Insecure Storage for iOS: iOS Static RE

Thangamuthu Balaji

I. Introduction

Data security is a crucial concern in today's mobile ecosystem, particularly for apps handling sensitive user information. This lab, titled 'Insecure Storage for iOS,' focuses on exploring common security risks in iOS app development related to data storage. The specific goal of the lab is to reverse engineer an iOS application package (.ipa) to analyze its internal structure, identify potential insecure storage methods, and review permissions and handling of sensitive data. By utilizing tools like strings, plistutil, and file in a Linux environment, we can examine the app's binary files, configuration files, and property lists (.plist) to uncover instances of insecure data storage. This process not only helps identify vulnerabilities but also offers valuable insights into best practices for securing sensitive data in mobile applications.

II. 1. a. Download the .ipa file

This is the proof for downloading the file uploaded on canvas.

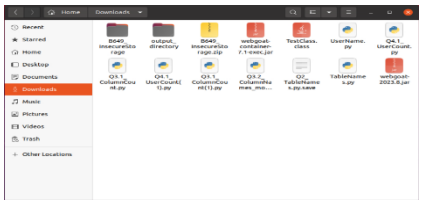
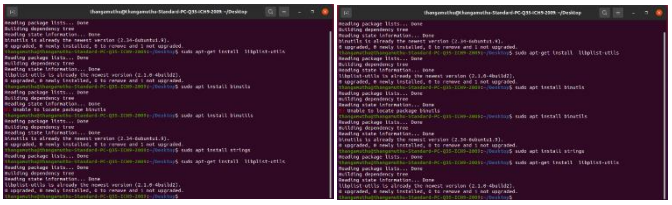


Fig. 1: File Downloaded

III. 1. b. Install Strings and Plistutil

The strings command, which is part of the binutils package, is crucial for my analysis. It helps me extract readable ASCII or Unicode text from binary files. This is really handy for reverse engineering as it allows me to find meaningful text like URLs, error messages, or sensitive data that might be hardcoded into the application. Once I installed binutils, I got access to the strings command and could start searching through binary files for any potentially insecure information. Plistutil is included in the libplist-utils package for Linux, and it is tailored for handling property list (.plist) files commonly used in macOS and iOS. These files store application settings, permissions, and occasionally sensitive information. With plistutil installed, I can examine and parse .plist files, enabling me to review the app's configurations, permissions, and any insecure storage practices. This tool played a crucial role in analyzing the Info.plist file to comprehend the app's permission requests and identify any insecure data storage.
Compare
Finetune

Command used:
sudo apt install binutils
sudo apt install libplist-utils



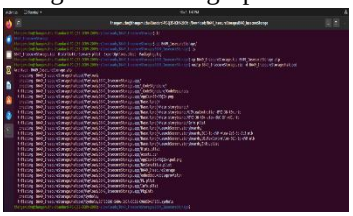
(a) Log Spoofing (b) Log Spoofing

IV. 2. a. Extracting the .ipa file

I extracted the contents of the 'B649 InsecureStorage.ipa' file into a directory called 'B649 InsecureStorage' using the 'unzip' command. I was able to view the program's internal folders and files, such as configuration files, resources, and the application binary, by unzipping the.ipa file.This phase was essential since it gave me a better understanding of the app's file structure and equipped me to look for unsafe storing practices in individual files.

Command used: unzip B649_InsecureStorage.ipa -d B649_InsecureStorage

Fig. 3: Extracting .ipa file



V. 2. b. Finding the Application Binary

I looked through every file in the B649 InsecureStorage.app directory using the file command. I was able to find the binary by searching for a file of the type Mach-O arm64 thanks to the file command, which identifies each file's type. This categorization lets me know that it's an ARM-based iOS application binary, which is what I need to look into further. Once I've located the binary, I can use strings to look for any readable text—like URLs or other potentially sensitive information that may be exposed in plaintext—within it.

Command used: file B649_InsecureStorage.app/*

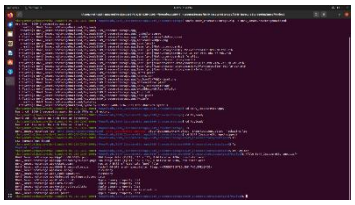


Fig. 4: Finding Binary

VI. 2. c. Searching for HTTPS URLs

I used grep and strings to look for any HTTP or HTTPS URLs in the app's files. Whereas grep -E using a regular expression screens for lines that include "http://" or "https://," the strings tool extracts readable text from files. I was able to find any hardcoded URLs in the binary or other files by using this command, which may have revealed external servers or APIs that the application communicates to. Since it may reveal communication endpoints that require security verification, this information is crucial for security analysis.
Command used: strings B649_InsecureStorage.app/* — grep -E ”http://—https://”

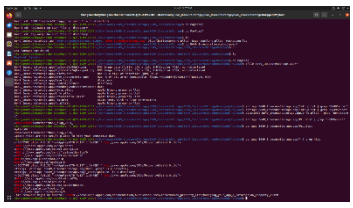


Fig. 5: HTTPS URLs

VII. 2. d. Permissions Requested

I used plistutil to view the Info.plist file, which contains the app's setup settings and permissions. I looked for keys that referred to critical rights like as location, camera, calendar, and Bluetooth access. This is an important step since excessive or unneeded permissions might suggest possible privacy violations. By studying the Info.plist file, I was able to determine which resources the program intended to access and if these requests were legitimate or possibly intrusive.

-i B649 InsecureStorage.app/Info.plist Permissios requested: Camera - We need to access the camera to scan QR codes. Location (Always) - We need to access the location to find the Luddy School.

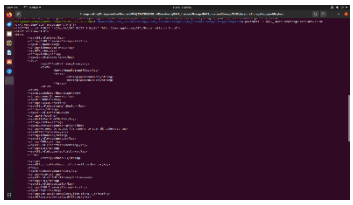


Fig. 6: Permissions

VIII. 2. e. Secret Key

I searched the Info.plist file for the B649Secret key using strings and grep. Since.plist files are not encrypted by default, storing sensitive data here might provide a serious security risk. By locating the B649Secret key and any related information, I was able to emphasize how crucial it is to store sensitive data securely rather than depending on unencrypted property list files. The necessity of improved security procedures in mobile app development was highlighted by this action.

Command used: – strings B649 InsecureStorage.app/Info.plist
— grep ”B649Secret”
What I Found: ZB649SecretXRememberB̂649 _flag _2024_

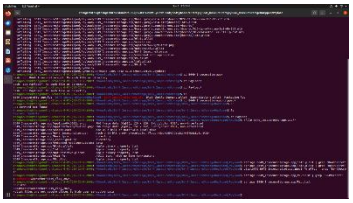


Fig. 7: Secret Key

IX. 2. f. Open Ended Exercise

I checked the directory structure of the B649 InsecureStorage.app folder using the tree command. Numerous files and folders that are typical of iOS applications are included in the structure. Notable components include Assets.car, an archive of graphics assets, and resource files like as AppI-con60x60@2x.png and AppIcon76x76@2x ipad.png, which are used for the app's icons. Storyboard files, such Main.storyboardc, that specify the application's user interface and visual layout are found in the Base.lproj directory.

There are also a number of configuration files, such as embedded.mobileprovision, which includes provisioning information, and Info.plist, which contains metadata and permissions sought by the application. App-specific settings may be stored in files like PL.plist, while other files like CodeSignature guarantee the integrity of the application. The resources, configurations, and executable components are all clearly separated by this well-organized structure.

Fig. 8: Structure of file

X. Conclusion

This lab gave us practical experience in identifying unsafe storage methods by doing static analysis on an iOS application. We found potential vulnerabilities by looking through the application's binary files, configuration files, and resources. These included unlimited permissions that might be abused and sensitive data kept in plaintext within.plist

files. This exercise emphasizes how crucial it is to handle data securely when developing mobile apps, particularly when dealing with sensitive data. Gaining knowledge of these security vulnerabilities and the methods for identifying them gives us the ability to assess and enhance the security posture of mobile apps, improving end-user data protection.

REFERENCES

[1] Strings command - https://www.sanfoundry.com/strings-command-usage-examples-in-linux/
[2] Plistutil command - https://manpages.debian.org/experimental/libplist-utils/plistutil.1.en.html
[3] File Command - https://phoenixnap.com/kb/linux-file-command
[4] Property Lists - https://developer.apple.com/documentation/bundleresources/information_property_list