INSE 6110:Foundations of Cryptography

Winter 2023

Analysis of Certificate Pinning in Android Apps

Submitted to:Dr. Amr Youssef

Submitted by:

Student Name	Student Id
Deep Bhavesh Gajiwala	40231725
Devina Shah	40238009
Meet Rakeshbhai Patel	40239187
Pratiksha Ashok Kumar Pole	40230412
Rohan Yogeshkumar Modi	40255454
Simran Kaur	40241517
Snehpreet Kaur	40254443
Yash Khosla	40232363

Abstract

TLS certificate pinning is a security technique created to stop man-in-the-middle attacks (MITM) by verifying that a client application only recognizes a server's SSL/TLS certificate if it corresponds to a predefined set of public keys or fingerprints. This process enhances security by making it more challenging for attackers to intercept and change encrypted traffic between the client and server.Our report extensively explores the implementation of certificate pinning on Android by examining 100 distinct apps obtained from the app store. To identify the usage of certificate pinning, we introduce innovative static and dynamic analysis techniques that function across multiple platforms.Recently, researchers have developed innovative techniques to detect certificate pinning using both static and dynamic analysis. These approaches have improved the accuracy and efficiency of pinning detection and allowed researchers to identify vulnerabilities in real-world applications.

1. Introduction

Certificate pinning is a significant security measure to safeguard against man-in-the-middle attacks, where an attacker intercepts and alters encrypted traffic between a client and server. Two approaches can be employed to detect if an application is using certificate pinning: static analysis and dynamic analysis.
Static analysis refers to the inspection of an application's code and configuration files without executing it. This method can identify whether an application has any code related to certificate pinning, such as hardcoded public keys or fingerprints in the code or configuration files.
On the other hand, dynamic analysis involves running an application and monitoring its behaviour during runtime. This method can detect whether an application uses certificate pinning by observing how it verifies the server's SSL/TLS certificate, such as checking the server's public key against a known value or relying solely on the certificate authority chain.
Both static and dynamic analysis have their advantages and disadvantages. While static analysis can recognize the presence of certificate pinning before execution, it may not detect some forms of dynamic pinning. Meanwhile, dynamic analysis can detect more complicated types of pinning but requires more resources to execute.

2. Certificate Pinning

Pinning is a security procedure that involves associating an SSL/TLS certificate or public key with a specific server or domain in order to prevent man-in-the-middle (MITM) attacks. These attacks can jeopardise the confidentiality and integrity of client-server communication by allowing an attacker to intercept and modify encrypted traffic.

Pinning prevents this by allowing a client application to accept connections only from servers that have the specified certificate or key. This ensures that the client is communicating with the intended server, rather than an attacker who has intercepted and altered the traffic.
There are two main types of certificate pinning:

1. HPKP (HTTP Public Key Pinning): HPKP is a security standard that allows web servers to send a header to the client's browser indicating which public keys should be used to validate the server's SSL/TLS certificate. The client's browser will then save these keys and accept only certificates that match at least one of them. HPKP provides strong defence against MitM attacks, but it has been deprecated due to the complexity of managing it and the risks of misuse.
2. CA Pinning: CA pinning entails hardcoding a trusted CA's public key or hash directly into the client's code or relying on the client's trust store. This ensures that the client will only accept certificates signed by the trusted CA, even if the server's certificate was signed by a different CA. CA pinning is easier to implement than HPKP and can

be used for a broader range of applications; however, it may be less secure because it is dependent on the trustworthiness of the trusted CA.

There are two ways to implement pinning within these two main types of pinning:

- Hard Pinning: Hard pinning involves hardcoding the server's certificate's public key or hash directly into the client's code. Because the client will only accept a certificate that matches the hardcoded value, this approach provides the highest level of security. Hard pinning, on the other hand, can be difficult to manage, especially if the server's certificate changes frequently.
- Soft Pinning: Soft pinning is based on the client's trust store, such as the operating system's trusted root CA list. The client verifies that the server's certificate chain is signed by a trusted CA and that the certificate presented by the server matches the expected details during soft pinning. Soft pinning is less secure than hard pinning because it relies on the trustworthiness of the underlying trust store.

3. Methodology

3.1 Static : Here first we are downloading the app apk files from the web . After that we are extracting build files from apk using apktool in kali linux. Command for that operation is “sudo apktool d APK_FILE_NAME.apk”. All the extracted files will be stored in a folder with the same name as apk file name. To check if any network configuration is available or not, we are using an IDE like atom/vs code to open the apk folder as a project. Every apk project contains an AndroidManifest.xml file and it contains information about network configuration files used in the project. So, if we can find that file name using the “network” keyword, we can get the network configuration file name which has information about certificates (Related to network) used in the project. If any certificate file name is mentioned in that file then we can directly search for that file throughout the project and get the certificate file but if no certificate file name is mentioned over there then we have to explicitly look for the files with extensions like .pem, .cer, .cert, .crt, .drt, .dert in project. Another way to find certificates is that we can also search for the “begin certificate” keyword in the whole project to see if there is any certificate used in the project or not.

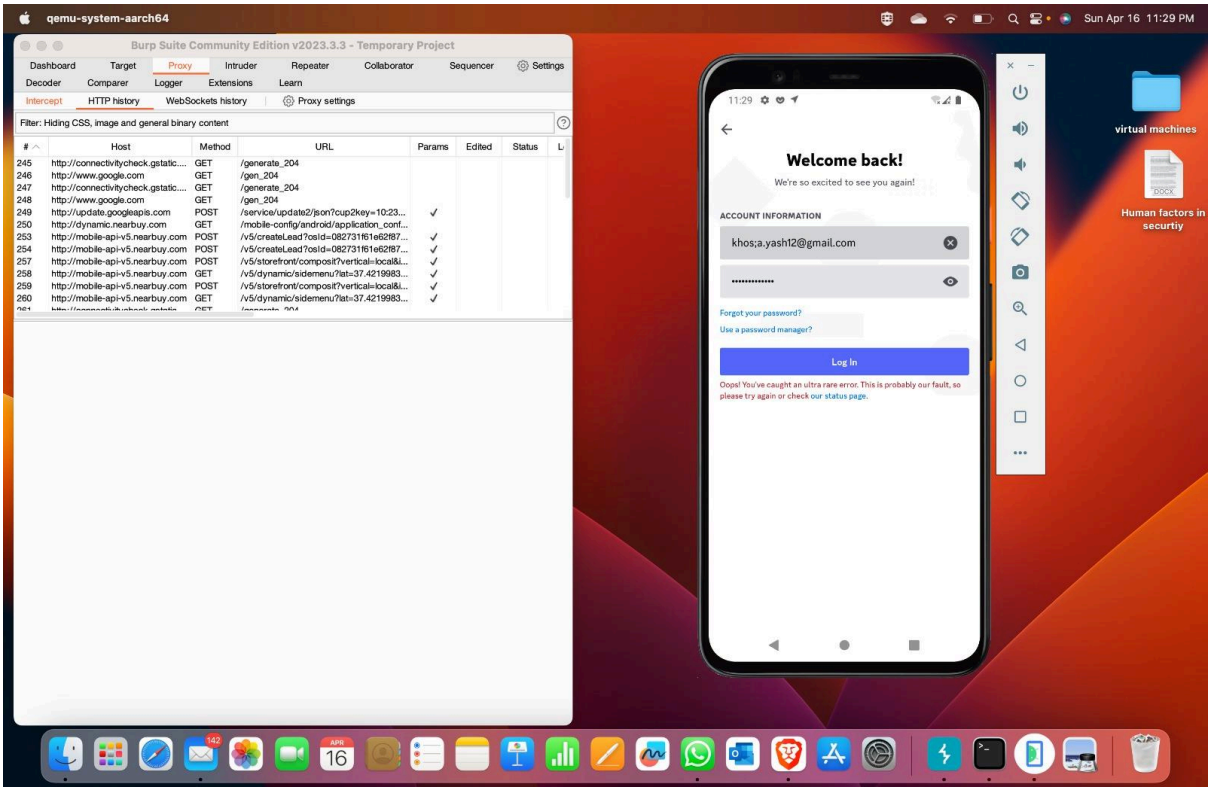
Dynamic:

We utilised Burp Suite Community Edition and Android Emulator API for performing Dynamic Analysis. To set up Burp Suite, we created a temporary project and configured proxy settings by adding a proxy with port 80 and binding it to all interfaces. We also enabled the Support invisible proxying checkbox under Request handling in the Edit proxy listener section, and turned off the intercept. We exported the Proxy's CA certificate to our local machine with a .crt extension by clicking the Import/Export CA certificate button and then dragged the exported certificate to the applications. We turned on the intercept under the Proxy -> Intercept section.

For setting up the Android emulator, we installed Android Studio API 28 and added the latest emulator. If Playstore was not available, we

installed it. We added the manual proxy details from Burp Suite under advanced wifi settings in the emulator settings, and dragged and dropped the installed certificate from our local machine to the emulator. We verified that the proxy settings were correct under advanced wifi settings and selected No Proxy under the Proxy tab in the emulator settings. We installed all required apps under the required categories from Play store and switched to Manual Proxy Configuration, ensuring that the proxy status was Success. We then opened each app one by one and analysed whether certificate pinning was enabled or not.

To perform the analysis, we opened an app and attempted to log in or create an account if required. Based on the app's behaviour, we determined whether certificate pinning was enabled or not. If the app showed an error code or network error after opening, then certificate pinning was enabled for the app (Hard Fail). If the app failed to authenticate even after entering valid credentials, then certificate pinning was enabled for the app (Soft Fail). If the app did not behave as expected in a normal Android device, then certificate pinning was enabled for the app. If the app successfully accepted valid credentials and worked normally as expected in a normal Android device, then certificate pinning was not enabled for the app. Finally, if the app successfully allowed us to interact with the user interface, we considered it functional.



Screenshot 1:

Conclusion

Overall, certificate pinning can be an effective security measure when properly implemented. However, it is not a silver bullet and should not be relied on as the sole means of securing a system. It can also introduce some operational challenges, such as the need to keep the pinned certificate up-to-date and the potential for service disruptions if the certificate changes unexpectedly.

As with any security measure, it is important to carefully consider the risks and benefits of certificate pinning and to evaluate its effectiveness in the context of the specific system being secured.