Secure Code Review for Mobile Applications

Zack Anderson (1602117@uad.ac.uk)

University of Abertay Dundee

Contents

Secure Code Review for Mobile Applications

A report by (Positive Technologies, 2019) found high-risk vulnerabilities in 38% of iOS applications and 43% of Android applications. Of these, a staggering 89% could be exploited by malware installed on the device rather than exploiting physical access to such devices. Following these findings, it is important developers are aware of areas mobile applications are commonly most vulnerable. Similar to OWASP's top 10 (OWASP Foundation, 2017) for web applications, many top 10 lists have been made to highlight key areas which mobile applications should focus improvements on, including WhiteHat Security's "*Top 10 Vulnerabilities in Mobile Applications"* (WhiteHat Security, 2017), and OWASP's own "*Mobile Top 10*" (OWASP Foundation, 2016), which provides a clear yet in-depth overview of common vulnerabilities in mobile applications.

The scenario provided states that the company has a mobile application which staff use to allow them access to view and change details about their accounts. This application presumably functions by communicating with a backend database running on a webserver to retrieve and update employee information. The application could also potentially store some of this data on the device to prevent duplicate requests to the webserver and allow the employee to view their information without an internet connection.

According to the report conducted by Positive Technologies, 76% of applications failed to store data securely. This means that the application could leak sensitive data which could provide personally identifiable information to an attacker, such as their employee details. In the OWASP Mobile Top 10, *Insecure Data Storage* was ranked as the 2nd highest issue facing mobile applications.

Common Vulnerabilities and Exposures Relating to Insecure Data Storage

CVE-2014-1664

This vulnerability was found in the Android application *GoToMeeting* (NIST, 2014). *GoToMeeting* is a video conferencing and productivity service by the creators of LastPass (LogMeIn, n.d). Version 5.0.799.1238 of this application discloses information such as user identification tokens, meeting details, and authentication tokens through system logs.

CVE-2018-11242

MakeMyTrip is an online travel company. In the *MakeMyTrip* Android application version 7.2.4, the database containing potentially sensitive information is stored with no encryption (NIST, 2018a). This could allow an attacker to profile a user as part of a spear-phishing campaign or track their whereabouts using trips they have booked.

CVE-2018-11505

Werewolf Online is a multiplayer role-playing game for Android and iOS. In version 0.0.8 of the Android application, the Firebase token of the user used to store information about the player can be recovered from the logcat output of the application (NIST, 2018b).

CVE-2018-11544

FTP Server by The Olive Tree is an FTP server application for Android. This vulnerability occurs in versions below 1.32 and refers to the application storing usernames and passwords in cleartext in an XML file. This could allow an attacker to access the victims FTP server running on the device using the recovered credentials (NIST, 2018c).

Another vulnerability in the *FTP Server* application allows the user to copy the password onto the clipboard from the password field which occurs due to the string being stored in cleartext (NIST, 2018d). This occurs because the stored password is loaded into a standard text

input box, rather than one specifically designed for password input. This is classed as a critical vulnerability on the national vulnerability database as applications share the device clipboard. This vulnerability falls under the *Improper Platform Use* section in the OWASP Mobile Top 10.

Implementation

The vulnerabilities in The Olive Tree's *FTP Server* application likely could have been avoided using a security-focused code review following OWASP's Mobile Top 10. The first vulnerability, CVE-2018-11544, falls into the *Insecure Data Storage* category. The application was loaded into an Android Emulator to observe these vulnerabilities. The file used to store user credentials can be found in the following file path with the password stored in cleartext under the name *"prefUserpass"*.

/data/data/com.theolivetree.ftpserver/shared_prefs/com.theolivetree.ftpserver_preferences.xml

```
    <string name="prefUsername">francis</
string>
    <string name="prefUserpass">plaintext-
p@ssword</string>
```

Figure 1 – FTP server password, stored in plaintext

This string is loaded into a regular text input field when the user attempts to change the password, resulting in a second CVE number being assigned to the issue. This vulnerability is more severe than the first as Android restricts applications from accessing the storage space of another application, however, does not force the same restriction on the clipboard. This could allow a malicious application to steal the password for the FTP Server. Perhaps worse still is that these CVE's still have not been fixed almost 2 years later, and the application is still available on the Google Play Store with over 1 million downloads.

Mitigation

The developers of this application could have prevented these vulnerabilities by completing a secure code review, either independently or using a security specialist to thoroughly review and test their application before uploading it for consumer use. This would ensure that their applications do not pose an unacceptable level of risk to their users. This section will focus

solely on mitigating vulnerabilities that fall into the *Insecure Data Storage* category of the OWASP Mobile Top 10.

Android

Google released the Android Jetpack library in 2018, which included functionality to easily store application data securely (Markoff, 2020). It has support for Trusted Execution Environments and strongboxes, meaning that cryptographic operations are carried out on a separate chip or virtual environment to the main operating system. It works by first creating a master key, which then encrypts all sub-keys used for encrypting and decrypting data on a per-application basis. Files can be encrypted using code similar to that seen in Appendix A.

iOS

Apple also has a framework to assist developers in simply and securely storing application data on iOS devices known as the *"Security Framework"* (Rajkumar, 2014). To interface with this framework, an open-source library was created by developers called *"RNCryptor"*. Passwords for iOS can be stored securely in the Apple KeyChain, which handles password checks on the device and returns a true or false result. This can be done using the *"SSKeyChain"* library. Code to encrypt and decrypt data can be found in Appendix B.

Another option available to both Android and iOS when using SQLite databases is the *SQLCipher* extension which automatically encrypts all database files using 256-bit AES symmetric encryption (Zetetic, n.d). This extension uses similar syntax to the options already found on Android and iOS, meaning developers can learn to develop an application using the technology more quickly.

Conclusion

A security-focused code review on the *FTP Server* application would have detected the vulnerabilities which occurred in this application. This would have given the developers enough time to research and implement a more secure solution to the issues and would have prevented the application from leaking potentially sensitive information about its users. Secure code reviews are a highly effective, relatively easy to implement solution, requiring only that the developers are trained in testing and securing their applications. Compared to other secure coding practises, this will help to cover a wide area of common attack vectors used to attack applications and could also prevent the company having to pay significant fines for failing to protect user information. In the European Union, the company could potentially face up to €20 million or 4% annual turnover in fines under GDPR laws if they are found to be acting negligently regarding user protection and security (Proton Technologies, n.d).

References

Conger, K. (2016, June 15). *Apple will require HTTPS connections for iOS apps by the end of 2016*. Retrieved from techcrunch.com: https://techcrunch.com/2016/06/14/apple-will-require-https-connections-for-ios-apps-by-the-end-of-2016/

Google. (n.d). *Specify the input method type*. Retrieved from developer.android.com: https://developer.android.com/training/keyboard-input/style

LogMeIn. (n.d). *GoToMeeting*. Retrieved from gotomeeting.com: https://www.gotomeeting.com/en-gb/features

Markoff, J. (2020, February 25). *Data Encryption on Android with Jetpack Security*. Retrieved from android-developers.googleblog.com: https://android-developers.googleblog.com/2020/02/data-encryption-on-android-with-jetpack.html

NIST. (2014, January 26). *CVE-2014-1664 Detail*. Retrieved from nvd.nist.gov: https://nvd.nist.gov/vuln/detail/CVE-2014-1664

NIST. (2018a, May 20). *CVE-2018-11242 Detail*. Retrieved from nvd.nist.gov: https://nvd.nist.gov/vuln/detail/CVE-2018-11242

NIST. (2018b, May 26). *CVE-2018-11505 Detail*. Retrieved from nvd.nist.gov: https://nvd.nist.gov/vuln/detail/CVE-2018-11505#vulnCurrentDescriptionTitle

NIST. (2018c, May 29). *CVE-2018-11544 Detail* . Retrieved from nvd.nist.gov: https://nvd.nist.gov/vuln/detail/CVE-2018-11544

NIST. (2018d, June 15). *CVE-2018-12481 Detail*. Retrieved from nvd.nist.gov: https://nvd.nist.gov/vuln/detail/CVE-2018-12481

OWASP Foundation. (2016). *OWASP Mobile Top 10*. Retrieved from owasp.org: https://owasp.org/www-project-mobile-top-10/

OWASP Foundation. (2017). *OWASP Top Ten*. Retrieved from owasp.org: https://owasp.org/www-project-top-ten/

Positive Technologies. (2019, June 19). *Vulnerabilities and threats in mobile applications*. Retrieved from ptsecurity.com: https://www.ptsecurity.com/ww-en/analytics/mobile-application-security-threats-and-vulnerabilities-2019/

Proton Technologies. (n.d). *What are the GDPR Fines?* Retrieved from gdpr.eu: https://gdpr.eu/fines/

Rajkumar. (2014, February 26). *Securing and Encrypting Data on iOS*. Retrieved from tutsplus.com: https://code.tutsplus.com/tutorials/securing-and-encrypting-data-on-ios--mobile-21263

rastating. (2019, October 2). *KSWEB for Android Remote Code Execution*. Retrieved from rastating.github.io: https://rastating.github.io/ksweb-android-remote-code-execution/

Spring, T. (2019, October 7). *Vulnerable Twitter API Leaves Tens of Thousands of iOS Apps Open to Attacks*. Retrieved from threatpost.com: https://threatpost.com/vulnerable-twitter-api-leaves-millions-open-to-attack/148945/

WhiteHat Security. (2017, May 16). *Top 10 Vulnerabilities in Mobile Applications*. Retrieved from whitehatsec.com: https://www.whitehatsec.com/blog/top-10-vulnerabilities-in-mobile-applications/

Zetetic. (n.d). *SQLCipher*. Retrieved from zetetic.net: https://www.zetetic.net/sqlcipher/

Appendices

Appendix A – Code to encrypt a file, then read and write to the encrypted file

```
val keyAlias = MasterKeys.getOrCreate(MasterKeys.AES256_GCM_SPEC)
val secretFile = File(filesDir, "super_secret")
val encryptedFile = EncryptedFile.Builder(
    secretFile,
    applicationContext,
    keyAlias,
    FileEncryptionScheme.AES256_GCM_HKDF_4KB)
    .setKeysetAlias("file_key") // optional
    .setKeysetPrefName("secret_shared_prefs") // optional
    .build()

encryptedFile.openFileOutput().use { outputStream ->
    // Write data to your encrypted file
}

encryptedFile.openFileInput().use { inputStream ->
    // Read data from your encrypted file
}
```

- *secretFile*: A File object containing the file to be encrypted
- *keyAlias*: MasterKey object

Appendix B – iOS Encrypt & Decrypt

```
NSData *imageData = UIImagePNGRepresentation(image);
    NSString *imageName = [NSString stringWithFormat:@"image-
%d.securedData", self.photos.count + 1];
    NSData *encryptedImage = [RNEncryptor encryptData:imageData
withSettings:kRNCryptorAES256Settings password:@"A_SECRET_PASSWORD"
error:nil];
    [encryptedImage writeToFile:[self.filePath
stringByAppendingPathComponent:imageName] atomically:YES];
    [self.photos addObject:image];
    [self.collectionView reloadData];
    [picker dismissViewControllerAnimated:YES completion:nil];
```

```
if ([fileName rangeOfString:@".securedData"].length > 0) {
                NSData *data = [NSData
dataWithContentsOfFile:[self.filePath
stringByAppendingPathComponent:fileName]];
                NSData *decryptedData = [RNDecryptor decryptData:data
withSettings:kRNCryptorAES256Settings password:@"A_SECRET_PASSWORD"
error:nil];
                UIImage *image = [UIImage imageWithData:decryptedData];
                [self.photos addObject:image];
```