# Engineering Resilient Systems (Part 1)

Jonah McElfatrick
1700463

## ABSTRACT

The given case study indicates that the IT company in question hosts multiple different aspects that make up their infrastructure. It has been noted that the company provides their staff with a mobile application which can be installed on their own devices, in which allows staff to change their own account details. Details regarding the functionality and complexity of the mobile application have not been provided, therefore due to this, certain presumptions have been taken when writing this report. It is presumed that when the user wants to change their details, the application accesses and stores the account details data in a back-end database. Due to this, it is also presumed that the application includes a localised backup on the mobile device in order to allow for the user to update their details when not connected to the internet.

The following document provides information regarding application security with an analysis of the relevant class of vulnerabilities in relation to the mobile application in question. Also included is a walkthrough of one of the discussed vulnerabilities, with possible solutions to the vulnerability, in order to demonstrate the skills and knowledge required to carry out the exploit. Alongside the demonstration, a recommendation around a secure software development practice in which could be implemented in order to reduce the chances of an exploit or vulnerability, such as the one demonstrated, being introduced into the mobile application during the development cycle.

## 1. Introduction

A blog published by Codersera ('Top 7 Vulnerabilities In Android Applications 2020 -', 2019), indicated that more vulnerabilities are found in Android applications vs iOS applications, with the weight being 43% vs 38%. Of these vulnerabilities, 60% of them are client-side vulnerabilities, with 89% of these being able to be exploited remotely with no physical access to the device and 56% in which can be carried out without the need for a jailbroken or rooted device.

There are many different types of classifications that vulnerabilities can be categorized under. According to OWASP's Mobile Top 10 (*OWASP Top 10 Mobile Vulnerabilities Developers Need to Understand*, no date), the top class of mobile vulnerabilities come under the category of 'Improper Platform Usage' with the second class being 'Insecure Data Storage'. According to ptsecurity (*Vulnerabilities and threats in mobile applications, 2019*, no date), 76% of mobile applications include a vulnerability under the class of 'Insecure Data Storage', leaking information such as login credentials and personal information.

For businesses, having one of these vulnerabilities can lead to issues such as:

- Identity or credential theft
- Fraud
- Market reputation damage
- Poor customer relationships

Having any one of these issues could be very detrimental to any form of business and could cost the company a vast amount of money to repair any damages incurred (*OWASP Top 10 Mobile Vulnerabilities Developers Need to Understand*, 29 June 2020).

The average cost a company incurs due to the effects of a cyber-attack is approximately $200,000 according to a Hiscox 2019 report (*Hiscox UK | Business Insurance, Home Insurance & more*, 2019). For larger corporations, this may not be a lot of money, but this can cause other smaller businesses to start having financial issues or in some cases cause them to go into bankruptcy. To help prevent a security breach, developers can implement different methods into their development cycle in order to help improve the focus on application security.

## 2. Background

There are many different types of vulnerabilities that mobile applications can be susceptible to. Some of these vulnerabilities include storing data in ways that would be considered insecure. Mitre's CVE database (*CVE - Common Vulnerabilities and Exposures (CVE)*, 4 February 2021), provides a list of found vulnerabilities in all manner of software and hardware. Below are some examples of vulnerabilities found related to the class of 'Insecure Data Storage'.

### 2.1 CVE-2018-11544

This vulnerability is found in The Olive Tree FTP Server application v1.32 for Android which involves insecure storage of the 'prefUsername' and 'prefUserass' strings, stored in the

/data/data/com.theolivetree.ftpserver/shared_prefs/com.theolivetree.ftpserver_preferences.xml file. (*CVE - CVE-2018-11544*, 2018)
This vulnerability would allow for non-root users to be able to find and view the username and password.

### 2.2 CVE-2019-5625

This vulnerability is found in the Android application Halo Home before v1.11.0, where the OAuth authentication and refresh tokens are stored in a plain text file. The file is present until the user is logged out of the application and the device restarted (*CVE - CVE-2019-5625*, 2019).
This vulnerability would allow an attacker with physical access to the device to be able to impersonate the real user by using the stored tokens which in turn would allow then to be able to view and change the user's personal information in the application backend service.

### 2.3 CVE-2019-13099

This vulnerability was found in the Android application Momo v2.1.9 where the application stores sensitive data such as usernames and passwords in plaintext.
This vulnerability allows for a non-root user with the right knowledge to view the username and password of a valid user through the use of logcat (*CVE - CVE-2019-13099*, 2019).

### 2.4 CVE-2019-13100

This vulnerability is present in the Android application Send Anywhere v9.4.18 where sensitive information is stored in plaintext in the 'data/data/com.estmob.android.sendanywhere/shared_prefs/sendanywhere_device.xml' file (*CVE - CVE-2019-13100*, 2019).
This vulnerability would allow for non-root users to be able to find and view the username and password.

## 3. Exploit Implementation

Each of the CVE's described above all come under the OWASP heading of 'Improper Data Storage'. This allows anyone with the correct knowledge and understanding to be able to gain access to the stored data to view and possibly exploit it for malicious gain.
In order to gain a deeper understanding of how these exploits can be carried out, following is the procedure taken to exploit the CVE-2019-13100 vulnerability in the Send Anywhere Android application.

First an Android emulator was initialized and launched through Android Studio (*Download Android Studio and SDK tools | Android Developers*, 24 February 2021). The emulator used was a Nexus 5X running Android 7.1.1 Nougat.

From here, the Send Anywhere application (*Send Anywhere*, no date), v9.4.18 was installed onto the emulator and a user account was logged into the application.

Next ADB (*Android Debug Bridge (adb)*, 18 February 2021), was used to connect the emulator to a computer and a backup of the application was taken. The following commands were used to carry this out.

- ./adb devices
- ./adb backup -f sendanywhere.ab -noapk com.estmob.android.sendanywhere

When executing the './adb backup' command, a prompt is displayed on the emulator screen, this requires a password to be entered for the backup file, this password is then used to decrypt the backup file when extracting the information.

The next step was to transfer the backup to a Linux machine. This is to install the required tools to convert the sendanywhere.ab file to a tar file for extraction of the data. The tools, android-backup-extractor (abe) (Elenkov, 2021), and Maven (*Maven – Welcome to Apache Maven*, 25 February 2021), are used for this conversion. The following commands are used to install these tools:

- 'git clone https://github.com/nelenkov/android-backup-extractor.git'
- 'sudo apt install maven'

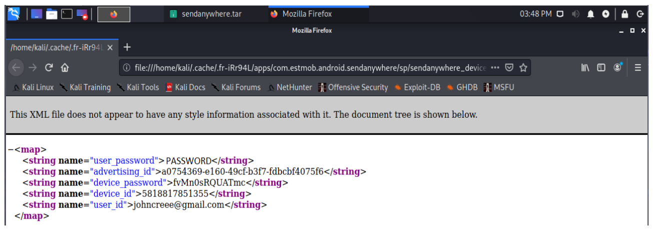
Once these commands have been carried out, the next step is to navigate to the cloned abe folder and compile the abe tool using the following commands:

- 'mvn clean package'

From here the sendanywhere.ab file can be converted to the required tar file using the following commands:

- 'java -jar target/abe.jar unpack sendanywhere.ab sendanywhere.tar “”'
- 'tar -xvf sendanywhere.tar'

Once converted, opening the tar folder and navigating to the directory '/apps/com.estmob.android.sendanywhere/sp/', and opening the 'sendanywhere_device.xml' file reveals the following screen.


file:///home/kali/.cache/.fr-iRr94L/apps/com.estmob.android.sendanywhere/sp/sendanywhere_device

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<map>
   <string name="user_password">PASSWORD</string>
   <string name="advertising_id">a0754369-e160-49cf-b3f7-fdbcbf4075f6</string>
   <string name="device_password">fvMn0sRQUATmc</string>
   <string name="device_id">5818817851355</string>
   <string name="user_id">johncreee@gmail.com</string>
 </map>
```

***Figure 1: sendanywhere_device.xml***

As can be seen in the figure above, the 'user_id' and 'user_password' field are stored in a non-encrypted xml file with the values displayed in plain text. This allows for anyone with the relevant skillset and physical access to the mobile device to gain access to the username/email and password of the user which could then allow them access to the Send Anywhere application. This could come with other unseen consequences as having access to the password for this account could give the attacker an indication of the type or trend of passwords that the user may use. This could potentially allow the attacker to gain access to other accounts using the same or similar credentials.

### 3.1 Mitigation

Possible solutions to the vulnerability shown above could include removing the localized storage of any sensitive data, hashing the data or encrypting the files at rest.
Removing the localized storage of this data would eliminate the issue of having sensitive data stored in plain text and would therefore eliminate this vulnerability. This however could come at the cost of functionality as removing localized storage could have negative effects regarding the features that the application provides.
Hashing the data before it is stored in the file prevents the issue from data being stored as plain text. Therefore, the data stored could not be read from and used for login details when required. Hashing algorithms although not reversible are not the strongest form of data protection. Hashes can be captured, and tools can then be used to brute force the hash quickly if not used in the correct manner.
Encryption of files that hold sensitive data for this application could allow for a more secure method of storage. This allows for the localized storage of the user details, which could be required for other features the application offers while maintaining device and account security. Both Googles Android and Apple's iOS platforms offer ways in which to encrypt and decrypt files through code to allow for a secure system of reading and writing to files. Google's 'Android Jetpack' library (*Android Developers*, 24 February 2021) and Apple's on disk encryption method (*Encrypting Your App's Files | Apple Developer Documentation*, 2021), provide the required features and functionality to be able to read and write to encrypted files.

## 4. Recommendation

A possible prevention method for the class of vulnerabilities described above could be to implement a secure code review before any developed piece of code is placed into a live production build. A secure code review is the process which involves manually and/or autonomously analyzing a piece of code or application for security vulnerabilities and bad practices. Both manual and automatic methods can include the use of scanners and/or tools to help indicate where vulnerabilities could be present in code.
A secure code review normally includes a checklist in order to understand what is being looked for in the review. An example of this is the OWASP Mobile App Security Checklist (*OWASP Mobile Security Testing Guide*, 2 October 2020). Alongside this a threat model must be defined in order to understand how the system could be attacked in order to prevent any vulnerabilities being present that could allow these attacks to be carried out.
Carrying out a secure code review is a relatively simple process in which could result in the prevention of multiple vulnerabilities, such as the ones noted above, from being introduced into a live application build, which in turn can save a company not only a lot of money but also prevent any negative reputational damage from the consumers.

### 4.1 Manual

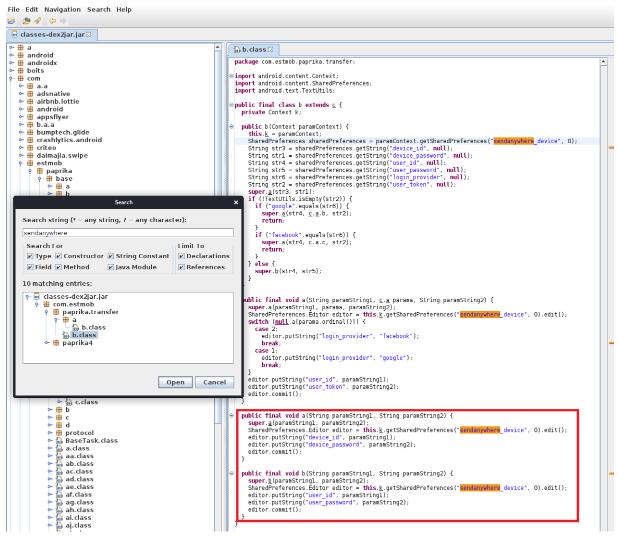
During a manual code review, the piece of code in question is manually analyzed, reviewed, and tested for different vulnerabilities or bad practices. This is mainly carried out by a human being with little help from some tools. To carry out a manual review on the Send Anywhere apk, first it was unzipped using the command:

- 'unzip Send\ Anywhere-Premium-v9.4.18_build_432096.apk -d SendAnywhere'

Then using the d2j dex2jar function (Pan, 2021), the classes.dex file is converted it to a jar file:

- 'd2j-dex2jar classes.dex'

The next step is to use the java decompiler graphical user interface (*Java Decompiler*, 25 December 2019), to open the jar file and view the source code. Due to having the previous knowledge of knowing what is vulnerable and to be found in this case, a search term can be used to find the file required. As can be seen in the figure below searching through the source code for the sendanywhere_device.xml file provides the result where, in the directory 'classes-dex2jar.jar/com.estmob/paprika.transfer/b.class', the functions for reading and writing the username and passwords to the file can be seen.



Search string (* = any string, ? = any character):
sendanywhere

Search For: ✓ Type ✓ Constructor ✓ String Constant ✓ Field ✓ Method ✓ Java Module
Limit To: ✓ Declarations ✓ References

10 matching entries:

```
classes-dex2jar.jar
  com.estmob
    paprika.transfer
      a
        b.class
      b.class
    paprika4
```

```
package com.estmob.paprika.transfer;

import android.content.Context;
import android.content.SharedPreferences;
import android.text.TextUtils;

public final class b extends c {
  private Context k;

  public b(Context paramContext) {
    this.k = paramContext;
    SharedPreferences sharedPreferences = paramContext.getSharedPreferences("sendanywhere_device", 0);
    String str3 = sharedPreferences.getString("device_id", null);
    String str1 = sharedPreferences.getString("device_password", null);
    String str4 = sharedPreferences.getString("user_id", null);
    String str5 = sharedPreferences.getString("user_password", null);
    String str6 = sharedPreferences.getString("login_provider", null);
    String str2 = sharedPreferences.getString("user_token", null);
    super.a(str3, str1);
    if (!TextUtils.isEmpty(str2)) {
      if ("google".equals(str6)) {
        super.a(str4, c.a.b, str2);
        return;
      }
      if ("facebook".equals(str6)) {
        super.a(str4, c.a.c, str2);
        return;
      }
    } else {
      super.b(str4, str5);
    }
  }

  public final void a(String paramString1, c.a parama, String paramString2) {
    super.a(paramString1, parama, paramString2);
    SharedPreferences.Editor editor = this.k.getSharedPreferences("sendanywhere_device", 0).edit();
    switch (null.a[parama.ordinal()]) {
      case 2:
        editor.putString("login_provider", "facebook");
        break;
      case 1:
        editor.putString("login_provider", "google");
        break;
    }
    editor.putString("user_id", paramString1);
    editor.putString("user_token", paramString2);
    editor.commit();
  }

  public final void a(String paramString1, String paramString2) {
    super.a(paramString1, paramString2);
    SharedPreferences.Editor editor = this.k.getSharedPreferences("sendanywhere_device", 0).edit();
    editor.putString("device_id", paramString1);
    editor.putString("device_password", paramString2);
    editor.commit();
  }

  public final void b(String paramString1, String paramString2) {
    super.b(paramString1, paramString2);
    SharedPreferences.Editor editor = this.k.getSharedPreferences("sendanywhere_device", 0).edit();
    editor.putString("user_id", paramString1);
    editor.putString("user_password", paramString2);
    editor.commit();
  }
}
```

***Figure 2: Vulnerable Code***

As can be seen in the figure above, the highlighted section in red shows the code used for writing the username and password to the file is not using any methods to store the credentials securely, it is simply writing the values of the 'device_id', 'device_password', 'user_id' and 'user_password' variables to the shared preferences sendanywhere_device file. There is no obscurification or protection in place to help secure the credentials from unauthorized users from gaining access and viewing them.

### 4.2 Automatic

During an autonomous code review, the piece of code in question is scanned and tested by scripts and tools. This saves on time and cost due to being much faster than a manual review as typically computers are faster than humans.
Examples of tools that could be used in an autonomous review could include Quixxi (*Mobile App Security Made Quick and Easy | Quixxisecurity*, 2021) or Mobile Security Framework (*MobSF/Mobile-Security-Framework-MobSF: Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.*, 27 February 2021).
The Quixxi website works by simply uploading the Send Anywhere apk to the website and running the automated scan. The MoSF works by using the following commands in a Linux terminal to be able to run the framework on a localized environment:

- 'git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git'
- 'cd Mobile-Security-Framework-MobSF'
- './setup.sh'
- './run.sh 127.0.0.1:8000'

From here, navigating to the address 127.0.0.1:8000 in a web browser and uploading the apk file to the web interface, a vulnerability scan can be initiated.
The Send Anywhere application was scanned using both of the tools above, the results can be seen in appendix A for the Quixxi results and appendix B for the MobSF results.
As can be seen from both autonomous scans, multiple classes of vulnerabilities were found with the Send Anywhere application, including the class specified in this report, Insecure Data Storage. Depending on how many and the class of the found vulnerabilities, scanners can sometimes give recommendations on actions to take in order to help resolve these issues.

### 4.3 Mitigation

After a secure code review and the source of the vulnerability has been identified, alteration to the code as mentioned above in section 3.1, implementing file encryption can allow the application to maintain full functionality while keeping information stored securely and locally on the device. This can be carried out by using the cryptographic library provided by Google for Android developers. This includes common functions such as reading and writing to encrypted files using the AES (Advance Encryption Standard) method. Included in appendix C is the functions provided by Google for reading and writing the user credentials to an encrypted file. These functions could be implemented into the send anywhere application in order to allow for localized secure storage of user credentials. Another option would be to remove the functions entirely and to no longer store user credentials locally on the device.

## 5. Limitations & Challenges

Implementing a secure code review can help to solve issues before they are turned into possible exploits or vulnerabilities in a live build. However, while implementing a secure code review can help in some cases, there are other aspects that should be considered before choosing the right kind of review to implement into the production lifecycle.

### 5.1 Manual

Manual code reviews can sometimes pick up vulnerabilities or issues that a scanner may not since scanners use pattern matching and database lookup techniques when searching for vulnerabilities. Although a manual code review could possibly find issues that an autonomous review may miss, due to it being a manual review by human beings, it can take a considerably longer length of time to review the same amount of code. This longer length of time can cause issues in the development lifecycle as to pass a code review, certain criteria must be met and then once reviewed, implemented into the application in question which then takes more time. This can cause delays in production time and therefore the extra time required would need to be taken into consideration at the start of the project and a buffer put in place in case of any issues that are encountered. Due manual reviews taking a longer length of time, this also usually means that it costs more to implement. This is due to the number of working hours that is required to be put into the review itself, although this may differ depending on the type of autonomous tools and scanners used.

### 5.2 Autonomous

An autonomous code review can be much faster than a manual review, this however can come with its own consequences. Autonomous code reviews can report not only vulnerabilities present in the code, but can also report false positives, where there is a report of something being wrong when there is nothing wrong, or even false negatives, where nothings is reported as wrong when there is something wrong. This then requires a user to check the review itself and ensure that what is reported should or should not be acted upon, which in turn can take an increasing amount of time depending on the size of the code review being carried out.

## 6. Conclusion

Implementing a secure code review into the development lifecycle could decrease the chances of a known or unknown vulnerability making it into the live production build of an application.

A secure code review implemented into the development lifecycle of the Send Anywhere application shown in this paper, could have identified the demonstrated vulnerability before it was released to the public.

Secure code reviews are easy to implement into the production cycle with minimal training being required for developers. Secure software reviews can be used in conjunction with other secure development practices in order to gain a further level of validation.

## References

'Top 7 Vulnerabilities In Android Applications 2020 -' (2019), 20 September. Available at: https://codersera.com/blog/top-7-vulnerabilities-in-android-applications-2019/ (Accessed: 16 February 2021).

*OWASP Mobile Top 10* (2016). Available at: https://owasp.org/www-project-mobile-top-10/ (Accessed: 23 February 2021).

*Vulnerabilities and threats in mobile applications, 2019* (no date). Available at: https://www.ptsecurity.com/ww-en/analytics/mobile-application-security-threats-and-vulnerabilities-2019/ (Accessed: 6 March 2021).

*WASP Top 10 Mobile Vulnerabilities Developers Need to Understand* (29 June 2020). Available at: https://www.cypressdefense.com/blog/owasp-mobile-top-10-vulnerabilities/ (Accessed: 23 February 2021).

*Business Insurance | Hiscox* (2019). Available at: https://www.hiscox.com/ (Accessed: 28 February 2021).

*CVE - Common Vulnerabilities and Exposures (CVE)* (February 4th, 2021). Available at: https://cve.mitre.org/ (Accessed: 18 February 2021).

*CVE - CVE-2018-11544* (2018). Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-11544 (Accessed: 16 February 2021).

*CVE - CVE-2019-5625* (2019). Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-5625 (Accessed: 16 February 2021).

*CVE - CVE-2019-13099* (2019). Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-13099 (Accessed: 16 February 2021).

*CVE - CVE-2019-13100* (2019). Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-13100 (Accessed: 16 February 2021).

*Download Android Studio and SDK tools | Android Developers* (24 February 2021). Available at: https://developer.android.com/studio (Accessed: 20 February 2021).

*Send Anywhere - File transfer* (no date). Available at: https://send-anywhere.com/ (Accessed: 23 February 2021).

*Android Debug Bridge (adb)* (18 February 2021) *Android Developers*. Available at: https://developer.android.com/studio/command-line/adb (Accessed: 20 February 2021).

Elenkov, N. (2021) *nelenkov/android-backup-extractor*. Available at: https://github.com/nelenkov/android-backup-extractor (Accessed: 19 February 2021).

*Maven – Welcome to Apache Maven* (25 February 2021). Available at: https://maven.apache.org/ (Accessed: 20 February 2021).

*Android Developers* (24 February 2021) *Android Developers*. Available at: https://developer.android.com/topic/security/data (Accessed: 18 February 2021).

*Encrypting Your App's Files | Apple Developer Documentation* (2021). Available at: https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encryptin g_your_app_s_files (Accessed: 28 February 2021).

*OWASP Mobile Security Testing Guide* (02 October 2020). Available at: https://owasp.org/www-project-mobile-security-testing-guide/ (Accessed: 14 March 2021).

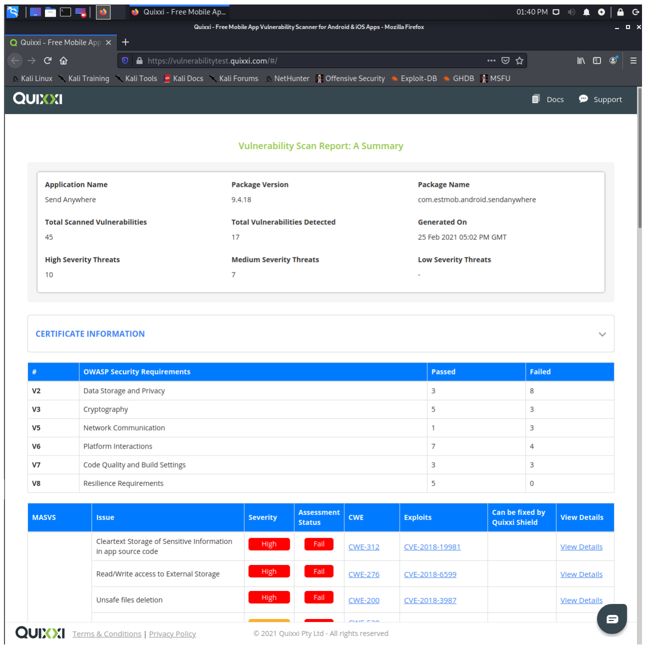
Pan, B. (2021) *pxb1988/dex2jar*. Available at: https://github.com/pxb1988/dex2jar (Accessed: 26 February 2021).

*Java Decompiler* (25 December 2019). Available at: http://java-decompiler.github.io/ (Accessed: 26 February 2021).

*Mobile App Security Made Quick and Easy | Quixxisecurity* (2021) *Quixxi Security*. Available at: https://quixxisecurity.com/ (Accessed: 27 February 2021).

*MobSF/Mobile-Security-Framework-MobSF: Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.* (no date). Available at: https://github.com/MobSF/Mobile-Security-Framework-MobSF (Accessed: 27 February 2021).

# Appendix A – Quixxi



Quixxi - Free Mobile App Vulnerability Scanner for Android & iOS Apps - Mozilla Firefox

https://vulnerabilitytest.quixxi.com/#/

Vulnerability Scan Report: A Summary

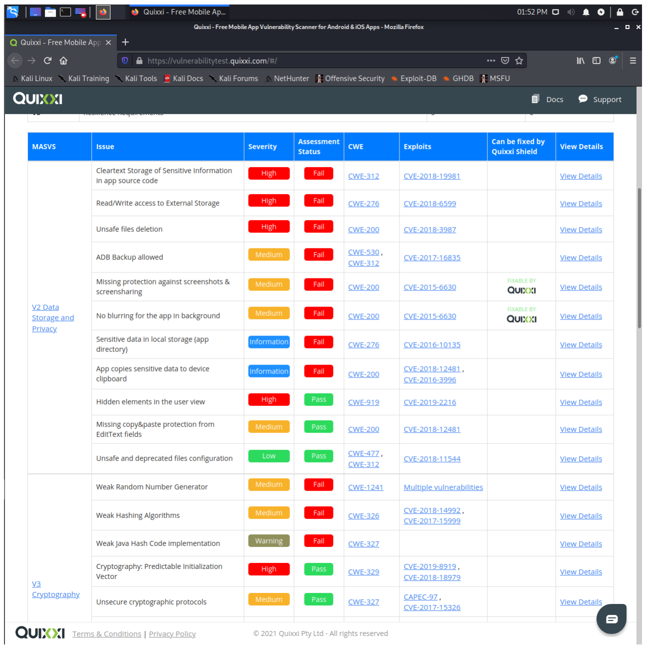
| | | |
|---|---|---|
| **Application Name** | **Package Version** | **Package Name** |
| Send Anywhere | 9.4.18 | com.estmob.android.sendanywhere |
| **Total Scanned Vulnerabilities** | **Total Vulnerabilities Detected** | **Generated On** |
| 45 | 17 | 25 Feb 2021 05:02 PM GMT |
| **High Severity Threats** | **Medium Severity Threats** | **Low Severity Threats** |
| 10 | 7 | - |

CERTIFICATE INFORMATION

| # | OWASP Security Requirements | Passed | Failed |
|---|---|---|---|
| **V2** | Data Storage and Privacy | 3 | 8 |
| **V3** | Cryptography | 5 | 3 |
| **V5** | Network Communication | 1 | 3 |
| **V6** | Platform Interactions | 7 | 4 |
| **V7** | Code Quality and Build Settings | 3 | 3 |
| **V8** | Resilience Requirements | 5 | 0 |

| MASVS | Issue | Severity | Assessment Status | CWE | Exploits | Can be fixed by Quixxi Shield | View Details |
|---|---|---|---|---|---|---|---|
| | Cleartext Storage of Sensitive Information in app source code | High | Fail | CWE-312 | CVE-2018-19981 | | View Details |
| | Read/Write access to External Storage | High | Fail | CWE-276 | CVE-2018-6599 | | View Details |
| | Unsafe files deletion | High | Fail | CWE-200 | CVE-2018-3987 | | View Details |

Terms & Conditions | Privacy Policy © 2021 Quixxi Pty Ltd - All rights reserved

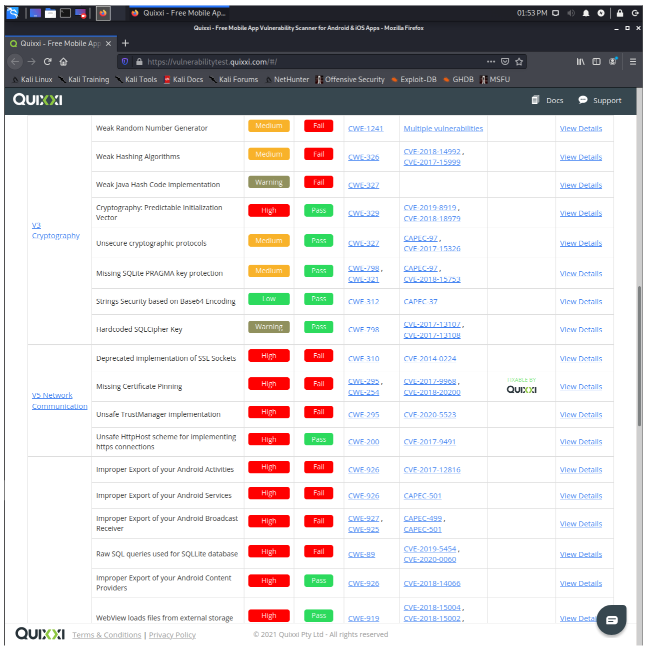


Quixxi - Free Mobile App Vulnerability Scanner for Android & iOS Apps - Mozilla Firefox

https://vulnerabilitytest.quixxi.com/#/

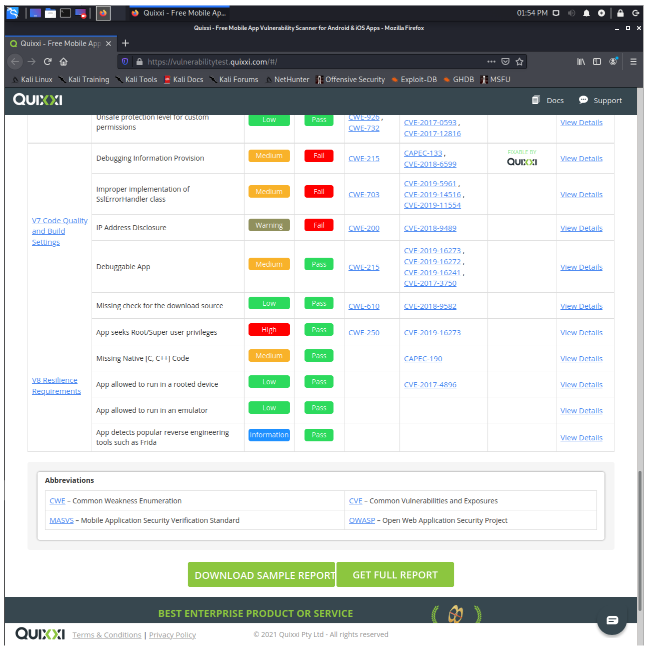
| MASVS | Issue | Severity | Assessment Status | CWE | Exploits | Can be fixed by Quixxi Shield | View Details |
|---|---|---|---|---|---|---|---|
| V2 Data Storage and Privacy | Cleartext Storage of Sensitive Information in app source code | High | Fail | CWE-312 | CVE-2018-19981 | | View Details |
| | Read/Write access to External Storage | High | Fail | CWE-276 | CVE-2018-6599 | | View Details |
| | Unsafe files deletion | High | Fail | CWE-200 | CVE-2018-3987 | | View Details |
| | ADB Backup allowed | Medium | Fail | CWE-530 , CWE-312 | CVE-2017-16835 | | View Details |
| | Missing protection against screenshots & screensharing | Medium | Fail | CWE-200 | CVE-2015-6630 | FIXABLE BY QUIXXI | View Details |
| | No blurring for the app in background | Medium | Fail | CWE-200 | CVE-2015-6630 | FIXABLE BY QUIXXI | View Details |
| | Sensitive data in local storage (app directory) | Information | Fail | CWE-276 | CVE-2016-10135 | | View Details |
| | App copies sensitive data to device clipboard | Information | Fail | CWE-200 | CVE-2018-12481 , CVE-2016-3996 | | View Details |
| | Hidden elements in the user view | High | Pass | CWE-919 | CVE-2019-2216 | | View Details |
| | Missing copy&paste protection from EditText fields | Medium | Pass | CWE-200 | CVE-2018-12481 | | View Details |
| | Unsafe and deprecated files configuration | Low | Pass | CWE-477 , CWE-312 | CVE-2018-11544 | | View Details |
| V3 Cryptography | Weak Random Number Generator | Medium | Fail | CWE-1241 | Multiple vulnerabilities | | View Details |
| | Weak Hashing Algorithms | Medium | Fail | CWE-326 | CVE-2018-14992 , CVE-2017-15999 | | View Details |
| | Weak Java Hash Code implementation | Warning | Fail | CWE-327 | | | View Details |
| | Cryptography: Predictable Initialization Vector | High | Pass | CWE-329 | CVE-2019-8919 , CVE-2018-18979 | | View Details |
| | Unsecure cryptographic protocols | Medium | Pass | CWE-327 | CAPEC-97 , CVE-2017-15326 | | View Details |

Terms & Conditions | Privacy Policy © 2021 Quixxi Pty Ltd - All rights reserved


Quixxi - Free Mobile App Vulnerability Scanner for Android & iOS Apps - Mozilla Firefox
https://vulnerabilitytest.quixxi.com/#/
Docs
Support

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| V3 Cryptography | Weak Random Number Generator | Medium | Fail | CWE-1241 | Multiple vulnerabilities | | View Details |
| | Weak Hashing Algorithms | Medium | Fail | CWE-326 | CVE-2018-14992, CVE-2017-15999 | | View Details |
| | Weak Java Hash Code implementation | Warning | Fail | CWE-327 | | | View Details |
| | Cryptography: Predictable Initialization Vector | High | Pass | CWE-329 | CVE-2019-8919, CVE-2018-18979 | | View Details |
| | Unsecure cryptographic protocols | Medium | Pass | CWE-327 | CAPEC-97, CVE-2017-15326 | | View Details |
| | Missing SQLite PRAGMA key protection | Medium | Pass | CWE-798, CWE-321 | CAPEC-97, CVE-2018-15753 | | View Details |
| | Strings Security based on Base64 Encoding | Low | Pass | CWE-312 | CAPEC-37 | | View Details |
| | Hardcoded SQLCipher Key | Warning | Pass | CWE-798 | CVE-2017-13107, CVE-2017-13108 | | View Details |
| V5 Network Communication | Deprecated implementation of SSL Sockets | High | Fail | CWE-310 | CVE-2014-0224 | | View Details |
| | Missing Certificate Pinning | High | Fail | CWE-295, CWE-254 | CVE-2017-9968, CVE-2018-20200 | FIXABLE BY QUIXXI | View Details |
| | Unsafe TrustManager implementation | High | Fail | CWE-295 | CVE-2020-5523 | | View Details |
| | Unsafe HttpHost scheme for implementing https connections | High | Pass | CWE-200 | CVE-2017-9491 | | View Details |
| | Improper Export of your Android Activities | High | Fail | CWE-926 | CVE-2017-12816 | | View Details |
| | Improper Export of your Android Services | High | Fail | CWE-926 | CAPEC-501 | | View Details |
| | Improper Export of your Android Broadcast Receiver | High | Fail | CWE-927, CWE-925 | CAPEC-499, CAPEC-501 | | View Details |
| | Raw SQL queries used for SQLLite database | High | Fail | CWE-89 | CVE-2019-5454, CVE-2020-0060 | | View Details |
| | Improper Export of your Android Content Providers | High | Pass | CWE-926 | CVE-2018-14066 | | View Details |
| | WebView loads files from external storage | High | Pass | CWE-919 | CVE-2018-15004, CVE-2018-15002, | | View Details |

QUIXXI Terms & Conditions | Privacy Policy © 2021 Quixxi Pty Ltd - All rights reserved

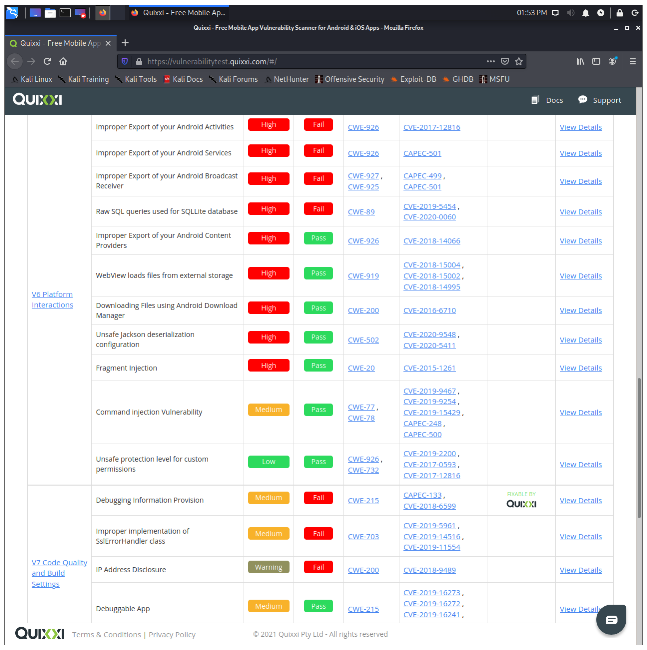

Quixxi - Free Mobile App Vulnerability Scanner for Android & iOS Apps - Mozilla Firefox
https://vulnerabilitytest.quixxi.com/#/
Docs
Support
DOWNLOAD SAMPLE REPORT
GET FULL REPORT
BEST ENTERPRISE PRODUCT OR SERVICE

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Unsafe protection level for custom permissions | Low | Pass | CWE-926, CWE-732 | CVE-2017-0593, CVE-2017-12816 | | View Details |
| V7 Code Quality and Build Settings | Debugging Information Provision | Medium | Fail | CWE-215 | CAPEC-133, CVE-2018-6599 | FIXABLE BY QUIXXI | View Details |
| | Improper implementation of SslErrorHandler class | Medium | Fail | CWE-703 | CVE-2019-5961, CVE-2019-14516, CVE-2019-11554 | | View Details |
| | IP Address Disclosure | Warning | Fail | CWE-200 | CVE-2018-9489 | | View Details |
| | Debuggable App | Medium | Pass | CWE-215 | CVE-2019-16273, CVE-2019-16272, CVE-2019-16241, CVE-2017-3750 | | View Details |
| | Missing check for the download source | Low | Pass | CWE-610 | CVE-2018-9582 | | View Details |
| V8 Resilience Requirements | App seeks Root/Super user privileges | High | Pass | CWE-250 | CVE-2019-16273 | | View Details |
| | Missing Native [C, C++] Code | Medium | Pass | | CAPEC-190 | | View Details |
| | App allowed to run in a rooted device | Low | Pass | | CVE-2017-4896 | | View Details |
| | App allowed to run in an emulator | Low | Pass | | | | View Details |
| | App detects popular reverse engineering tools such as Frida | Information | Pass | | | | View Details |

**Abbreviations**

| | |
|---|---|
| CWE – Common Weakness Enumeration | CVE – Common Vulnerabilities and Exposures |
| MASVS – Mobile Application Security Verification Standard | OWASP – Open Web Application Security Project |

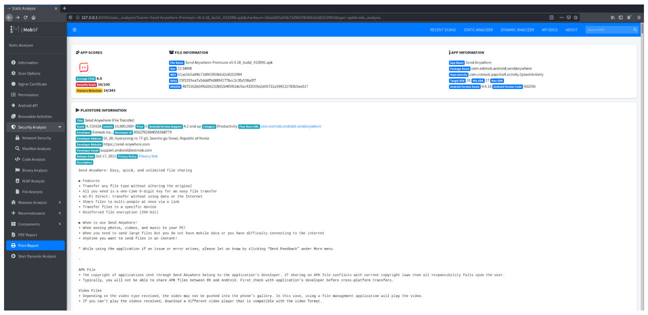
QUIXXI Terms & Conditions | Privacy Policy © 2021 Quixxi Pty Ltd - All rights reserved


Quixxi - Free Mobile App Vulnerability Scanner for Android & iOS Apps - Mozilla Firefox
https://vulnerabilitytest.quixxi.com/#/
Docs
Support

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| V6 Platform Interactions | Improper Export of your Android Activities | High | Fail | CWE-926 | CVE-2017-12816 | | View Details |
| | Improper Export of your Android Services | High | Fail | CWE-926 | CAPEC-501 | | View Details |
| | Improper Export of your Android Broadcast Receiver | High | Fail | CWE-927, CWE-925 | CAPEC-499, CAPEC-501 | | View Details |
| | Raw SQL queries used for SQLLite database | High | Fail | CWE-89 | CVE-2019-5454, CVE-2020-0060 | | View Details |
| | Improper Export of your Android Content Providers | High | Pass | CWE-926 | CVE-2018-14066 | | View Details |
| | WebView loads files from external storage | High | Pass | CWE-919 | CVE-2018-15004, CVE-2018-15002, CVE-2018-14995 | | View Details |
| | Downloading Files using Android Download Manager | High | Pass | CWE-200 | CVE-2016-6710 | | View Details |
| | Unsafe Jackson deserialization configuration | High | Pass | CWE-502 | CVE-2020-9548, CVE-2020-5411 | | View Details |
| | Fragment Injection | High | Pass | CWE-20 | CVE-2015-1261 | | View Details |
| | Command injection Vulnerability | Medium | Pass | CWE-77, CWE-78 | CVE-2019-9467, CVE-2019-9254, CVE-2019-15429, CAPEC-248, CAPEC-500 | | View Details |
| | Unsafe protection level for custom permissions | Low | Pass | CWE-926, CWE-732 | CVE-2019-2200, CVE-2017-0593, CVE-2017-12816 | | View Details |
| V7 Code Quality and Build Settings | Debugging Information Provision | Medium | Fail | CWE-215 | CAPEC-133, CVE-2018-6599 | FIXABLE BY QUIXXI | View Details |
| | Improper implementation of SslErrorHandler class | Medium | Fail | CWE-703 | CVE-2019-5961, CVE-2019-14516, CVE-2019-11554 | | View Details |
| | IP Address Disclosure | Warning | Fail | CWE-200 | CVE-2018-9489 | | View Details |
| | Debuggable App | Medium | Pass | CWE-215 | CVE-2019-16273, CVE-2019-16272, CVE-2019-16241, | | View Details |

QUIXXI Terms & Conditions | Privacy Policy © 2021 Quixxi Pty Ltd - All rights reserved

## Appendix B - MobSF



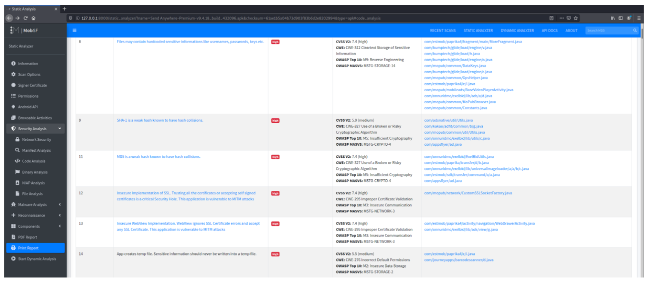
**APP SCORES**
Average CVSS 6.5
Security Score 10/100
Trackers Detection 14/343

**FILE INFORMATION**
File Name Send Anywhere-Premium-v9.4.18_build_432096.apk
Size 22.54MB
MD5 61ae1b5a04b73d903f83b6d2e8202994
SHA1 25f1295ea7a5dddf9d8894177bcc2c3fa538a0f7
SHA256 4075362b04fb206233002b4f5f83dc9ac433059e2d06731a594132783b9ae517

**APP INFORMATION**
App Name Send Anywhere
Package Name com.estmob.android.sendanywhere
Main Activity com.estmob.paprika4.activity.SplashActivity
Target SDK 26 Min SDK 17 Max SDK
Android Version Name 9.4.18 Android Version Code 432096

**PLAYSTORE INFORMATION**
Title Send Anywhere (File Transfer)
Score 4.726928 Installs 10,000,000+ Price 0 Android Version Support 4.2 and up Category Productivity Play Store URL com.estmob.android.sendanywhere
Developer Estmob Inc., Developer ID 8562762884556568779
Developer Address 5F, 28, Hyoryeong-ro 77-gil, Seocho-gu Seoul, Republic of Korea
Developer Website https://send-anywhere.com
Developer Email support.android@estmob.com
Release Date Oct 17, 2012 Privacy Policy Privacy link
Description

```
Send Anywhere: Easy, quick, and unlimited file sharing

▶ Features
• Transfer any file type without altering the original
• All you need is a one-time 6-digit key for an easy file transfer
• Wi-Fi Direct: transfer without using data or the Internet
• Share files to multi-people at once via a link
• Transfer files to a specific device
• Reinforced file encryption (256-bit)

▶ When to use Send Anywhere!
• When moving photos, videos, and music to your PC!
• When you need to send large files but you do not have mobile data or you have difficulty connecting to the internet
• Anytime you want to send files in an instant!

* While using the application if an issue or error arises, please let us know by clicking "Send Feedback" under More menu

-

APK File
• The copyright of applications sent through Send Anywhere belong to the application's developer. If sharing an APK file conflicts with current copyright laws then all responsibility falls upon the user.
• Typically, you will not be able to share APK files between OS and Android. First check with application's developer before cross-platform transfers.

Video Files
• Depending on the video type received, the video may not be pushed into the phone's gallery. In this case, using a file management application will play the video.
• If you can't play the videos received, download a different video player that is compatible with the video format.
```


127.0.0.1:8000/static_analyzer/?name=Send Anywhere-Premium-v9.4.18_build_432096.apk&checksum=61ae1b5a04b73d903f83b6d2e8202994&type=apk#code_analysis
MobSF
RECENT SCANS
STATIC ANALYZER
DYNAMIC ANALYZER
API DOCS
ABOUT

| | | | | |
|---|---|---|---|---|
| 8 | Files may contain hardcoded sensitive informations like usernames, passwords, keys etc. | high | **CVSS V2:** 7.4 (high) **CWE:** CWE-312 Cleartext Storage of Sensitive Information **OWASP Top 10:** M9: Reverse Engineering **OWASP MASVS:** MSTG-STORAGE-14 | com/estmob/paprika4/fragment/main/MoreFragment.java com/bumptech/glide/load/engine/v.java com/bumptech/glide/load/h.java com/bumptech/glide/load/engine/o.java com/mopub/common/DataKeys.java com/bumptech/glide/load/engine/c.java com/mopub/common/GpsHelper.java com/estmob/paprika4/e/i.java com/mopub/mobileads/BaseVideoPlayerActivity.java com/onnuridmc/exelbid/lib/ads/a/d.java com/mopub/common/MoPubBrowser.java com/mopub/common/Constants.java |
| 9 | SHA-1 is a weak hash known to have hash collisions. | high | **CVSS V2:** 5.9 (medium) **CWE:** CWE-327 Use of a Broken or Risky Cryptographic Algorithm **OWASP Top 10:** M5: Insufficient Cryptography **OWASP MASVS:** MSTG-CRYPTO-4 | com/adsnative/util/Utils.java com/kakao/adfit/common/b/g.java com/mopub/common/util/Utils.java com/onnuridmc/exelbid/lib/utils/c.java com/appsflyer/ad.java |
| 11 | MD5 is a weak hash known to have hash collisions. | high | **CVSS V2:** 7.4 (high) **CWE:** CWE-327 Use of a Broken or Risky Cryptographic Algorithm **OWASP Top 10:** M5: Insufficient Cryptography **OWASP MASVS:** MSTG-CRYPTO-4 | com/onnuridmc/exelbid/ExelBidUtils.java com/estmob/paprika/transfer/d/b.java com/onnuridmc/exelbid/lib/universalimageloader/a/a/b/c.java com/estmob/sdk/transfer/command/a/a.java com/appsflyer/ad.java |
| 12 | Insecure Implementation of SSL. Trusting all the certificates or accepting self signed certificates is a critical Security Hole. This application is vulnerable to MITM attacks | high | **CVSS V2:** 7.4 (high) **CWE:** CWE-295 Improper Certificate Validation **OWASP Top 10:** M3: Insecure Communication **OWASP MASVS:** MSTG-NETWORK-3 | com/mopub/network/CustomSSLSocketFactory.java |
| 13 | Insecure WebView Implementation. WebView ignores SSL Certificate errors and accept any SSL Certificate. This application is vulnerable to MITM attacks | high | **CVSS V2:** 7.4 (high) **CWE:** CWE-295 Improper Certificate Validation **OWASP Top 10:** M3: Insecure Communication **OWASP MASVS:** MSTG-NETWORK-3 | com/estmob/paprika4/activity/navigation/WebDrawerActivity.java com/onnuridmc/exelbid/lib/ads/view/g.java |
| 14 | App creates temp file. Sensitive information should never be written into a temp file. | high | **CVSS V2:** 5.5 (medium) **CWE:** CWE-276 Incorrect Default Permissions **OWASP Top 10:** M2: Insecure Data Storage **OWASP MASVS:** MSTG-STORAGE-2 | com/estmob/paprika4/c/i.java com/journeyapps/barcodescanner/d.java |