Digital Payment Apps : Week 3

Shreejeet Golhait and Tushman Khalse

Tasks Completed

- **Figured out SNI of TLS handshakes which occurred during Payment using Python** : We made a python script that uses dpkt library to analyse PCAP files. It returns the SNI of all TLS handshakes transactions present in the file. It also gives a graph of time vs TLS handshakes. We used this script to figure out what SNI had network traffic during payment transaction of various Digital Payment Apps.

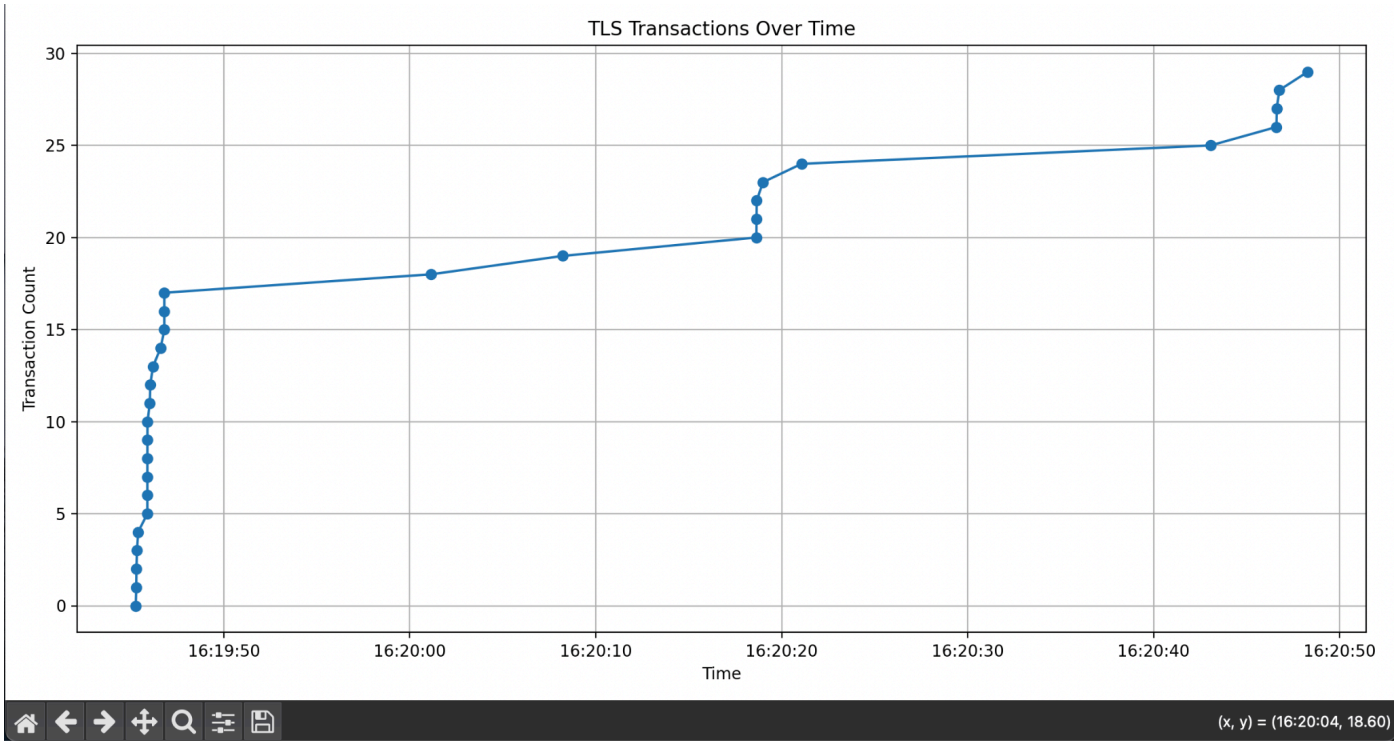
```
Total packets processed: 1174
IP packets found: 1174
TCP packets found: 1016
Possible TLS packets (port 443): 999
TLS Handshakes found: 66
ClientHello messages found: 30
SNIs extracted: 30

SNI found:
- digitalapiproxy.paytm.com
- ws-7cc3e8d0-d35b-4685-b603-135df578e7de.sendbird.com
- people-pa.googleapis.com
- app-measurement.com
- sig.paytm.com
- securegw-online.paytm.in
- update.googleapis.com
- firebaselogging-pa.googleapis.com
- api-7cc3e8d0-d35b-4685-b603-135df578e7de.sendbird.com
- batchlogging4-noneu.truecaller.com
- kyc.paytmbank.com
- instantmessaging-pa.googleapis.com
- jumbo.zomato.com
- graph.facebook.com
- tvybx4-cdn-settings.appsflyersdk.com
- assetscdn1.paytm.com
- m.media-amazon.com
- tvybx4-launches.appsflyersdk.com
- storefront.paytm.com
- paytm.com
- firebaselogging.googleapis.com
- firebase-settings.crashlytics.com

Total TLS transactions: 30
```

- **We have uploaded the PCAP Metadata for multiple transactions on various Digital Payment Apps**: This data has been uploaded on github. We have also noted down the

transaction time for each file. Also included are the SNI for each PCAP. Out of all the SNI we were able to observe, we noted down the common ones for multiple Digital Payment Apps.



- **Figured out the Certificates trusted by UPI apps**: I started by decompiling the Paytm APK using Jadx to explore the app's code and configuration files, aiming to understand how network security and certificate pinning were implemented. I first examined the network_security_config.xml file, which showed that the app only trusted system certificates by default, with no custom certificates listed there.

  Next, I searched for references to custom trust managers and SSL configurations to find any non-standard implementations for handling certificates. This led me to discover the VisaPubKeyManager class, which implements X509TrustManager and uses hard-coded certificate keys for server certificate validation, indicating the use of certificate pinning.

  I then investigated the VisaPubKeyManager and TLSSocketFactory classes and confirmed that these classes were performing certificate pinning by validating server certificates against hard-coded public keys. To understand where and how certificates and public keys were stored or used in the app, I searched for keywords like CERTIFICATE, Base64.decode, and PUBLIC KEY, finding several .cer files related to UIDAI (Unique Identification Authority of India), such as uidai_aadhar.cer and uidai_auth.cer. These certificates, however, were not directly related to the custom trust manager but seemed to be used for other specific authentication purposes.

I found the test_ap.cer file in the assets folder of the APK. Unlike the UIDAI certificates, test_ap.cer contained a private key instead of a certificate. I searched for references to test_ap.cer in the code and found it was used in a single class, mv.a, where it was processed into a PrivateKey object and used for signing data. This private key was base64-decoded and converted into a PrivateKey object, which was then used to sign data, with the generated signatures added to JSON objects in HTTP requests, likely for client-side authentication or data integrity.

Continuing my exploration, I searched for uses of HttpURLConnection and OkHttpClient to understand how the app handled network connections and found several methods configuring HTTP requests, including custom SSL settings and potential certificate pinning configurations. I discovered that the app implements certificate pinning dynamically using the CJRSSLPin class. The SSL pinning configuration is provided in a JSON string (vp.h.f64496i), which is parsed into CJRSSLPin objects. These objects contain domain names and corresponding certificate pins, which are used to build a CertificatePinner for OkHttpClient and configure the CronetEngine.

Finally, I investigated how these signed requests were sent by exploring the classes using HttpURLConnection and OkHttpClient, confirming that the signed data was indeed used in network communication. The app's implementation of certificate pinning involves dynamically building CertificatePinner configurations using JSON data and HashMaps, which allows it to adjust its certificate pinning settings at runtime. This comprehensive analysis gave me a clear understanding of the app's approach to network security, including the role of test_ap.cer, the use of UIDAI certificates, and the dynamic SSL pinning configurations.