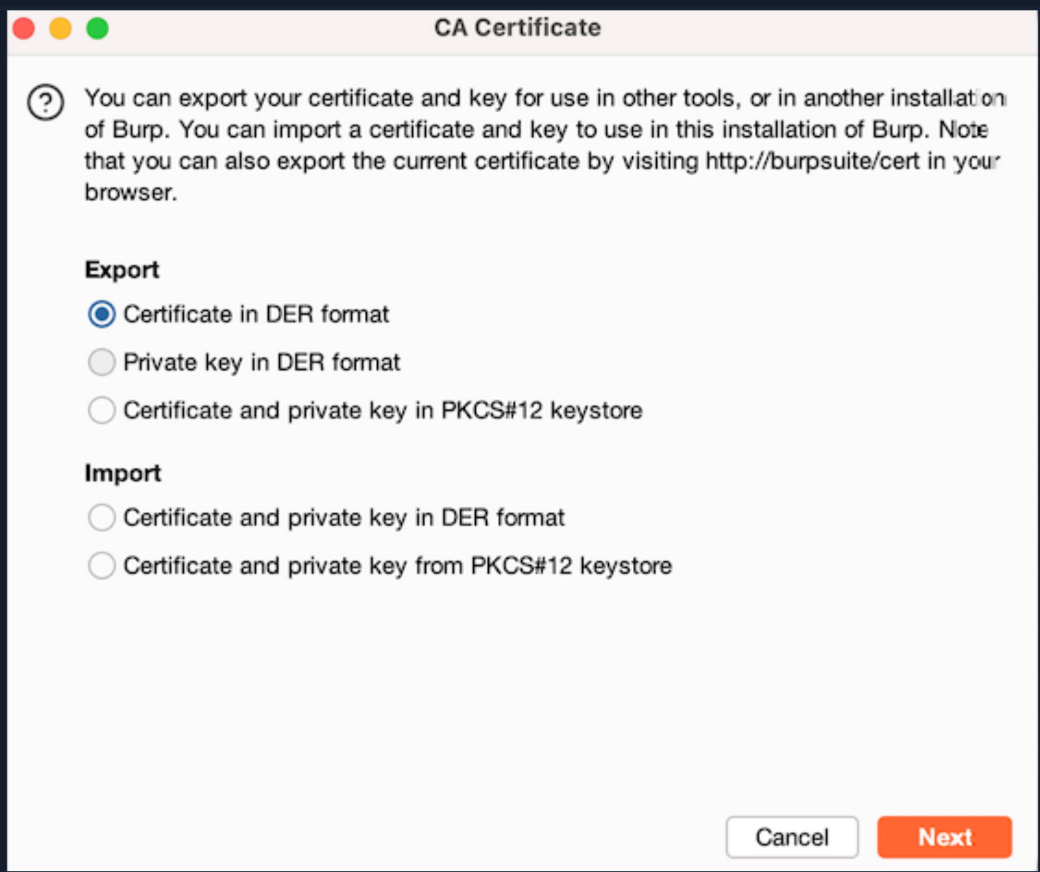
Although the request was issued, Burp's intercept tab shows nothing. However, the Event Log shows activity.



The alert indicates that the client does not trust or recognize the server's TLS certificate. We can resolve this issue by creating a CA certificate in Burp, installing it on the device, and configuring the application to trust this custom certificate. In Burp, we navigate to `Proxy` -> `Proxy Settings`, and under the `Proxy listeners` section, we click on `Regenerate CA certificate` and click `Yes` to the pop-up window.



Next, click on `Import/export CA certificate`, select `Certificate in DER format` under the `Export` section, and follow the steps to save the file.



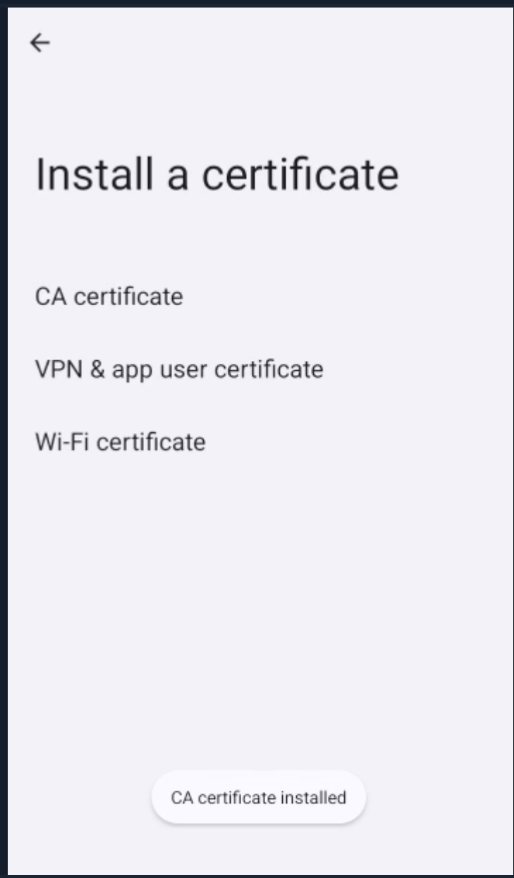
In this example, we will save the file on the Desktop using the name `BurpCA.cer`. Then, we can exit the `Settings` window and upload and install the exported file on the device. We can upload the file using ADB.

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ adb push ~/Desktop/BurpCA.cer /sdcard/Download/

/Users/bertolis/Desktop/BurpCA.cer: 1 file pushed, 0 skipped. 0.3 MB/s (939 bytes in 0.003s)
```

Now, we can install the certificate by navigating to Settings -> Security and privacy -> More security & privacy -> Encryption & credentials -> Install a certificate -> CA certificate -> INSTALL ANYWAY, and tap the file BurpCA.cer. The message CA certificate installed should be displayed on the screen.



From here, we can inspect the app's network configuration using APKTool:

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ apktool d ChatApp.apk
rl1k@htb[/htb]$ ls -l ChatApp/res/xml/

total 24
-rw-r--r--  1 bertolis  bertolis   62 Mar  7 21:05 backup_rules.xml
-rw-r--r--  1 bertolis  bertolis  108 Mar  7 21:05 data_extraction_rules.xml
-rw-r--r--  1 bertolis  bertolis  304 Mar  7 21:05 network_security_config.xml
```

This reveals the file network_security_config.xml, which contains the network security configurations of the app. Therefore, it can be used to specify custom trusted Certificate Authorities (CA).

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ cat ChatApp/res/xml/network_security_config.xml

<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config cleartextTrafficPermitted="true">
        <trust-anchors>
            <certificates src="system" />
            <certificates src="@raw/certificate" />
        </trust-anchors>
    </base-config>
</network-security-config>%
```

The line <certificates src="@raw/certificate" /> indicates that the app stores and trusts a certificate located in ./raw/certificate. Listing this directory shows the following results.

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ ls -l ChatApp/res/raw

-rw-r--r--  1 bertolis  bertolis  955 Mar  7 21:05 certificate.der
```

Let's patch the application and replace this certificate with the one we exported from Burp.

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ cp BurpCA.cer ChatApp/res/raw/certificate.der
```

Next, we re-build and sign the app using the following commands.

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ apktool b ChatApp
rl1k@htb[/htb]$ echo -e "password\npassword\njohn doe\ntest\ntest\ntest\ntest\ntest\nyes" > params.txt
rl1k@htb[/htb]$ cat params.txt | keytool -genkey -keystore key.keystore -validity 1000 -keyalg RSA -alias john
rl1k@htb[/htb]$ zipalign -p -f -v 4 ChatApp/dist/ChatApp.apk myChatApp.apk
rl1k@htb[/htb]$ echo password | apksigner sign --ks key.keystore myChatApp.apk

Keystore password for signer #1:
```

Uninstall `ChatApp` and install the patched `myChatApp`. We already know the app's package name from our earlier analysis with JADX.

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ adb uninstall com.hackthebox.chatapp
rl1k@htb[/htb]$ adb install myChatApp.apk

Performing Incremental Install
Serving...
All files should be loaded. Notifying the device.
Success
Install command complete in 125 ms
```

After replacing the trusted `certificate.der` file in `network_security_config.xml` with the one exported from Burp, sending a message and intercepting it with Burp no longer triggers any alert messages or errors. This indicates that the application now trusts the Burp certificate. However, to fully intercept the requests in plaintext, we also need to address certificate pinning. The app likely uses a hardcoded SHA256 hash of the server's p public key, and it may still block the connection if the received certificate doesn't match this value. To bypass this, we must replace the hardcoded hash with the SHA256 hash of the Burp certificate.

As a first step, we'll use Frida to hook the return value of the `certHash()` method and confirm that it returns the expected SHA256 value. Let's create a file named `snippet.js` and add the following content:

Code: js

```js
// Wait for 5 seconds before executing the function to ensure the Java environment is fully loaded.
setTimeout(function() {
    // Perform operations within the Java VM
    Java.perform(function () {
        // Reference the MainActivity class of the target application
        var MainActivity = Java.use("com.hackthebox.chatapp.MainActivity");

        // Check if the certHash method exists in MainActivity
        if (MainActivity.certHash) {
            // Override the implementation of certHash method
            MainActivity.certHash.implementation = function () {
                // Call the original certHash method and store its return value
                var returnValue = this.certHash();
                // Log the original return value of certHash() method
                console.log("\ncertHash() return value: " + returnValue);

                // Return the original certificate hash value without making any changes
                return returnValue;
            };
            // Log a message indicating successful hooking of the certHash method
            console.log("certHash method hooked successfully.");
        } else {
            // Log a message if the certHash method cannot be found in MainActivity
```

```
            console.log("certHash method not found.");
        }
    });
// Set a delay of 5000 milliseconds (5 seconds) before executing the above script
}, 5000);
```

Then, we execute the following command to run the script, and once the app is started, we tap the `SEND` button..

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ frida -U -l snippet.js -f com.hackthebox.chatapp                                   14s ≡
     ____
    / _  |   Frida 16.1.11 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.hackthebox.chatapp`. Resuming main thread!
[Android Emulator 5554::com.hackthebox.chatapp ]-> certHash method hooked successfully.

certHash() return value: sha256/dsWDLWseOwJ5FOuYjjooIdLtY49WuQAYWE4V9ZkuhHE=
```

The return value confirms that it's the SHA256 hash of the public key. Let's change the return value to the SHA256 hash of the Burp's certificate. First, we have to extratct it by issueing the following command.

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ openssl x509 -in BurpCA.cer -pubkey -noout | openssl rsa -pubin -outform der | openssl dgst -sha256 -binary

writing RSA key
oIRt9h6JBHnXEXpbd3R/SocR4j4Clv2+lyZXssKOFTA=
```

Next, update the script to look like this:

Code: js

```js
// Wait for 5 seconds before executing the function to ensure the Java environment is fully loaded.
setTimeout(function() {
    // Perform operations within the Java VM
    Java.perform(function () {
        // Reference the MainActivity class of the target application
        var MainActivity = Java.use("com.hackthebox.chatapp.MainActivity");

        // Check if the certHash method exists in MainActivity
        if (MainActivity.certHash) {
            // Override the implementation of certHash method
            MainActivity.certHash.implementation = function () {
                // Call the original certHash method and store its return value
                var returnValue = this.certHash();
                // Define a new certificate hash value
                var newCertHash = "sha256/oIRt9h6JBHnXEXpbd3R/SocR4j4Clv2+lyZXssKOFTA=";

                // Log the original return value of certHash() method
                console.log("\ncertHash() return value: " + returnValue);
                // Log the new certificate hash value you wish to use
                console.log("certHash() new value: " + newCertHash);

                // Return the original certificate hash value without making any changes
                return newCertHash;
```

```
            };
            // Log a message indicating successful hooking of the certHash method
            console.log("certHash method hooked successfully.");
        } else {
            // Log a message if the certHash method cannot be found in MainActivity
            console.log("certHash method not found.");
        }
    });
// Set a delay of 5000 milliseconds (5 seconds) before executing the above script
}, 5000);
```

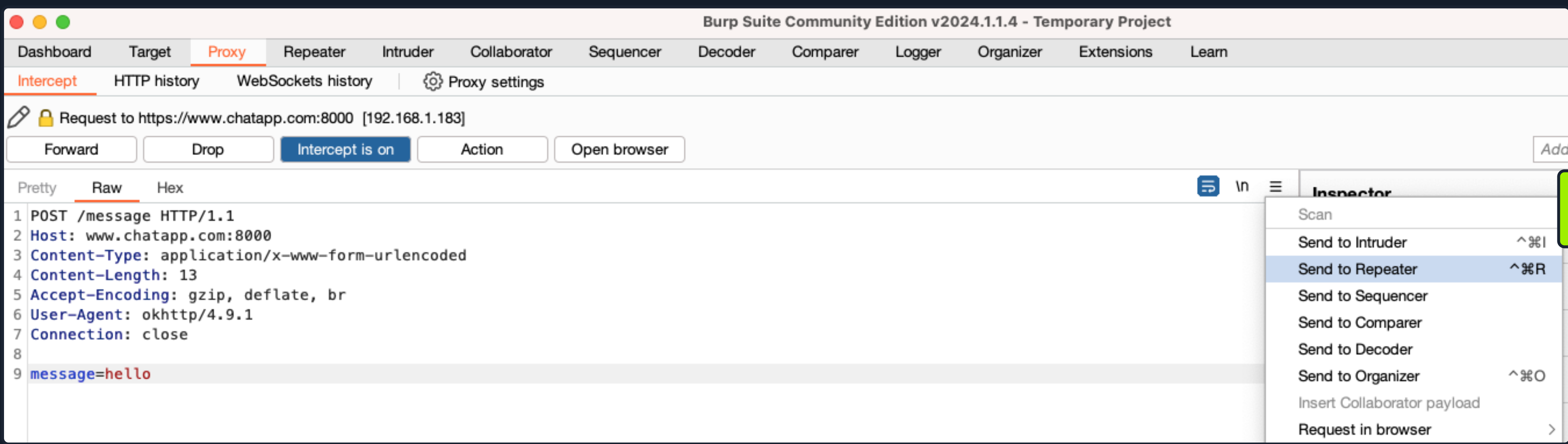
This time, the SHA256 hash `oIRt9h6JBHnXEXpbd3R/SocR4j4Clv2+lyZXssKOFTA=` will be returned to the application, overriding the original pinned hash. This forces the app to trust the Burp certificate, allowing the request to be intercepted in plaintext. Run the following command to execute the script:

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ frida -U -l snippet.js -f com.hackthebox.chatapp

<SNIP>
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.hackthebox.chatapp`. Resuming main thread!
[Android Emulator 5554::com.hackthebox.chatapp ]-> certHash method hooked successfully.

certHash() return value: sha256/dsWDLWseOwJ5FOuYjjooIdLtY49WuQAYWE4V9ZkuhHE=
certHash() new value: sha256/oIRt9h6JBHnXEXpbd3R/SocR4j4Clv2+lyZXssKOFTA=
```

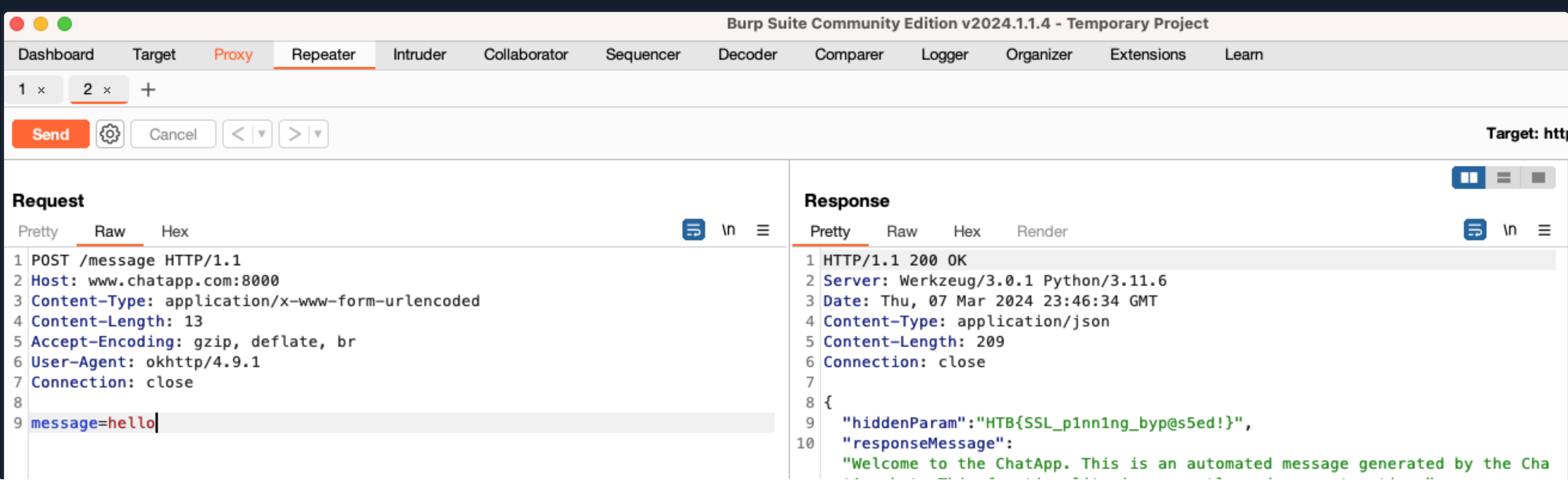


Just as we anticipated, the parameter `message=hello` is intercepted successfully. Let's now click on three rows on the right of the window and select `Send to Repeater` to then intercept the response of the request. To accomplish this, we first need to configure our host machine's `/etc/hosts` file to map the domain name `www.chatapp.com` to the server's IP address.

SSL/TLS Certificate Pinning Bypass

```
rl1k@htb[/htb]$ echo "192.168.1.183   www.chatapp.com" | sudo tee -a /etc/hosts > /dev/null
```

Finally, we navigate to `Repeater` from the top menu bar and click on `Send`.



```
tApp-bot. This functionality is currently under constraction."
}
```

We have effectively bypasses the certificate pinning, as the parameter `hiddenParam` is ultimately revealed.

Connect to Pwnbox

Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK 41ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Cheat Sheet

Answer the question(s) below to complete this Section and earn cubes!

Target(s): Click here to spawn the target system!

+ 5 What is the value of the parameter "hiddenParam" found in the server's HTTP response?

Submit your answer here...

+10 Streak pts Submit ssl_pinning.zip

Go to Questions

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left