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.

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.

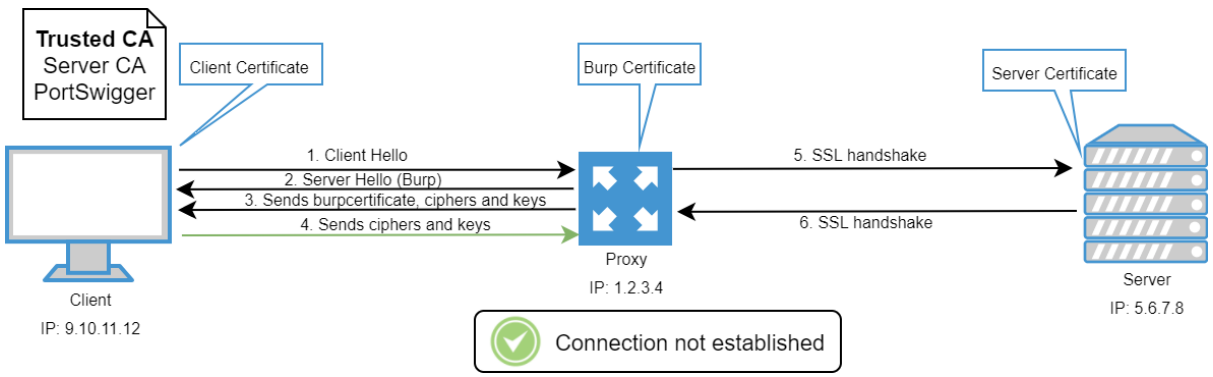

Figure 1.1: SSL Pinning

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:

- 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 :

First, we downloaded the app APK files from the web. We then extracted files from the APK using Apktool with the command `sudo apktool d APK_FILE_NAME.apk`. To check if any network security configuration file is available, we used IDEs like ATOM and VS Code.

Every APK file contains an `AndroidManifest.xml` file, which provides information about Android build tools, the Android operating system, and Google Play files used in the project. Our next step involved examining the network security configuration file to determine if the application includes any CA certificates.

If a network security configuration file was found, we traversed through its lines, specifically looking for the `trust-anchors` tag, which contains details about authority certificates. If no network security configuration file was present, we searched for related file extensions such as .pem, .cer, .cert, .crt, .drt, or .dert within the project files.

Another method we used to find certificates involved searching for the keyword "BEGIN CERTIFICATE" to check if hashes were stored as comments on separate lines. We also

examined files for any encoded certificates or embedded keys, which could be part of the application's security mechanism.

In addition to these checks, we reviewed the application's code for any custom implementations of network security protocols, which might bypass standard configuration files. This included analyzing source code files for hardcoded certificate hashes, custom trust managers, or any use of certificate pinning libraries.

By combining these methods, we aimed to thoroughly assess the presence and implementation of certificate pinning and other network security configurations within each application.

3.2.Dynamic:

We utilized Burp Suite Community Edition and an Android Emulator API for dynamic analysis. Following the proxy setup guide from Burp Suite (https://portswigger.net/burp/documentation/desktop/mobile/config-android-device), we configured the proxy and examined each app to determine if certificate pinning was enabled. During the analysis, we attempted to log in or create an account if required. The app's behavior indicated the results of certificate pinning: a network error or error code upon opening the app signified that certificate pinning was enabled (Hard Fail). If the app failed to authenticate despite valid credentials, it indicated certificate pinning with a Soft Fail. If the app functioned normally on a standard Android device, it meant certificate pinning was disabled.

To set up Burp Suite, we created a temporary project, added a proxy with port 80 bound to all interfaces, enabled the Support invisible proxying checkbox under Request handling, and turned off intercept. We exported the Proxy's CA certificate to our local machine with a .crt extension and then transferred the certificate to the applications. We enabled intercept under the Proxy -> Intercept section.

For the Android emulator setup, we installed Android Studio API 28, added the latest emulator, and, if necessary, installed the Play Store. We configured the manual proxy details from Burp Suite under advanced Wi-Fi settings in the emulator settings and transferred the installed certificate from our local machine to the emulator. After verifying the correct proxy settings and ensuring the proxy status was Success, we installed all required apps from the Play Store and switched to Manual Proxy Configuration.

We then opened each app, attempting to log in or create an account if required, to analyze whether certificate pinning was enabled. An error code or network error indicated a Hard Fail (certificate pinning enabled). Failure to authenticate with valid credentials indicated a Soft Fail. If the app functioned normally as on a standard Android device, certificate pinning was disabled. If the app allowed interaction with the user interface, we considered it functional.

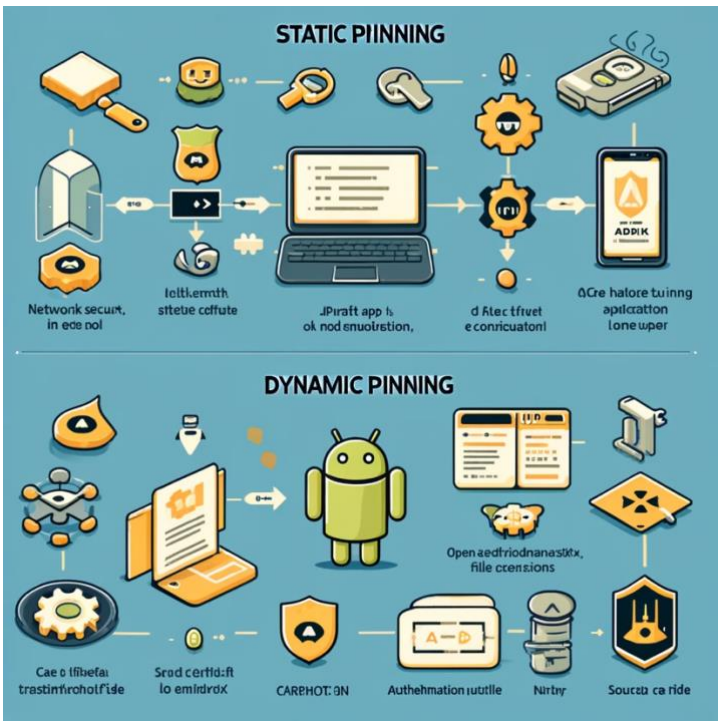

Figure 3.2.1: comparison between Static and Dynamic Pinning

4. Tools and Technologies:

- **Kali linux:** Kali Linux is mainly used to initiate advanced-level Security Auditing and Penetration Testing. The OS comprises numerous tools responsible for carrying out tasks like information security, security research, penetration testing, reverse engineering, and computer forensics.
- **Android Emulator:** An Android emulator is a tool that creates virtual Android devices (with software and hardware) on your computer. Note that: It is a program (a process that runs on your computer's operating system). It works by mimicking the guest device's architecture (more on that in a bit).
- **Burp suite:** Burp Suite is an integrated platform/graphical tool for performing security testing of web applications. Its various tools work seamlessly together to support the entire testing process, from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities.
- **Atom:** Its developers call it a "hackable text editor for the 21st Century" (Atom 1.0). Atom enables users to install third-party packages and themes to customize the features and looks of the editor, so you can set it up according to your preferences and with ease (Atom).
- **Apktool**: A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications; it makes possible to debug smali code step by step.

5. **Application analysis and details:**
   The table provides a breakdown of the total count percentage for various categories of mobile applications analysed in a study. Each category represents a different type of app, and the percentages indicate the proportion of the total sample that each category constitutes.

Catagories	Total count
Finance	**4 %**
Social	**11 %**
Books and Navigation	**14 %**
Dating	**17 %**
Food	**7 %**
Shopping	**8 %**
Communication	**2 %**
Sports	**11 %**
Travel	**6 %**
Weather	**4 %**
Upskill	**1 %**
Random	**15 %**

   The distribution shows a diverse range of app types, with dating apps being the most prevalent at 17% and upskill apps being the least common at 1%. This variety reflects the broad scope of applications users interact with daily.

- **Details of every application with both static and dynamic method :**

  In the link below you can find the applications which were tested,
  https://docs.google.com/spreadsheets/d/1Z1JW5LTlZQACukgPokLxekEf9vEq3zB1YYPFtAFTzo0/edit?usp=sharing

6. Explanation

Figure 6.1 demonstrates the process of testing SSL pinning in an Android application using Burp Suite. Here's an explanation of the different components and steps illustrated:

1. Burp Suite Community Edition:
   - Setup: Burp Suite is configured as a proxy to intercept and analyze the network traffic between the Android emulator and the server. The Burp Suite window shows various tabs such as Dashboard, Target, Proxy, Intruder, etc.

- HTTP History: The HTTP history tab displays a log of the HTTP requests and responses intercepted by Burp Suite. Each entry lists details such as the host, method, URL, and status.

2. Android Emulator:
    - The Android emulator runs a specific application that you are testing for SSL pinning. The screen shows a login interface for the app, indicating a test account with an email ID and password fields ready for input.

- In figure 6.1, if the login attempt proceeds without errors and you can successfully log in, it suggests that SSL pinning is not enabled.

- If you encounter a network error or fail to log in despite valid credentials, it suggests that SSL pinning is enabled and functioning as intended.

By using Burp Suite to intercept the network traffic of the Android app running on the emulator, you can determine whether the app employs SSL pinning to protect its communication channels against man-in-the-middle attacks.

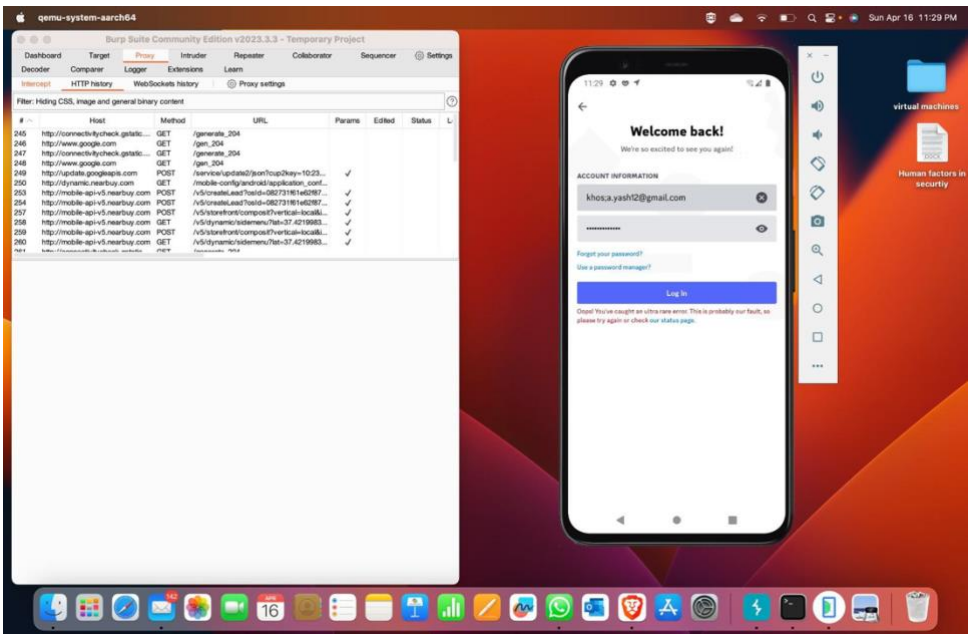

Figure 6.1

7. Conclusion

Certificate pinning can be an effective security measure when properly implemented. By associating an SSL/TLS certificate or public key with a specific server or domain, it prevents

man-in-the-middle (MITM) attacks, ensuring that the client communicates only with the intended server. However, it is not a silver bullet and should not be relied on as the sole means of securing a system.

Benefits:

- **Prevents MITM Attacks**: Pinning ensures the client only accepts connections from servers with the specified certificate or key, thereby blocking attackers who attempt to intercept and modify encrypted traffic.
- **Enhances Security**: Adds an additional layer of security by verifying the server's identity beyond standard CA validation.

Challenges:

- **Operational Complexity**: Implementing certificate pinning can introduce operational challenges. Certificates have expiration dates, and the need to update the pinned certificate regularly can complicate maintenance.
- **Service Disruptions**: If the certificate changes unexpectedly (e.g., due to an emergency certificate rotation or compromise), clients with hardcoded pins may experience service disruptions until the application is updated with the new certificate.
- **Development and Maintenance Overhead**: Developers need to manage the lifecycle of pinned certificates, which includes updating the certificates in the application code and ensuring timely deployment of these updates.
- **Compatibility Issues**: Applications with strict pinning policies may face compatibility issues, particularly when dealing with third-party services that may rotate certificates more frequently.

Best Practices:

- **Backup Pins**: Implement backup pins to provide a fallback option if the primary certificate changes. This can minimize service disruptions.
- **Use Soft Pinning**: Consider soft pinning, which leverages the operating system’s trust store, to balance security and flexibility.
- **Monitor Certificate Expiry**: Actively monitor the expiry dates of pinned certificates and plan certificate rotations well in advance.
- **Graceful Degradation**: Design the system to handle pinning failures gracefully, providing informative error messages to users and fallback mechanisms where possible.
- **Evaluate Risks and Benefits**: Carefully consider the specific context and requirements of your system when deciding to implement certificate pinning. Weigh the security benefits against the operational challenges.

Conclusion:

Certificate pinning can significantly enhance the security of client-server communications when used correctly. However, it requires careful planning and management to avoid potential pitfalls. As with any security measure, it is important to evaluate its effectiveness in the context of the specific system being secured and to use it in conjunction with other security practices to achieve a robust security posture.