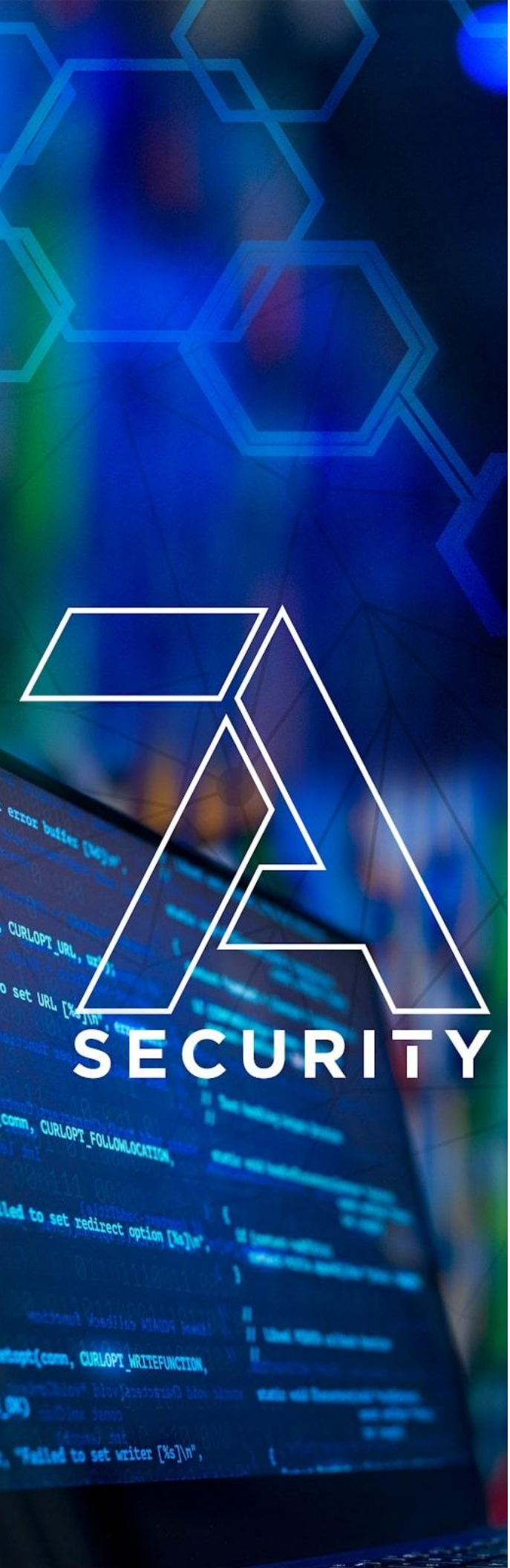

Test Targets:

LeaveHomeSafe Android app

LeaveHomeSafe iOS app

Pentest Report

Client:

Hong Kong Democracy Council

7ASecurity Test Team:

- Abraham Aranguren, MSc.
- Various 7ASecurity members

7ASecurity

Protect Your Site & Apps From Attackers

sales@7asecurity.com

INDEX

Introduction

"Let's Fight the Virus! Scan with "LeaveHomeSafe"[...]
help protect the community while protecting your privacy."
From: https://www.leavehomesafe.gov.hk/en/

This report outlines the results of a blackbox penetration test and a privacy audit conducted against the *LeaveHomeSafe* solution. The work was requested by the *Hong Kong Democracy Council* (HKDC), funded by the *Open Technology Fund (OTF)*, and carried out by *7ASecurity* in April and May 2022. A total of 17 days were invested to reach the coverage expected for this project.

The background of this project is to address the general concern about the potential security and privacy risks that might be introduced by the *LeaveHomeSafe* Android and iOS applications[1][2]. Please note that this *COVID-19* digital contact tracing application is mandated in all government venues, hospitals, markets, shopping malls, supermarkets and places of worship, among other places in Hong Kong at the time of writing[3][4].

The goal of this engagement was therefore to have an independent third party verify whether the official *LeaveHomeSafe* privacy and security claims are accurate. *7ASecurity* downloaded and reviewed the official *LeaveHomeSafe* Android (3.1.0, 3.2.0, 3.2.3) and iOS (3.1.0, 3.2.0, 3.2.3) applications, which were the latest available during this assignment.

The methodology implemented was *blackbox*. The *7ASecurity* team had no access to test users, documentation or source code. The lack of *Hong Kong Health Code System* credentials, valid vaccination status QR codes and valid COVID testing status QR codes were major limitations during this assignment, as the testers could not fully exercise the application logic in those functional areas. All efforts were therefore focused on reverse engineering, decompiling the applications and analyzing their behavior at runtime. A team of 4 senior testers was assigned for the preparation, execution and finalization of this project.

All preparations were done in March and April of 2022, ahead of the test, to ensure a smooth start for the *7ASecurity* team. Communications during the test were done using a shared *Signal* chat group. The *Hong Kong Democracy Council* (HKDC) and *OTF* were helpful and responsive throughout the audit.

[1] https://www.bloomberg.com/news/articles/2022-02-21/hong-kong-s-contact-tracing-app-...
[2] https://www.scmp.com/news/hong-kong/health-environment/article/3166508/tracking-function-...
[3] https://en.wikipedia.org/wiki/LeaveHomeSafe#Mandatory_use
[4] https://zh.wikipedia.org/wiki/...

Communications were smooth and only several questions had to be asked. The scope was well prepared and clear, with no noteworthy roadblocks encountered during the test. *7ASecurity* gave frequent status updates about the test and the related findings.

This exercise split the scope items in the following work packages, which are referenced in the ticket headlines as applicable:

- WP1 - Mobile Security tests against *LeaveHomeSafe* Android & iOS apps
- WP2 - Privacy tests against *LeaveHomeSafe* Android & iOS apps

Regarding the security audit (WP1), the team acquired adequate coverage over the scope items and managed to spot a total of 12 findings, 8 of which were classified as security vulnerabilities and 4 as general weaknesses with lower exploitation potential. Please note that 3 of the findings in this report had an estimated severity level of *high* or *critical*. This poor result strongly suggests that the *LeaveHomeSafe* mobile apps have not been audited by any competent security firm previously. This is in stark contrast to the documentation in the official *LeaveHomeSafe* website, which indicates the mobile applications were audited previously on December 10th 2021[5], without any significant findings.

Regarding the privacy audit (WP2), *7ASecurity* directly answers 13 privacy-related questions with a confidence level ranging from *Unclear* to *Proven*. These are described in the Privacy Analysis Findings section of this report.

The report sheds further light on the scope and test setup as well as the available material for testing. It subsequently lists all findings in chronological order beginning with the vulnerabilities found and then the general weaknesses discovered in this test.

Each finding is accompanied by a technical description, a proof-of-concept (*PoC*) where possible, as well as mitigation or fix advice. *7ASecurity* closes the report by elaborating on the general impressions gained throughout this test and shares some views on the perceived security and privacy posture of the scope that is *LeaveHomeSafe*.

[5] https://www.leavehomesafe.gov.hk/en/technical-document/

Scope

The following list outlines the items in scope for this project:

WP1 - Mobile Security tests against *LeaveHomeSafe* Android & iOS apps

- **Android (versions 3.1.0, 3.2.0, 3.2.3):**
  - https://play.google.com/store/apps/details?id=hk.gov..leavehomesafe
  - https://www.leavehomesafe.gov.hk/en/download/
  - https://appgallery.huawei.com/#/app/C103081261
- **iOS (versions 3.1.0, 3.2.0, 3.2.3):**
  - https://apps.apple.com/app/leavehomesafe/id1536377801

WP2 - Privacy tests against *LeaveHomeSafe* Android & iOS apps

- As above

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation flaws identified during the testing period. Please note that the findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability has additionally been given a unique identifier (e.g. *LHS-01-001*) for the purpose of facilitating any future follow-up correspondence.

LHS-01-001 WP1: MitM without Warnings via invalid TLS Certificates *(**Critical**)*

It was found that the latest *LeaveHomeSafe* Android app at the time of writing (3.2.3) fails to validate TLS certificates correctly, which allows MitM attacks without any user warnings. This issue was verified on a non-rooted *Google Nexus 5* device running *Android 6.0.1* (i.e. The last update available for this device), other supported devices are likely affected. Please note that the iOS app is not affected by this issue. A malicious attacker, with a valid domain name and able to manipulate network communications (i.e. public Wi-Fi without guest isolation), could leverage this weakness to intercept traffic between the *LeaveHomeSafe* application and its backend servers. For example, an attacker could intercept the login to the *Hong Kong Health Code System* and gain access to the *Hong Kong Identity Card ID* and password of the user. Other possibilities include obtaining the personal *One Time Password (OTP)* provided by the Hong Kong

Centre for Health Protection (CHP) as well as intercepting user-reported COVID infections. This issue was confirmed as follows:

Step 1: Configure MitM using CA-signed certificates for an invalid hostname

The Android device HTTP proxy settings were first changed to point to a test proxy server, with capabilities to craft CA-signed certificates trusted by the device.

The proxy server configuration was then amended to always use certificates signed for *7asecurity.com*, regardless of the inbound host header in the request:

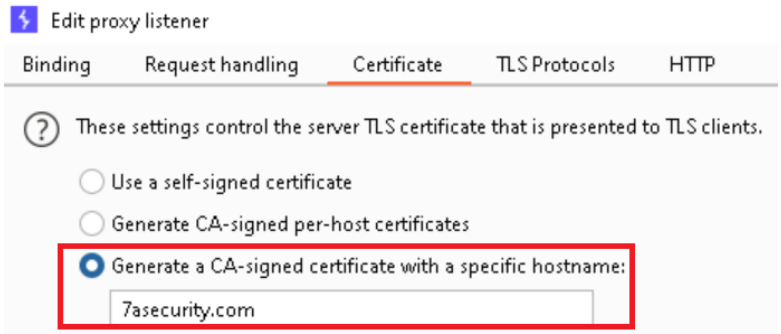

Fig.: Proxy settings for CA-signed certificates with a hostname of 7asecurity.com

The above configuration simulates a malicious attacker, able to supply a valid certificate for *7asecurity.com* to TLS clients. This configuration is invalid and should result in security warnings for any TLS connection attempt to any host that is not *7asecurity.com*.

Step 2: Verify the Android browser shows Security Warnings

The above setup supplies CA-signed certificates for *7asecurity.com* to all TLS clients, regardless of the hostname they attempt to connect to. Appropriate TLS validation should reject such certificate, which can be verified in the Android browser as follows:

Run the following ADB Command:

ADB Command:

```
adb shell am start -a "android.intent.action.VIEW" -d https://www.leavehomesafe.gov.hk
```

The following security warning appears:

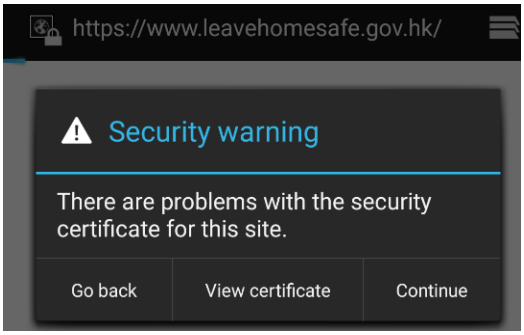

Fig.: The Android browser shows security warnings, as expected

Step 3: Confirm the complete lack of warnings in *LeaveHomeSafe*

Open the *LeaveHomeSafe* application, and try to login to the *Hong Kong Health Code System*, use any randomly generated HKID, and any random password:

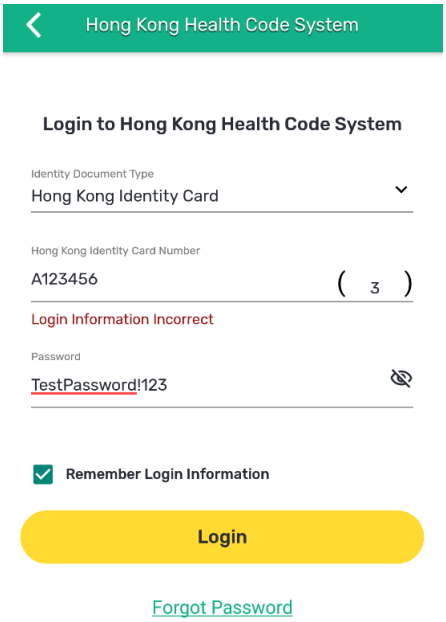

Fig.: Attempt to login with invalid credentials

Observe the captured login credentials without user warnings:

Captured HTTP Request:

```
POST /lhsapi/loginV2 HTTP/1.1
Host: apply.ehc.gov.hk
Accept: application/json, text/plain, */*
```

```
Content-Type: application/json;charset=utf-8
Content-Length: 197
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.12.1
Connection: close

{"docType":"0","docNum":"A1234563","docCountryCode":"HKG","hashId":"3600dd5f-9d40-414a
-b239-3205d0a29f7e","password":"TestPassword!123","lhsInstallDate":"1650525935885","se
cretCode":"JTp#-v4jN#@v"}
```

This issue can be further confirmed by submitting a report for COVID infection, entering any random OTP:

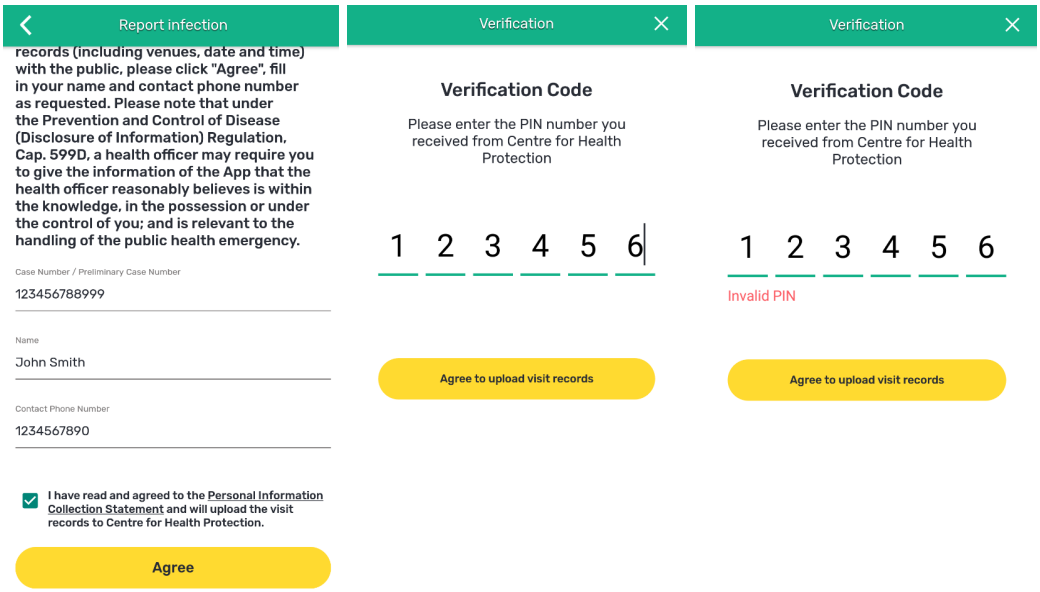

Fig.: Steps to report an infection

Note how no user warnings were shown to the user following the above steps. Now confirm that the OTP was successfully intercepted without any errors. The captured HTTP request contains the personal *One Time Password (OTP)* provided by the Hong Kong *Centre for Health Protection (CHP)*, as well as the case number:

Resulting HTTP Request:

```
POST /app/pin/verify HTTP/1.1
Host: app.regqr.gov.hk
Accept: application/json, text/plain, */*
Content-Type: application/json;charset=utf-8
Content-Length: 113
Accept-Encoding: gzip, deflate
```

```
User-Agent: okhttp/3.12.1
Connection: close

{"verifyCode":"123456","uploadBatchSize":1,"caseNum":"123456788999","uid":"03f94924-43
60-4ced-a3f4-dcc09d013a2a"}
```

Please note that many more HTTP requests can be intercepted without warnings on Android. The above steps are shown to illustrate this vulnerability in a concise manner.

The root cause for this issue appears to be in the following file:

Affected File (decompiled):
hk/gov/ogcio/leavehomesafe/e.java

Affected Code (decompiled):

```
public class e implements HostnameVerifier {
    public e(MainApplication mainApplication) {
    }

    @Override // javax.net.ssl.HostnameVerifier
    public boolean verify(String str, SSLSession sSLSession) {
        Log.d("XANA", "verify: " + str);
        if (str.contains("regqr.gov.hk") || str.contains("leavehomesafe.gov.hk") ||
str.contains("ehc.gov.hk")) {
            return true;
        }
        return HttpsURLConnection.getDefaultHostnameVerifier().verify(str,
sSLSession);
    }
}
```

It is recommended to improve the TLS validation of the Android app to resolve this issue. The *OWASP Pinning CheatSheet*[6] could then be used to secure TLS communications further, so the application only trusts the expected server certificates.

[6] https://cheatsheetseries.owasp.org/cheatsheets/Pinning_Cheat_Sheet.html

LHS-01-002 WP1: Possible Phishing via Task Hijacking on Android *(Medium)*

Testing confirmed that the Android app is currently susceptible to a number of task hijacking attacks. The app is vulnerable to *StrandHogg*[7] and other techniques documented since 2015[8]. A malicious app could leverage this weakness to manipulate the way in which users interact with the app. More specifically, this would be instigated by relocating a malicious attacker-controlled activity in the screen flow of the user, which may be useful to perform Phishing, Denial-of-Service or capturing user-credentials. This issue has been exploited by banking malware trojans in the past[9].

Malicious applications typically exploit task hijacking using one or more of the following techniques:

- ***Task Affinity Manipulation***: The malicious application has two activities M1 and M2 wherein *M2.taskAffinity* = *com.victim.app* and *M2.allowTaskReparenting* = *true*. If the malicious app is opened on M2, once the victim application has initiated, M2 is relocated to the front and the user will interact with the malicious application.
- ***Single Task Mode***: If the victim application has set *launchMode* to *singleTask*, malicious applications can use *M2.taskAffinity* = *com.victim.app* to hijack the victim application task stack.
- ***Task Reparenting***: If the victim application has set *taskReparenting* to *true,* malicious applications can move the victim application task to the malicious application stack.

This issue can be confirmed by reviewing the *AndroidManifest* of the Android application, which fails to set the *android:taskAffinity* attribute at both the application and activity level:

Affected File:
AndroidManifest.xml

Affected Code:

```
<application android:theme="@style/AppTheme" android:label="@string/app_name"
android:icon="@mipmap/ic_launcher"
android:name="hk.gov.ogcio.leavehomesafe.MainApplication" android:allowBackup="false"
android:usesCleartextTraffic="false" android:roundIcon="@mipmap/ic_launcher_round"
```

[7] https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/
[8] https://s2.ist.psu.edu/paper/usenix15-final-ren.pdf
[9] https://arstechnica.com/.../...fully-patched-android-phones-under-active-attack-by-bank-thieves/

```
android:appComponentFactory="androidx.core.app.CoreComponentFactory">
 <activity android:label="@string/app_name"
android:name="hk.gov.ogcio.leavehomesafe.MainActivity" android:exported="false"
android:launchMode="singleTask" android:screenOrientation="portrait"
android:configChanges="keyboard|keyboardHidden|orientation|screenSize|uiMode"
android:windowSoftInputMode="adjustPan" />
 <activity android:label="@string/app_name"
android:name="hk.gov.ogcio.leavehomesafe.SplashActivity">
        <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
```

The issue was further validated at runtime using the *AttackerApp*[10] from the *Task_Hijacking_Strandhogg* github project[11]. Only the following change was made prior to building the app:

File:
app/src/main/AndroidManifest.xml

Contents Before:

```
android:taskAffinity="com.zombie.ssa"
```

Contents After:

```
android:taskAffinity="hk.gov.ogcio.leavehomesafe"
```

It is recommended to implement as many of the following countermeasures as deemed feasible by the development team:

- The task affinity should be set to an empty string. This is best implemented in the Android manifest **at the application level**, which will protect all activities and ensure the fix works even if the launcher activity changes. The application should use a randomly generated task affinity instead of the package name to prevent task hijacking, as malicious apps will not have a predictable task affinity to target.
- The *launchMode* should then be changed to *singleInstance* (instead of *singleTask*). This will ensure continuous mitigation in *StrandHogg 2.0*[12] while improving security strength against older task hijacking techniques[13].
- A custom *onBackPressed*() function could be implemented to override the default behavior.

[10] https://github.com/az0mb13/Task_Hijacking_Strandhogg/tree/main/AttackerApp
[11] https://github.com/az0mb13/Task_Hijacking_Strandhogg
[12] https://www.xda-developers.com/strandhogg-2-0-android-vulnerability-explained.../
[13] http://blog.takemyhand.xyz/2021/02/android-task-hijacking-with.html

- The *FLAG_ACTIVITY_NEW_TASK* should not be set in *activity launch* intents. If deemed required, one should use the aforementioned in combination with the *FLAG_ACTIVITY_CLEAR_TASK* flag[14].

Affected File:
AndroidManifest.xml

Proposed fix:

```
<application android:theme="@style/AppTheme" android:label="@string/app_name"
android:icon="@mipmap/ic_launcher"
android:name="hk.gov.ogcio.leavehomesafe.MainApplication" [...]
android:taskAffinity="">
 <activity android:label="@string/app_name"
android:name="hk.gov.ogcio.leavehomesafe.MainActivity" android:exported="false"
android:launchMode="singleInstance" android:screenOrientation="portrait"
android:configChanges="keyboard|keyboardHidden|orientation|screenSize|uiMode"
android:windowSoftInputMode="adjustPan" />
 <activity android:label="@string/app_name"
android:name="hk.gov.ogcio.leavehomesafe.SplashActivity"
android:launchMode="singleInstance" >
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
```

LHS-01-003 WP1: Leaks via Missing Security Screen on Android & iOS *(Low)*

It was found that the Android and iOS apps fail to render a security screen when they are backgrounded. This allows attackers with physical access to an unlocked device to see data displayed by the apps before they disappeared into the background. A malicious app or an attacker with physical access to the device could leverage these weaknesses to gain access to user-information, such as sensitive or compromising data related to user visit records, *Hong Kong Health Code System* credentials or PII.

To replicate this issue in Android or iOS, simply navigate to some sensitive screen and then send the application to the background. After that, show the open apps and observe how the text which has been input can be read by the user. This text will be readable even after a phone reboot.

[14] https://www.slideshare.net/phdays/android-task-hijacking

Example 1: *Hong Kong Health Code System* HK ID & Credentials leak

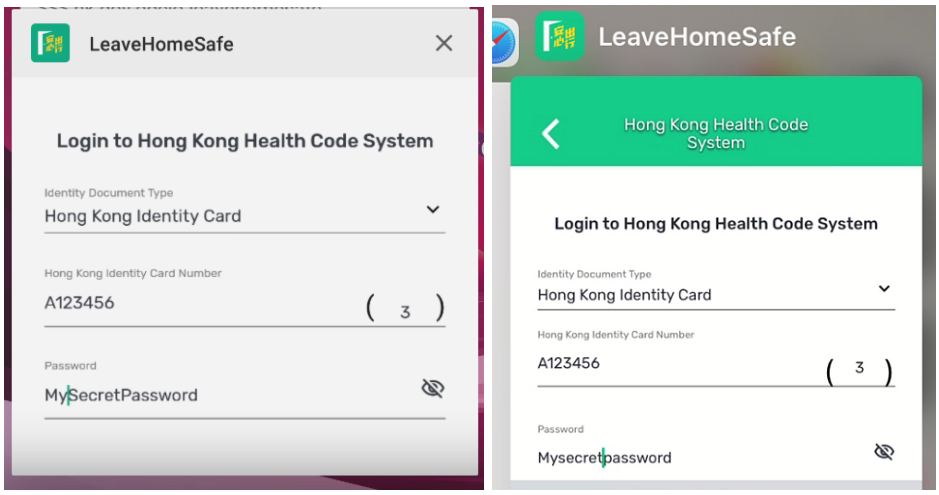

Fig.: Login leak via missing security screen on Android (left) and *iOS (right)*

Example 2: PII and COVID infection leak on Report Infection screen

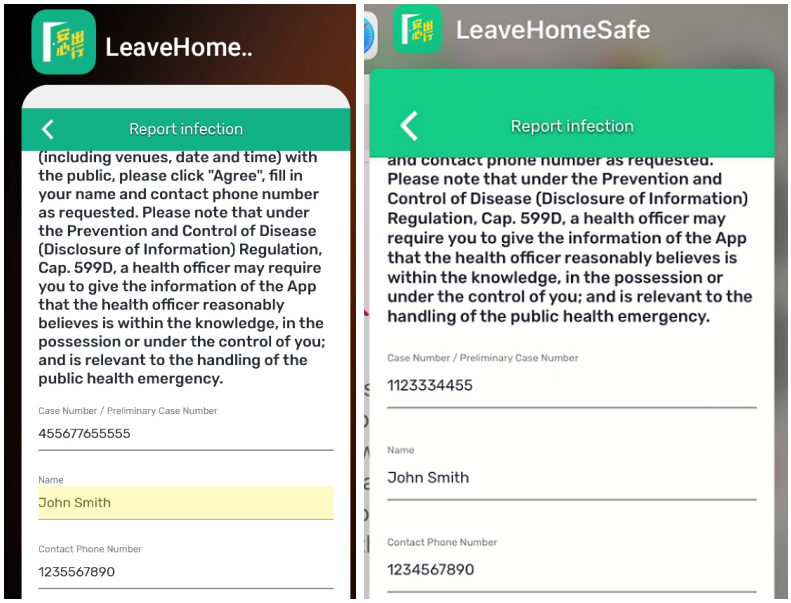

Fig.: COVID injection leak via missing security screen on Android (left) and *iOS (right)*

Example 3: Visit Record Leaks

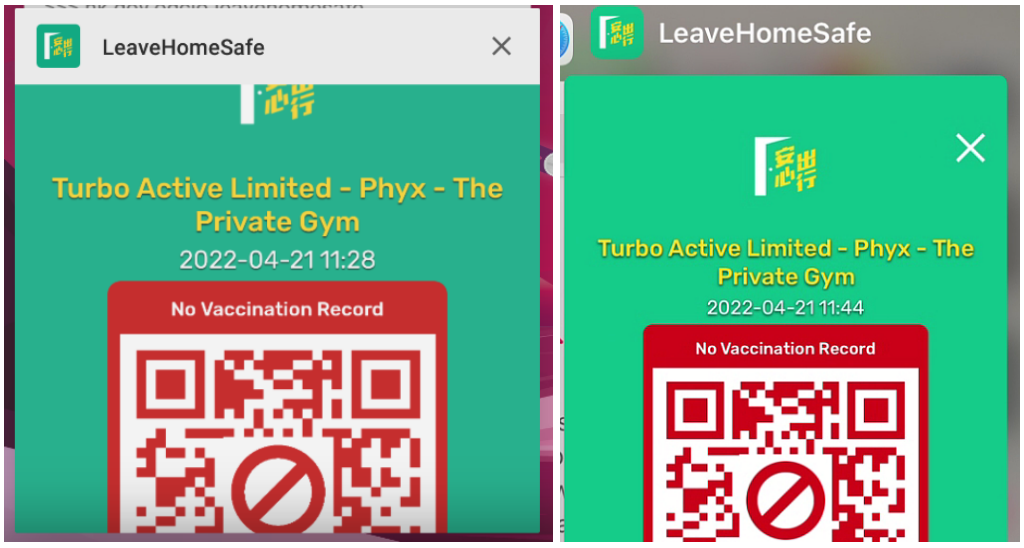

Fig.: Possible ongoing visit leak on Android (left) & iOS (right)

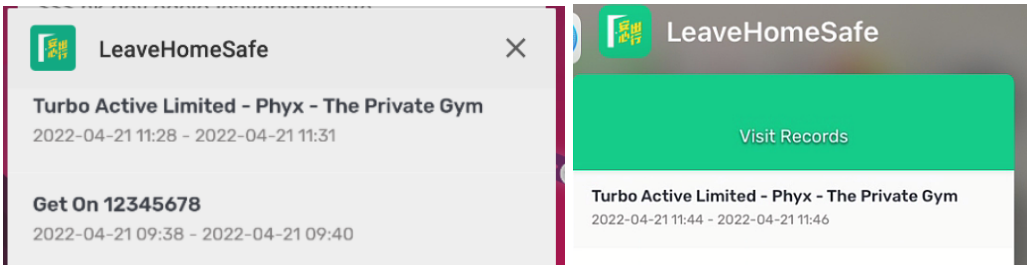

Fig.: Possible visit record leaks on Android (left) and iOS (right)

It is recommended to render a security screen on top when the app is going to be sent to the background:

For iOS apps, the application being sent into the background can be detected in *Swift*[15] and *Objective-C*[16]. After that, a different screen, namely the security screen without user-data, can be shown. A revised approach prevents leakage of sensitive information via iOS screenshots. This is typically accomplished in the *AppDelegate* file, using the *applicationWillResignActive* or *applicationDidEnterBackground* methods. Alternatively,

[15] https://www.hackingwithswift.com/example-code/system/how-to-detect-when-your-app-mo...ackground
[16] https://developer.apple.com/...-applicationwillresignactive?language=objc

the *react-native-privacy-snapshot* plugin[17] or a React Native approach based on monitoring *AppState*[18] transitions into the background state would also work for iOS[19].

For Android apps, it is recommended to implement a security screen by capturing the relevant backgrounding events, typically *onActivityPause*[20] or the *ON_PAUSE* Lifecycle event[21] are used for such purposes. After that, if possible, ensure that all views have the Android *FLAG_SECURE* flag[22] set. This will guarantee that even apps running with root privileges are unable to directly capture information displayed by the app on screen. Alternatively, the *MainActivity.java* file could be amended to always set this flag, regardless of the focus[23]. Unlike iOS, React Native Android apps cannot use the React *AppState* to reliably implement a security screen[2425], however, it is still possible to prevent screenshots and achieve the security screen protection that way using the *expo-screen-capture* package[26].

In addition to the above, some apps implement an app-specific PIN or password to unlock the app. However, solutions like Face or Touch ID might be a more user-friendly choice while providing users with strong security measures. In such cases, the app would lock automatically when backgrounded and require Face or Touch ID to be unlocked.

LHS-01-007 WP1: COVID Status Access via Unsafe SD Card Usage *(High)*

It was found that the *LeaveHomeSafe* Android app stores COVID vaccination & COVID test status images in the SD Card. This occurs when the user attempts to import such QR Codes from safer locations, such as Google Drive. This finding is concerning because the Android SD Card is an inappropriate location for sensitive data. For example, an unskilled thief could extract the SD Card and plug it to a computer to read this data, without having to know the PIN or unlock pattern. Similarly, anything stored in the SD Card can be read or modified by all other apps on the device with SD Card access. This issue was confirmed as follows:

Navigate to the import function to enter a COVID vaccination or test record:

[17] https://www.npmjs.com/package/react-native-privacy-snapshot
[18] https://reactnative.dev/docs/appstate
[19] https://forums.expo.io/t/how-to-blur-the-ios-screenshot-when-app-in-background/43526/4
[20] https://developer.android.com/.../Application.ActivityLifecycleCallbacks#onActivityPaused...
[21] https://developer.android.com/reference/androidx/lifecycle/Lifecycle.Event
[22] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE
[23] https://gist.githubusercontent.com/jonaskuiler/.../raw/.../MainActivity.java
[24] https://medium.com/...creating-a-security-screen-on-ios-and-android-in-react-native-97703092e2de
[25] https://forums.expo.io/t/hide-screen-content-when-switching-apps/33355/3
[26] https://docs.expo.io/versions/latest/sdk/screen-capture/

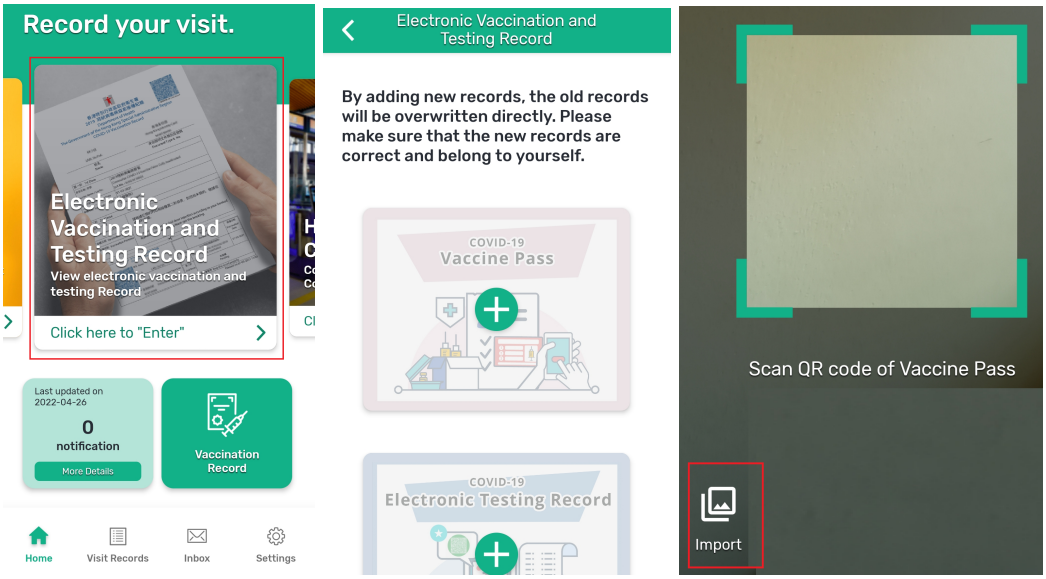

Fig.: Navigation to the Electronic Vaccination/Testing import

Select any image from a safe location that requires authentication, such as Google Drive, then complete the crop process by centering the QR code and clicking on the check icon at the top right of the screen:

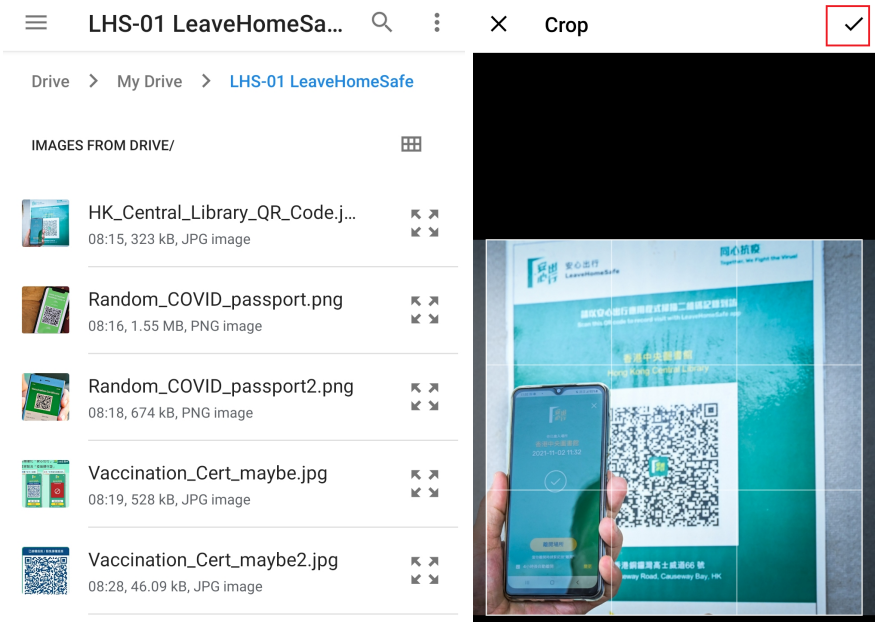

Fig.: Completing the Import process

Optionally, reboot the device, to verify that these images will remain in the SD Card even after the app is closed and the device is restarted. Then, run the following ADB command:

ADB Command:

```
adb shell ls "/mnt/sdcard/Android/data/hk.gov.ogcio.leavehomesafe/files/Pictures/"
```

Output:

```
9e4e788e-1961-4c13-87fe-cced0906be31.jpg
```

Result:
The scanned image remains in the SD Card, this can be trivially downloaded and verified in a computer using the following ADB command:

ADB Command:

```
adb pull
"/mnt/sdcard/Android/data/hk.gov.ogcio.leavehomesafe/files/Pictures/9e4e788e-1961-4c13
-87fe-cced0906be31.jpg"
```

It is recommended to completely avoid the SD Card for storing sensitive data such as COVID vaccination or COVID test status. Such images should instead be stored in the internal storage of the application (i.e. /data/data/…), where Android can enforce permissions. If needed, the application can then grant access to the relevant application (i.e. Android Camera) utilizing a *FileProvider*[27]. Alternatively, at a minimum, the application should consider encryption or regularly delete all SD Card QR Codes as soon as they are used, as well as when the application is opened or closed. If this latter approach is chosen, please note that even shredding[28] is not guaranteed to safely erase files for flash storage[29]. However, it will reduce the forensic recovery chances for an attacker with SD Card access.

[27] https://developer.android.com/training/secure-file-sharing/setup-sharing
[28] https://www.gnu.org/software/coreutils/manual/html_node/shred-invocation.html
[29] https://unix.stackexchange.com/questions/593181/is-shred-bad-for-erasing-ssds

LHS-01-008 WP1: COVID Status Access via Auth Bypass *(High)*

The *LeaveHomeSafe* Android and iOS apps have a feature to enable authentication to access COVID vaccination and test status, this requires entering the PIN or fingerprint for the device. It was found that this feature can be trivially bypassed due to a logic flaw. A malicious attacker, with access to an unlocked phone, could gain access to the user COVID vaccination and COVID test status by simply tapping through screens, hence defeating the intended security feature with minimal effort and skill. In short, this security control provides absolutely no security at the time of writing. This issue was verified using the following steps on the Android and iOS apps:

Navigate to the app settings, enable Authentication and verify that the Fingerprint/PIN appears to be required from that moment to access COVID vaccination or test status:

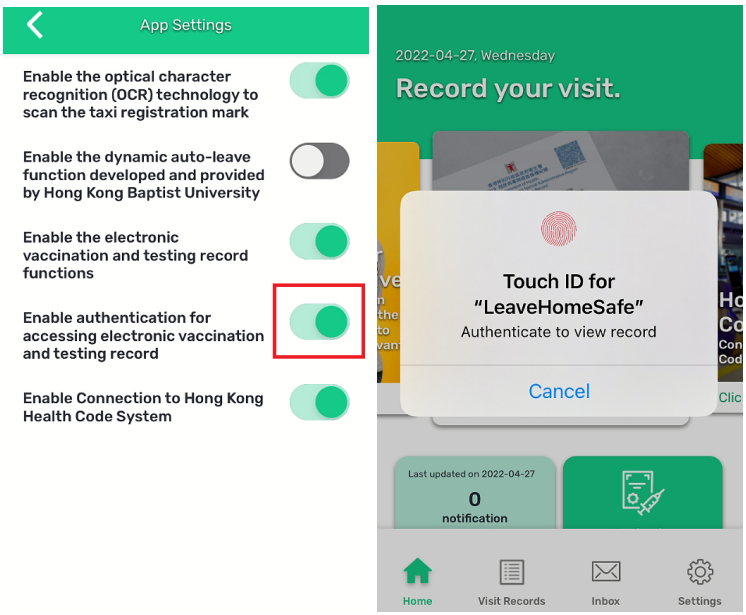

Fig.: Enabling authentication (iOS) requires Touch ID to access COVID status data

The steps to enable authentication are identical for Android:

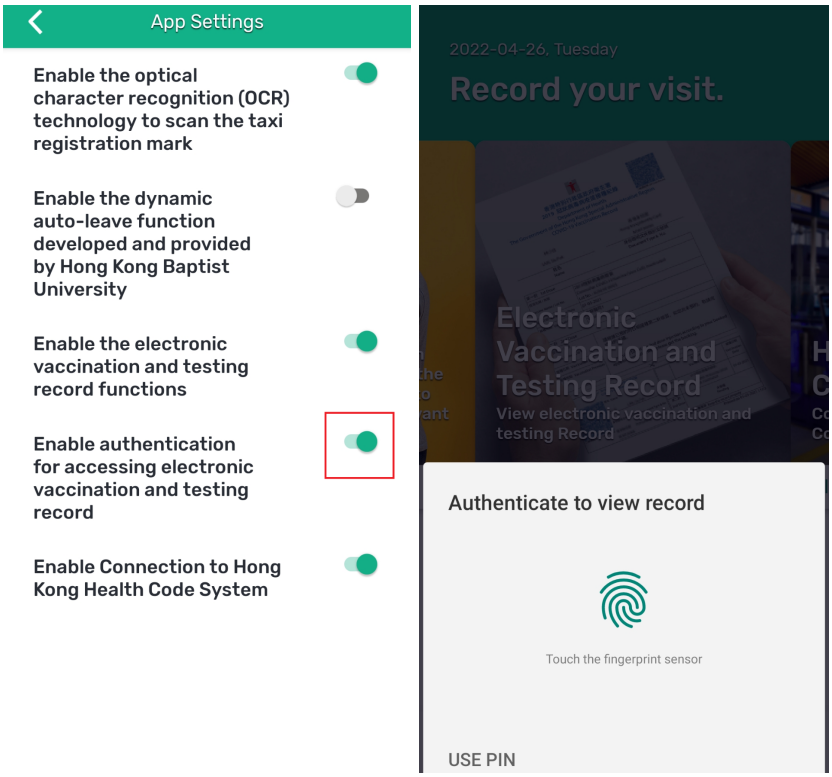

Fig.: Authentication (Android) requires the Fingerprint or PIN for COVID status data

Optionally, restart the device and open the app again to verify the intended restrictions.

On iOS, confirm Touch ID is still required, then disable authentication and confirm access:

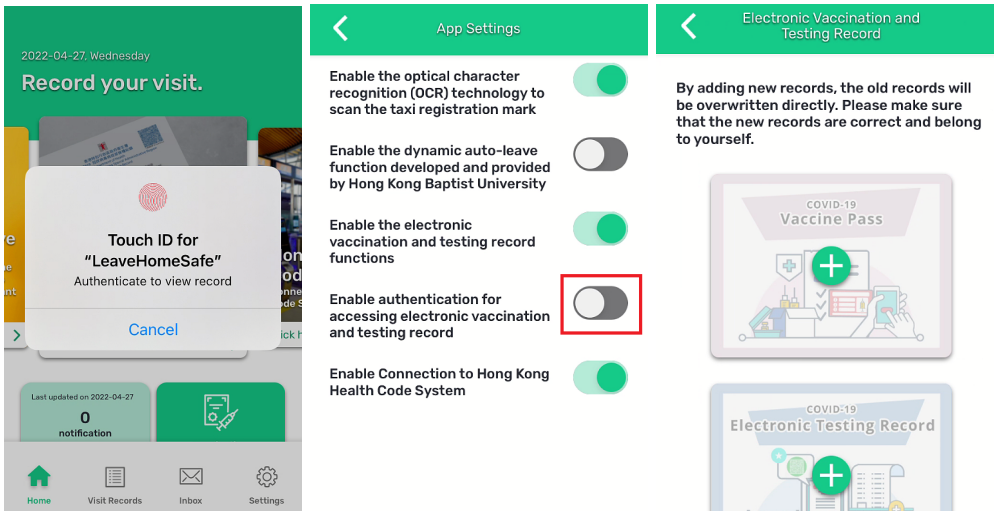

Fig.: Auth bypass in iOS

Following the same steps on Android results in an identical bypass:

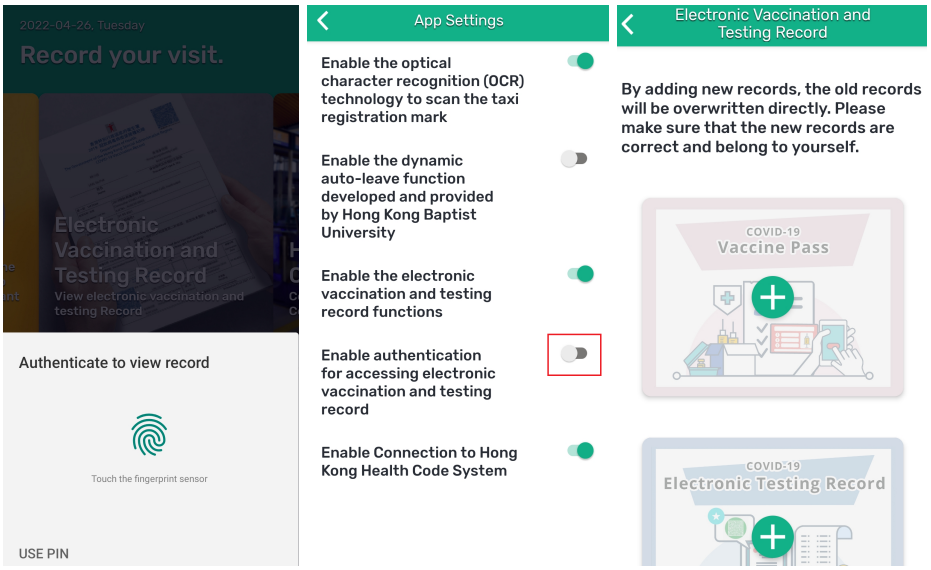

Fig.: Auth bypass in Android

It is recommended to require the Fingerprint or PIN whenever the *"Enable authentication for accessing electronic vaccination and testing record"* setting is enabled or disabled. Furthermore, this feature should ideally protect the entire application, including the user Visit Record, the *Hong Kong Health Code System* screens, etc.

LHS-01-009 WP1: Weaknesses in iOS Keychain usage *(Medium)*

It was found that most information is saved in the iOS keychain with an access level of *AfterFirstUnlockThisDeviceOnly*[30]. Although this prevents leaks via iCloud and iTunes backups, it still keeps the keychain data accessible for the app and *root* processes while the phone is locked. The data could therefore be leaked by physical attackers able to scrape it from memory. Seemingly sensitive fields such as *com.google.iid.checkin*, *hk.gov.ogcio.leavehomesafe, and com.firebase.FIRInstallations.installations* are affected. Slightly more concerning is the use of *AccessibleAfterFirstUnlock*[31] for the encrypted *Hong Kong Health Code System* credentials, which may additionally leak via iCloud or iTunes backups, this affects the *hk.gov.ogcio.leavehomesafe.hcs* field.

A malicious attacker with access to memory could leverage this weakness to gain access to most iOS keychain fields and, thereby, effectively acquire insight or access to citizen information. The level of iOS keychain access identified during the test is

[30] https://developer.apple.com/.../security/ksecattraccessibleafterfirstunlockthisdeviceonly
[31] https://developer.apple.com/documentation/security/ksecattraccessibleafterfirstunlock

summarized in the table below.

Level of Access	Field	Value(s)
AfterFirstUnlock	*hk.gov.ogcio.leavehomesafe.hcs*	*SglUx[...]* *8vmN[...]*
AfterFirstUnlock ThisDeviceOnly	*hk.gov.ogcio.leavehomesafe*	*54095[...]:**
AfterFirstUnlock ThisDeviceOnly	*com.google.iid.checkin*	*52851300[...]|83[...]*
AfterFirstUnlock ThisDeviceOnly	*com.firebase.FIRInstallations.installations*	*1:540[...]:ios:087048[...]__FIRAPP_DEFAULT*

For keychain items that are not required by processes running in the background, it is recommended to use a more restricted level of access. The best options for approaching this are noted below, ordered by the protection level they provide (i.e. ideal option first):

Option 1: *AccessibleWhenPasscodeSetThisDeviceOnly*[32]:

This is the absolute best option, it requires users to have a passcode set in the device and makes keychain items only available while the device is unlocked. Data will not be exported to backups and credentials will not be restored on another device when backups are restored.

Please note this option can be further secured by requiring the user to authenticate via *Face* or *Touch ID* prior to the application being able to access the relevant keychain item[33].

Option 2: *AccessibleWhenUnlockedThisDeviceOnly*[34]:

This is the best option if the data should not be exported to backups. Credentials will not be restored on another device when the backup is restored.

[32] https://developer.apple.com/documentation/security/ksecattraccessiblewhenpasscodesetthisdeviceonly
[33] https://developer.apple.com/.../accessing_keychain_items_with_face_id_or_touch_id
[34] https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlockedthisdeviceonly

Option 3: *AccessibleWhenUnlocked*[35]:

This is the best option if the data should be exported to backups Credentials will be restored on another device when the backup is restored.

Please note that, for keychain items that require to be accessible while the device is locked, the *AccessibleAfterFirstUnlockThisDeviceOnly*[36] Keychain level of access will at least prevent potential leaks via iCloud or iTunes backups.

LHS-01-010 WP1: COVID Status Access via missing Data Protection *(Medium)*

It was found that the iOS app does not currently implement the available *Data Protection* features in iOS. This means that most files are encrypted with the default *NSFileProtectionCompleteUntilFirstUserAuthentication*[37] encryption, which keeps the decryption key in memory while the device is locked. Moreover, this is the least secure form of data protection available on iOS. A malicious attacker with physical access to the device could leverage this weakness to read the decryption key from memory and gain access to local app data files, without needing to unlock the device. Further scrutiny revealed that some of the unprotected files display COVID vaccination & test status, the device Firebase *FCM* token, as well as alternative information.

To replicate this issue, a jailbroken phone was left at rest for a few minutes on the lock screen, then all application files were retrieved for inspection of any potential data leak. A handful of examples revealed by the app files retrieved during device lock can be consulted below:

Example 1: Access to COVID vaccination and test status

In situations when the user imported the COVID vaccination or test status into the app using the “Import” function, those files will be saved without encryption and are fully readable while the device is on the lock screen:

Affected Files:
tmp/react-native-image-crop-picker/BF5AD9CD-5F28-4DE2-97E5-31D197FEF225.jpg
tmp/react-native-image-crop-picker/F2E827A0-8AA9-4890-A2A2-98928FB899CE.jpg

[35] https://developer.apple.com/documentation/security/ksecattraccessiblewhenunlocked
[36] https://developer.apple.com/documentation/security/ksecattraccessibleafterfirstunlockthisdeviceonly
[37] https://developer.apple.com/.../nsfileprotectioncompleteuntilfirstuserauthentication

Example 2: Access to Firebase FCM token

Based on the application implementation, it appears that access to the Firebase device token would allow an attacker to subscribe to *LeaveHomeSafe* notifications for the victim user.

Affected File:
Library/Application Support/hk.gov.ogcio.leavehomesafe/RCTAsyncLocalStorage_V1/manifest.json

Affected Contents:


```
{"ASYNC_SHOW_TNC":"true","ASYNC_AGREE_HCS":"true","ASYNC_BACK_TO_PREVIOUS_VENUE_DATA":
"","ASYNC_ENABLE_EVT_SECURITY":"false","ASYNC_FCM_TOKEN":"fPlJrBcYckRJgqj2rX_S41:APA91
bHLzBo[...]","ASYNC_LAST_DOWNLOAD_TS":"1651119311000","ASYNC_SHOW_NEW_FEATURE":"EVT_OP
T_OUT2","databases\/vaccineMapping_ts.txt":"1644199454005","ASYNC_JOIN_TS":"1650532198
359","databases\/revokeList_ts.txt":"164419945400","ASYNC_UID":"f36af6b8-8cdf-4c19-b25
2-208ddd41983d","ASYNC_SHOW_TUTORIAL":"true","ASYNC_IN_PROGRESS_END_TS":"1651120808501
","ASYNC_VENUE_DB_TS":"1649378060000","ASYNC_LAST_DELETE_TS":"1651120036405","database
s\/vaccinePassMultipleStage_ts.txt":"1644199454006","ASYNC_IN_PROGRESS_START_TS":"1651
120808175","ASYNC_AGREE_EVT":"true","ASYNC_USER_UPDATE_TIME":"1651120808474","ASYNC_HC
_RMB_ME":"true"}
```


The extent of this issue is perhaps best illustrated by the output of the *tar* command, which is able to read most files after the phone has remained passive on the lock screen for a few minutes. This clearly demonstrates that most files are currently unprotected at rest.

Commands:

```
tar cvfz files_locked.tar.gz * > unprotected_files.txt 2> protected_files.txt
wc -l protected_files.txt
wc -l unprotected_files.txt
```

Output:

```
5 protected_files.txt
80 unprotected_files.txt
```

It is recommended to add the *Data Protection* capability at the application level[38]. This will ensure that application data files are protected at rest with the strongest form of

[38] https://developer.apple.com/documentation/.../com_apple_developer_default-data-protection

encryption available on iOS: *NSFileProtectionComplete*[39]. Furthermore, in order to protect cached entries, it is possible to subclass *NSURLCache* with a custom subclass that stores URL responses in a custom SQLite database with file protection set to *NSFileProtectionComplete*[40]. Alternatively, before the request is sent, caching could be disabled with a code snippet similar to the one shown below.

Proposed fix (to be used before a request is sent):

```
configuration.requestCachePolicy = .reloadIgnoringCacheData
```

An alternative mitigatory action could be to clear all cached responses after the response is received.

Proposed fix (for after the response is received):

```
URLCache.shared.removeAllCachedResponses()
```

In addition to the above, *SQL Cipher*[41] could be considered to encrypt SQLite databases at rest. The encryption key should be stored in the iOS keychain while data remains protected. For additional mitigation guidance, please see the blog post titled "*Best practices to avoid security vulnerabilities in your iOS app*"[42].

LHS-01-011 WP1: Possible App Notification Access via iOS Backups *(Low)*

It was found that the iOS Firebase device token is leaked in iOS backups. A malicious attacker with access to the iOS backups of a *LeaveHomeSafe* user could leverage this weakness to receive messages intended for the victim user, which might potentially reveal sensitive information. This issue was found to affect both encrypted and unencrypted iOS backups. Please note that the Android application is not affected by this issue, as it explicitly disables backups. To replicate this issue, simply backup the iOS device with iTunes and review the backup contents:

Affected File:
Library/Application Support/hk.gov.ogcio.leavehomesafe/RCTAsyncLocalStorage_V1/manifest.json

Affected Contents:

```
{"ASYNC_SHOW_TNC":"true","ASYNC_AGREE_HCS":"true","ASYNC_BACK_TO_PREVIOUS_VENUE_DATA":
"","ASYNC_ENABLE_EVT_SECURITY":"false","ASYNC_FCM_TOKEN":"fPlJrBcYckRJgq[...]","ASYNC_
```

[39] https://developer.apple.com/documentation/foundation/nsfileprotectioncomplete
[40] https://stackoverflow.com/questions/27933387/nsurlcache-and-data-protection
[41] https://www.zetetic.net/sqlcipher/ios-tutorial/
[42] http://blogs.quovantis.com/best-practices-to-avoid-security-vulnerabilities-in-your-ios-app/

```
LAST_DOWNLOAD_TS":"1651119311000","ASYNC_SHOW_NEW_FEATURE":"EVT_OPT_OUT2","databases\/
vaccineMapping_ts.txt":"1644199454005","ASYNC_JOIN_TS":"1650532198359","databases\/rev
okeList_ts.txt":"164419945400","ASYNC_UID":"f36af6b8-8cdf-4c19-b252-208ddd41983d","ASY
NC_SHOW_TUTORIAL":"true","ASYNC_IN_PROGRESS_END_TS":"1651120808501","ASYNC_VENUE_DB_TS
":"1649378060000","ASYNC_LAST_DELETE_TS":"1651120036405","databases\/vaccinePassMultip
leStage_ts.txt":"1644199454006","ASYNC_IN_PROGRESS_START_TS":"1651120808175","ASYNC_AG
REE_EVT":"true","ASYNC_USER_UPDATE_TIME":"1651120808474","ASYNC_HC_RMB_ME":"true"}
```


It is recommended to exclude this file from iOS backups[43] or encrypt this information at rest to avoid potential attacks. Please note that, while iOS backups cannot be disabled entirely, a number of features exist to limit what iTunes and iCloud will back up[44].

Miscellaneous Issues

This section covers notable findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are weaknesses that did not provide an easy way to be exploited. To conclude, while a vulnerability is present, an exploit might not always be possible.

LHS-01-004 WP1: Missing Jailbreak/Root Detection on Android & iOS *(Info)*

It was found that the Android and iOS apps do not currently implement any form of Root or Jailbreak detection features at the time of writing. Hence, the applications fail to alert users about the security implications of running the app in such an environment[45]. This issue can be confirmed by installing the application on a jailbroken/rooted device and validating the complete lack of application warnings.

It is recommended to implement a comprehensive Jailbreak and root detection solution to address this problem. Please note that, since the user has root access and the application does not, the application is always at a disadvantage. **Mechanisms like these should always be considered bypassable** when enough dedication and skill characterize the attacker.

Some freely available libraries for iOS are *IOSSecuritySuite*[46] and *DTTJailbreakDetection*[47], although custom checks are also possible in Swift

[43] https://developer.apple.com/documentation/foundation/optimizing_your_app_s_data_.../
[44] https://docs.microsoft.com/en-us/answers/questions/731588/xamarin-ios-disable-back-up....html
[45] https://www.bankinfosecurity.com/jailbreaking-ios-devices-risks-to-users-enterprises-a-8515
[46] https://cocoapods.org/pods/IOSSecuritySuite
[47] https://github.com/thii/DTTJailbreakDetection

applications[48]. Such solutions should be considered bypassable but sufficient to warn users about the dangers of running the application on a jailbroken device. For best results, it is recommended to test some commercial and open source[4950] solutions against well-known *Cydia tweaks* like *LibertyLite*[51], *Shadow*[52], *tsProtector 8+*[53] or *A-Bypass*[54]. Based on this, *LeaveHomeSafe* could determine the most solid approach.

The freely available *rootbeer* library[55] for Android could be considered for the purpose of alerting users on rooted devices, while bypassable, this would be sufficient for alerting users of the dangers of running the app on rooted devices.

Please note that React Native applications may easily implement the aforementioned recommendations using third party solutions such as *jail-monkey*[5657] or *react-native-jailbreak*[58], both of which support Android and iOS.

LHS-01-005 WP1: Support of Insecure v1 Signature on Android *(Info)*

It was found that the Android build currently in production is signed with an insecure *v1 APK signature*. Using the *v1* signature makes the app prone to the known Janus[59] vulnerability on devices running Android < 7. The problem lets attackers smuggle malicious code into the APK without breaking the signature. At the time of writing, the app supports a minimum SDK of 16 (Android 4.1), which also uses the v1 signature, hence being vulnerable to this attack. Furthermore, Android 4.1 devices no longer receive updates and are vulnerable to many security issues, it can be assumed that any installed malicious app may trivially gain root privileges on those devices using public exploits[606162].

The existence of this flaw means that attackers could trick users into installing a malicious attacker-controlled APK which matches the v1 APK signature of the legitimate

[48] https://sabatsachin.medium.com/detect-jailbreak-device-in-swift-5-ios-programatically-da467028242d
[49] https://github.com/thii/DTTJailbreakDetection
[50] https://github.com/securing/IOSSecuritySuite
[51] http://ryleyangus.com/repo/
[52] https://ios.jjolano.me/
[53] http://apt.thebigboss.org/repofiles/cydia/
[54] https://repo.rpgfarm.com/
[55] https://github.com/scottyab/rootbeer
[56] https://github.com/GantMan/jail-monkey
[57] https://infinitbility.com/how-to-detect-device-rooted-or-jailbroken-in-react-native/
[58] https://www.npmjs.com/package/react-native-jailbreak
[59] https://www.guardsquare.com/en/blog/new-android-vulnerability-allows-atta….affecting-their-signatures
[60] https://www.exploit-db.com/exploits/35711
[61] https://github.com/davidqphan/DirtyCow
[62] https://en.wikipedia.org/wiki/Dirty_COW

Android application. As a result, a transparent update would be possible without warnings appearing in Android, effectively taking over the existing application and all of its data.

It is recommended to increase the minimum supported SDK level to at least 24 (Android 7) to ensure that this known vulnerability cannot be exploited on devices running older Android versions. In addition, future production builds should only be signed with v2 and greater APK signatures.

LHS-01-006 WP1: Android Binary Hardening Recommendations *(Info)*

It was found that a number of binaries embedded into the Android application are currently not leveraging the available compiler flags to mitigate potential memory corruption vulnerabilities. This unnecessarily puts the application more at risk for such issues.

Issue 1: Missing usage of -D_FORTIFY_SOURCE=2 on most binaries

Missing this flag means common libc functions are missing buffer overflow checks, so the application is more prone to memory corruption vulnerabilities. Please note that most binaries are affected, the following is a reduced list of examples for the sake of brevity.

Example binaries:
lib/arm64-v8a/libnative-imagetranscoder.so
lib/arm64-v8a/libglog_init.so
lib/arm64-v8a/libhermes-executor-release.so
lib/arm64-v8a/libucrop.so
lib/arm64-v8a/libjsijniprofiler.so
[...]

Issue 2: Missing RELRO on some binaries

A number of binaries leave the GOT section writable. Without the RELRO flag, buffer overflows on a global variable can overwrite GOT entries.

Affected Binaries:
lib/arm64-v8a/libnative-imagetranscoder.so
lib/arm64-v8a/libucrop.so
lib/arm64-v8a/libconceal.so
lib/arm64-v8a/libnative-filters.so

lib/arm64-v8a/libimagepipeline.so

It is recommended to compile all binaries using the *-D_FORTIFY_SOURCE=2* argument so that common insecure glibc functions like *memcpy*, etc. are automatically protected with buffer overflow checks.

Regarding RELRO, two mitigation options are available:

Option 1: Using -z,relro,-z,now

This will enable full RELRO and is the best protection available

Option 2: Using only -z,relro

This will enable partial RELRO

LHS-01-012 WP1: Usage of Insecure Crypto functions and PRNG *(Medium)*

It was found that the *LeaveHomeSafe* Android app makes use of a number of cryptographic functions with known security weaknesses, either directly or through inherited libraries. Specifically, AES/CBC may be vulnerable to padding oracle attacks[63], MD5 and SHA1 are obsolete hashing algorithms with known weaknesses[64][65]. Furthermore, the code audit revealed that multiple values are generated with the weak random number generator *java.util.Random*. This does not provide secure random numbers in terms of a Cryptographically-Secure Pseudorandom Number Generator (CSPRNG)[66]. Usage of these suboptimal choices makes the security of the Android app more brittle and should be avoided.

Issue 1: Usage of insecure hashing functions (MD5, SHA1)

This appears to be an obfuscated version of *SecureHashUtil* from *com.facebook.common.util*, with potentially low security implications in practice[67].

Affected File (decompiled):
g/e/e/k/c.java

[63] https://jiang-zhenghong.github.io/blogs/PaddingOracle.html
[64] https://en.wikipedia.org/wiki/MD5#Overview_of_security_issues
[65] https://en.wikipedia.org/wiki/SHA-1#Attacks
[66] https://en.wikipedia.org/wiki/Cryptographically-secure_pseudorandom_number_generator
[67] https://github.com/facebook/fresco/issues/965

Affected Code (decompiled):

```
public static String a(byte[] bArr) {
        try {
            MessageDigest instance = MessageDigest.getInstance("SHA-1");
            instance.update(bArr, 0, bArr.length);
            return Base64.encodeToString(instance.digest(), 11);
        } catch (NoSuchAlgorithmException e2) {
            throw new RuntimeException(e2);
        }
    }
```

The following example corresponds to the *RNFetchBlobUtils.getMD5* function from *com.RNFetchBlob*[68]. This appears to be used for generating temporary file names[69].

Affected File (decompiled):
com/RNFetchBlob/h.java

Affected Code (decompiled):

```
public static String b(String str) {
 try {
        try {
                MessageDigest instance = MessageDigest.getInstance("MD5");
                instance.update(str.getBytes());
                [...]
```

It is recommended to replace *MD5* and *SHA1* with adequate replacements without cryptographic weaknesses[70]. It has to be noted that certain secrets should be stored in a deliberately slow manner to avoid brute force attacks, these require a different set of hashing algorithms for secure storage as explained in the *OWASP Password Storage Cheat Sheet*[71].

Issue 2: Usage of AES/CBC

The below example appears to be from the *FingerprintCipher.java* file of the *react-native-touch-id* third-party module[72].

Affected File (decompiled):

[68] https://github.com/joltup/rn-fetch-blob/.../RNFetchBlob/RNFetchBlobUtils.java#L22-L44
[69] https://github.com/joltup/rn-fetch-blob/issues/708#issuecomment-755337764
[70] https://en.wikipedia.org/wiki/Secure_Hash_Algorithms
[71] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
[72] https://github.com/naoufal/react-native-touch-id/.../rnfingerprint/FingerprintCipher.java

com/rnfingerprint/c.java

Affected Code (decompiled):

```
public Cipher b() {
 Cipher cipher = this.a;
 if (cipher != null) {
        return cipher;
 }
 try {
        KeyStore a = a();
        this.a = Cipher.getInstance("AES/CBC/PKCS7Padding");
```

It is recommended to change the current value from "*AES/CBC/PKCS7Padding*" to "*AES/CTR/NoPadding*". The reason for this is that alternative implementations using *Cipher Mode Chaining* mode (CBC) may be vulnerable to padding oracle attacks[73]. Counter mode (CTR) is considered a better choice because it is free from weaknesses.

Issue 3: Usage of insecure PRNG

LeaveHomeSafe makes use of *java.util.Random* for cryptographic purposes on multiple locations:

Affected File (decompiled):
k/a/a/b/g.java

Affected Code (decompiled):

```
import java.util.Random;
[...]
protected byte[] d(int i2) throws a {
 if (i2 > 0) {
        byte[] bArr = new byte[i2];
        Random random = new Random();
        for (int i3 = 0; i3 < i2; i3++) {
               bArr[i3] = b((byte) random.nextInt(256));
        }
        return bArr;
 }
 throw new a("size is either 0 or less than 0, cannot generate header for standard
encryptor");
}
```

[73] https://jiang-zhenghong.github.io/blogs/PaddingOracle.html

Affected File (decompiled):
g/f/a/c/i/b/b.java

Affected Code (decompiled):

```
import java.util.Random;
[...]
public final class b {
    private static final Random a = new Random();
    public static String a() {
        byte[] bArr = new byte[16];
        a.nextBytes(bArr);
        return Base64.encodeToString(bArr, 11);
    }
}
```

Please note that the above code appears to be inherited from *zip4j*, which fixed this issue in 2020[74]. This proves that *LeaveHomeSafe* uses older versions of Java libraries, potentially putting users at risk through inherited publicly-known vulnerabilities.

It is recommended to replace all occurrences of *java.util.Random* with a cryptographically-secure alternative such as *java.security.SecureRandom*[75]. The PRNG will then be sufficiently safeguarded against cryptographic attacks, whilst ensuring all functionality remains backwards compatible.

Issue 4: Usage of a weak initialization vector

The following code suggests that the *LeaveHomeSafe* app is using a weak initialization vector (IV) containing only zeros, which weakens the overall security of the cryptographic implementation:

Affected File:
hk/gov/ogcio/leavehomesafe/BuildConfig.java

Affected Code:

```
public static final String HKCT_IV = "0000000000000000";
```

[74] https://github.com/srikanth-lingala/zip4j/commit/613279a7843045d180b6e5b0d64bc682178b3f5a
[75] https://developer.android.com/reference/java/security/SecureRandom

Its usage was also identified inside the Hermes compiled binary:

Affected File:
resources/assets/index.android.bundle

Affected Code (decompiled):

```
Function<x>8338(3 params, 16 registers, 0 symbols):
    GetEnvironment              Reg8:0, UInt8:0
    LoadFromEnvironment     Reg8:1, Reg8:0, UInt8:12
    GetById                     Reg8:1, Reg8:1, UInt8:1, UInt16:11287
    ; Oper[3]: String(11287) 'enc'

[...]
    GetById                     Reg8:1, Reg8:1, UInt8:5, UInt16:13145
    ; Oper[3]: String(13145) 'HKCT_IV'

    Call2                       Reg8:1, Reg8:2, Reg8:3, Reg8:1
    LoadFromEnvironment     Reg8:2, Reg8:0, UInt8:12
    GetById                     Reg8:5, Reg8:2, UInt8:6, UInt16:14094
    ; Oper[3]: String(14094) 'AES'

    GetById                     Reg8:4, Reg8:5, UInt8:7, UInt16:10811
    ; Oper[3]: String(10811) 'encrypt'

    NewObject                   Reg8:3
    PutNewOwnByIdShort          Reg8:3, Reg8:1, UInt8:127
    ; Oper[2]: String(127) 'iv'
```

It is recommended to generate IVs randomly for better security. For additional mitigation guidance, please see the *CWE-1204: Generation of Weak Initialization Vector (IV)* page[76].

[76] https://cwe.mitre.org/data/definitions/1204.html

Privacy Analysis Findings

This section covers the privacy-related analysis results that attempt to answer 13 questions for *WP2 - Privacy tests against LeaveHomeSafe Android & iOS apps*. For this portion of the engagement, the *7ASecurity* team utilizes the following classification to specify the level of certainty regarding the documented findings. Given that this research occurred on the basis of reverse-engineering, and needed to be executed in a stealthy manner, it is necessary to classify the findings to address the level of confidence that can be assumed for each discovery:

- Proven: Source code and runtime activity clearly confirm the finding as fact
- Evident: Source code strongly suggests a privacy concern, but this could not be proven at runtime
- Assumed: Indications of a potential privacy concern was found but a broader context remains unknown.
- Unclear: Initial suspicion was not confirmed. No privacy concern can be assumed.

LHS-01-Q02 WP2: Files & Information gathered by LeaveHomeSafe (Assumed)

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q2: What files/information are gathered by the Android & iOS apps?

7ASecurity found evidence of the existence of source code that collects user information in *LeaveHomeSafe* dependencies. However, during dynamic analysis such data was not conclusively found to be collected by the app or sent to the *LeaveHomeSafe* servers at runtime. Hence, this appears to be an artifact, rather than a malicious privacy violation. Furthermore, this type of code is common in the dependencies of commercial applications.

An example of this is the *react-native-device-info* module[77], used by both the Android & iOS apps, which contains a large number of functions to collect device information, such as MAC address[78], phone number[79], among other data. The following source code snippet shows the retrieval of carrier information from the decompiled app:

[77] https://github.com/react-native-device-info/react-native-device-info
[78] https://github.com/react-native-device-info/react-native-device-info#getmacaddress
[79] https://github.com/react-native-device-info/react-native-device-info#getphonenumber

Affected Packages:
com.learnium.RNDeviceInfo.RNDeviceModule
com.learnium.RNDeviceInfo.DeviceType
com.learnium.RNDeviceInfo.Device.p049d.TypeResolver
com.learnium.RNDeviceInfo.p049d.DeviceIdResolver

Example Affected Code:

```
@ReactMethod(isBlockingSynchronousMethod = true)
    public String getCarrierSync() {
        TelephonyManager telephonyManager = (TelephonyManager)
getReactApplicationContext().getSystemService("phone");
        if (telephonyManager != null) {
            return telephonyManager.getNetworkOperatorName();
        }
        System.err.println("Unable to get network operator name. TelephonyManager was
null");
        return "unknown";
    }
```

In summary the identified third-party components may obtain the following information:

1. General phone information: Device ID, system name, model, brand, device type
2. Connection information: WiFi SSID, carrier, IP address, MAC address, etc.
3. User-information: UIDs, cookies, session, event, page and track IDs
4. Available location providers
5. Running processes and services

Please note that, while the capability is there for this information to be captured, its usage could not be confirmed at runtime.

A related behavior noticed while instrumenting the applications was a number of read and write operations on the following files, however, upon further inspection they were found to be empty:

Affected Files:
/data/user/0/hk.gov.ogcio.leavehomesafe/shared_prefs/react-native-device-info.xml
/data/user/0/hk.gov.ogcio.leavehomesafe/shared_prefs/react-native-device-info.xml.bak

LHS-01-Q03 WP2: Where & How LeaveHomeSafe transmits Data *(Unclear)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q3: Where and how are the files/information gathered transmitted?

The *LeaveHomeSafe* application contains a number of third-party dependencies with capabilities to collect information about the device (LHS-01-Q02). However, *7ASecurity* did not find evidence of sensitive data exfiltration at runtime. Please note that the test team was not able to access all functionality in the mobile applications, this was due to lack of data such as valid *Hong Kong Health Code System* credentials, which was a major limitation during this assignment.

This being said, the decompiled code declares the following endpoints for communication:

Affected File (decompiled):
hk/gov/ogcio/leavehomesafe/BuildConfig.java

Affected Code (decompiled):

```
public static final String HCS_BASE_URL = "https://apply.ehc.gov.hk/lhsapi";
public static final String HCS_REG_URL = "https://register.ehc.gov.hk";
public static final String HCS_WEB_URL = "https://www.ehc.gov.hk";
public static final String HKEN_BASE_URL = "https://www.leavehomesafe.gov.hk";
public static final String HKEN_DEFAULT_BASE_URL = "https://app.regqr.gov.hk/app";
public static final String HKEN_JSON_BASE_URL = "https://regqr.gov.hk/app";
public static final String HKEN_WEB_URL = "https://www.leavehomesafe.gov.hk";
```

Based on the analysis of the Hermes/React binary, the main communication endpoints of the application are *https://app.regqr.gov.hk/app* and *https://regqr.gov.hk/app*. Those are used for communication between the application internals and the backend servers. Additionally, they appear within the first web calls during the start of the application:

Command:

```
curl "https://www.leavehomesafe.gov.hk/site.json"
```

Output:

```
{"api":"https://app.regqr.gov.hk/app", "json":"https://www.regqr.gov.hk/app" }
```

Please note that traffic to the above endpoints was noticed while exercising the Android and iOS applications, however, no PII or device information was observed in transit.

LHS-01-Q04 WP2:LeaveHomeSafe fails to protect PII at rest & in transit *(*Proven*)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q4: Is sensitive PII such as COVID Vaccination/Infection status information insecurely stored or easily retrievable from the apps?

The security review of the *LeaveHomeSafe* application comprehensively proves that COVID Vaccination/Infection status data is not sufficiently protected:

1. On at least some supported Android devices, *LeaveHomeSafe* fails to validate TLS certificates correctly, which can result in access to COVID data in transit, as well as *Hong Kong Health Code System credentials*, among other possibilities (LHS-01-001).
2. COVID status (and other information) can be leaked via Android and iOS screenshots, due to a missing security screen (LHS-01-003).
3. COVID status may be available to unauthenticated physical attackers able to extract the SD Card from an Android device, as well as malicious applications with SD Card access (LHS-01-007).
4. The COVID PIN/Fingerprint protection feature can be trivially bypassed on both Android & iOS simply tapping through screens (LHS-01-008).
5. COVID status may additionally be leaked to attackers with physical access to a locked iOS device due to failure to leverage the appropriate iOS file system protection features (LHS-01-010).

LHS-01-Q05 WP2: Visit Record Weaknesses in Transit & at Rest *(*Proven*)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q5: Do the apps protect the user visit record appropriately at rest and in transit?

The *LeaveHomeSafe* apps were found to contain the following weaknesses related to this question during the security review:

1. On at least some supported Android devices, *LeaveHomeSafe* fails to validate TLS certificates correctly, which can result in access to the Visit Record data in

transit, as well as *Hong Kong Health Code System credentials*, among other possibilities (LHS-01-001).
2. The Visit Record (and other information) can be leaked via Android and iOS screenshots, due to a missing security screen (LHS-01-003).

This being said, the Android & iOS apps were found to use the *HKEN.db* SQLite database to store user PII, the Visit Record and other information encrypted at rest:

Affected Files:
/data/user/0/hk.gov.ogcio.leavehomesafe/databases/HKEN.db **[Android]**
/var/mobile/Containers/Data/Application/[...]/Library/LocalDatabase/HKEN.db **[iOS]**

Example UserInfoData from the HealthCodeUse Table (encrypted):
```
1|a30110f9-3ed9-4c01-83b6-7c2d14cf359e|U2FsdGVkX182Hl08n3l1Ngji+Tmo/ITLy5DJY4Tg
q8jEZ9mELeoxSe8/Pu0XO/igR9p5f3B1TbSz5W2w1PsL/pVP2fncIsMLihK6cxDyX6g=
```

Example UserVisitData from the LocalHistory Table (encrypted):
```
1|2751123|YgzNLVh+7PNBbTy9awcoCbcLn5P4FUAjKRtAX1n7dElsic/o2PoDyfl9t4n5n3stNQ0vC
NpPxzh4GLWmYYfDI4v246QIUDwyEP9GuoyoknS8w/3Px272Qz5M233hpaePOR6aOl6SBJma2RUivgB4
ZkIJLEXpDFBzJWzqXzOnagtnnjPKVZ959SmSl5q7SMpkBeV0VodQukIBjbyiB6J9/36HVzE1mPAXzW9
wnvhtNeQdmPfbQWMZ5w0mNmj2a/MwM4Oe96yHflO2dbQ1sk1MxMcFK/G931O5mio+Fg5z3uJ93W5rtl
MKqtZykwMdcCGQ5SGA5SQ1YHtoE3v+Zi+HqA==|0|0
2|2751124|wbqsr8KUp2g71zVhxSPBhJRdbky112jHGZx5kHF2sKaS2fBhEDYiE0duzyUnTGc25XHId
7KaUjxPPGKobpM0X5oY+aCi7FUTq8WFby0W2aaN/RJ4X3p0cvC7mmAekiSV|0|0
```

Please note that due to the lack of *Hong Kong Health Code System* credentials, *7ASecurity* was not able to access all functionality related to this area. However, given credentials can be intercepted (LHS-01-001), it has been assumed that the visit record will either be retrievable in transit or accessible from the *Hong Kong Health Code System*.

LHS-01-Q06 WP2: Presence of Face Recognition Code *(Evident)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q6: Do the apps implement any sort of user tracking function via location or other means?

A concerning finding during the privacy analysis was the presence of multiple face detection artifacts in the decompiled Android application, although usage of these could not be confirmed at runtime. Please note that no equivalent face recognition traces could

be found on the iOS app. For example, analysis of the Android app revealed face recognition code from the following libraries:

- *Google Face detector*[80]
- *React Native Face Detector*[81]

Evidence of code from the aforementioned libraries was found in multiple files:

Google Face Detector Example

The affected files appear to be located within the decompiled directory *com/google/android/gms/vision/face/*. The following examples prove presence of code for the following face recognition factory classes in the latest Android version (3.2.3):

- *INativeFaceDetectorCreator*
- *NativeFaceDetectorV2Creator*
- *ChimeraNativeFaceDetectorCreator*

Affected File Example (decompiled):
com/google/android/gms/vision/face/internal/client/k.java

Affected Code Example (decompiled):

```
package com.google.android.gms.vision.face.internal.client;

import android.os.IBinder;
import android.os.IInterface;
import k.f.a.c.i.o.a;

public abstract class k extends a implements h {
    public static h k0(IBinder iBinder) {
        if (iBinder == null) {
            return null;
        }
        IInterface queryLocalInterface =
iBinder.queryLocalInterface("com.google.android.gms.vision.face.internal.client.INativ
eFaceDetectorCreator");
        if (queryLocalInterface instanceof h) {
            return (h) queryLocalInterface;
        }
        return new j(iBinder);
    }
}
```

[80] https://developers.google.com/android/reference/com/google/android/gms/vision/face/FaceDetector
[81] https://github.com/huutaiit/react-native-camera/.../RNFaceDetector.java

Affected File (decompiled):
com/google/android/gms/vision/face/internal/client/b.java

Affected Code (decompiled):

```
@Override // k.f.a.c.i.o.v6
protected final /* synthetic */ g a(DynamiteModule dynamiteModule, Context context)
throws RemoteException, DynamiteModule.a {
 h hVar;
 if (w6.a(context, "com.google.android.gms.vision.dynamite.face")) {
        hVar =
k.k0(dynamiteModule.c("com.google.android.gms.vision.face.NativeFaceDetectorV2Creator"
));
 } else {
        hVar =
k.k0(dynamiteModule.c("com.google.android.gms.vision.face.ChimeraNativeFaceDetectorCre
ator"));
 }
 if (hVar == null) {
        return null;
 }
 k.f.a.c.f.a n0 = k.f.a.c.f.b.n0(context);
 f fVar = this.f1068i;
 q.j(fVar);
 return hVar.C(n0, fVar);
}
```

Please note that the above classes are used in some obfuscated Java files, outside of the google vision directory. However, upon further inspection it was discovered this appears to be an obfuscated version of the following library file:
com.google.android.gms:play-services-vision@@20.1.3

Affected File (decompiled, obfuscated path):
k/f/a/c/n/e/*c.java*

Affected Code (decompiled):

```
import com.google.android.gms.vision.face.internal.client.b;
import com.google.android.gms.vision.face.internal.client.f;
[...]

public a d(float f) {
 if (f < 0.0f || f > 1.0f) {
        StringBuilder sb = new StringBuilder(47);
        sb.append("Invalid proportional face size: ");
        sb.append(f);
```

```
        throw new IllegalArgumentException(sb.toString());
 }
 this.f3212g = f;
 return this;
}
```

Example 2: React Native Face Detector

Similarly, traces of the React Native *FaceDetectorModule* were also found during the privacy audit in the decompiled *org/reactnative/facedetector* and *org/reactnative/camera* directories. The following example illustrates some of this evidence:

Affected File (decompiled):
org/reactnative/camera/c.java

Affected Code (decompiled):

```
import org.reactnative.facedetector.FaceDetectorModule;
[...]
public class c implements ReactPackage {
    @Override // com.facebook.react.ReactPackage
    public List<NativeModule> createNativeModules(ReactApplicationContext
reactApplicationContext) {
        return Arrays.asList(new RCTCameraModule(reactApplicationContext), new
CameraModule(reactApplicationContext), new
FaceDetectorModule(reactApplicationContext));
    }
```

Please note that, unlike the Google Face Detector example above, no usage of the React Native Face Detector could be found outside of the *org/reactnative/camera* directory, within the decompiled java files.

All this being said, while testing the application at runtime only the rear camera was found to be used, hence this potential privacy issue could not be validated at runtime:

ADB Logcat Examples:

```
I/CameraService(  258): CameraService::connect call (PID 2835
"hk.gov.ogcio.leavehomesafe", camera ID 0) for HAL version default and Camera API
version 1
I/CameraService::connect call (PID 2835 "hk.gov.ogcio.leavehomesafe", camera ID 0) for
HAL version default and Camera API version 1
I/CameraService(  258): CameraService::connect call (PID 2835
"hk.gov.ogcio.leavehomesafe", camera ID 0) for HAL version default and Camera API
```

```
version 1
```

Furthermore, extensive analysis of the *LeaveHomeSafe* Hermes binary revealed no signs of face detection usage. While an *onFaceDetected* event handler is present, this code appears to be inherited from an underlying library and no indicators of usage could be found either at rest during reversing or at runtime while instrumenting the application.

Additionally, the Android & iOS apps were not found to request any location tracking in their permissions or at runtime.

For Android, in the *AndroidManifest.xml* file only the *android.permission.GET_TASKS* permission seems suspicious. This allows *LeaveHomeSafe* to retrieve information about currently and recently running tasks, which appears unnecessary for what the app is supposed to do. However, no Android location tracking permissions[82] are requested.

Similarly, for iOS the Info.plist only requests *NSCameraUsage*, *NSFaceIDUsage*, *NSMotionUsage* and *NSPhotoLibraryUsage*. Hence, the apps are simply unable to track the user location directly.

This said, as already mentioned in LHS-01-Q02, functionality present in the underlying dependencies of the apps, such as the the *react-native-device-info* module[83] used by both the Android & iOS apps, contain some tracking-related code. For example, the following decompiled code snippet reveals how this module may obtain location providers availables in the phone:

Affected File (decompiled):
com/learnium/RNDeviceInfo/RNDeviceModule.java

Affected Code (decompiled):

```
public WritableMap getAvailableLocationProvidersSync() {
   LocationManager locationManager = (LocationManager)
getReactApplicationContext().getSystemService("location");
   WritableMap createMap = Arguments.createMap();
   try {
       for (String str : locationManager.getProviders(false)) {
           createMap.putBoolean(str, locationManager.isProviderEnabled(str));
       }
   } catch (Exception unused) {
```

[82] https://developer.android.com/training/location/permissions
[83] https://github.com/react-native-device-info/react-native-device-info

```
        System.err.println("Unable to get location providers. LocationManager was
null");
    }
    return createMap;
```

While reversing the compiled React part of the Android application, the *RNDeviceModule*, particularly *RNDeviceInfo*, contains code to handle the usage of location features, which can be seen by the appearance of the following strings inside the decompiled code:

- *saveLocationHistoryToDB*
- *getLocationHistoryList*
- *getLocationHistoryToUpload*
- *markLocationHistoryAsUploaded*
- *deleteAllLocationHistory*
- *deleteOverRetentionDays*
- *deleteLocationHistoryAfterRetenDays*
- *getMyLocationHistoryList*

Similarly, the above strings are also present in the minified *main.jsbundle* JavaScript file of the analyzed iOS application:

Affected File:
Payload/LeaveHomeSafe.app/main.jsbundle

Affected Code (Minified JavaScript):

```
__d(function(g,r,i,a,m,e,d){var
t=r(d[0]),n=r(d[1]);Object.defineProperty(e,"__esModule",{value:!0}),e.saveLocationHis
toryToDB=function(t){return
c.default.async(function(n){for(;;)switch(n.prev=n.next){case 0:return
n.abrupt("return",new Promise(function(n,o){var
c=null;h.initLocationHistoryDB().then(function(s){c=s,Promise.all([L(s,t)]).then(funct
ion(u){if(console.log('saveLocationHistoryToDB values:',u),u.length>0)[...]
```

All this being said, *7ASecurity* did not find any evidence of this data being collected at runtime.

LHS-01-Q07 WP2: LeaveHomeSafe weakens TLS Communications *(Proven)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q7: Do the apps intentionally weaken cryptographic procedures to ensure third-party decryption?

It is unclear whether this weakening is intentional or simply a bug, however, evidence of defeating TLS validation with a faulty hostname verifier was found and proven at runtime against the Android app during this assignment (LHS-01-001). The bug allows malicious third parties to intercept traffic between the app and backend servers, when the host name contains the strings *regqr.gov.hk*, *leavehomesafe.gov.hk* or *ehc.gov.hk*.

Affected File (decompiled):
hk/gov/ogcio/leavehomesafe/e.java

Affected Code (decompiled):

```
public class e implements HostnameVerifier {
    public e(MainApplication mainApplication) {
    }

    @Override // javax.net.ssl.HostnameVerifier
    public boolean verify(String str, SSLSession sSLSession) {
        Log.d("XANA", "verify: " + str);
        if (str.contains("regqr.gov.hk") || str.contains("leavehomesafe.gov.hk") ||
str.contains("ehc.gov.hk")) {
            return true;
        }
        return HttpsURLConnection.getDefaultHostnameVerifier().verify(str,
sSLSession);
    }
}
```

Additionally, other cryptographic weaknesses were identified and reported during the security review in LHS-01-012. Nevertheless, it is not as evident that those weaknesses are intentional to facilitate third party decryption.

LHS-01-Q08 WP2: LeaveHomeSafe insecure SD Card Usage *(*Proven*)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q8: Is data dumped in the SD Card from where it could be retrieved later without even entering the PIN to unlock the device?

While the *LeaveHomeSafe* Android and iOS apps generally encrypt user PII and data at rest, a scenario was found during the security audit demonstrating that COVID vaccination status can be available to unauthenticated physical attackers able to extract the SD Card from an Android device, as well as malicious applications with SD Card access, this is explained in more detail in LHS-01-007.

LHS-01-Q09 WP2: Potential LeaveHomeSafe RCE Issues *(*Unclear*)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q9: Do the apps contain vulnerabilities or shell commands that could lead to RCE in any way?

7ASecurity could not find any evidence of potential vulnerabilities, shell commands or any other weakness that could lead to RCE in the Android or iOS applications during this engagement.

LHS-01-Q10 WP2: Potential LeaveHomeSafe Backdoors *(*Unclear*)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q10: Do the apps have any kind of backdoor?

The *7ASecurity* team was unable to find any evidence of process or command execution calls commonly used by backdoors and malware in the Android and iOS applications.

LHS-01-Q11 WP2: Potential LeaveHomeSafe root Access *(Unclear)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q11: Do the apps attempt to gain root access through public Android/iOS vulnerabilities or in other ways?

At the time of writing, no evidence could be identified to suggest that the mobile applications are trying to leverage or exploit platform-specific vulnerabilities. Additionally, the applications are not making any reference to the root status of the device or checking for any privilege at the device level.

LHS-01-Q12 WP2: Potential LeaveHomeSafe Obfuscation *(Proven)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q12: Do the apps use obfuscation techniques to hide code and if yes for which files and directories?

The Android application is using code obfuscation methods for package, class and function names, where they are renamed to single character strings. These findings indicate that identifier renaming is enabled, while string encryption is not used and Java reflection was not present.

However, there are specific instances that have names untampered, including the *hk.gov.ogcio.leavehomesafe.hkbu* package. In that case, inline local strings and method parameters are not altered.

Example Classes (with code obfuscation):
package i.e.a
package f.a.a.a.a

Affected Code (decompiled, obfuscated):

```
package p123g.p124a.p125a;

/* renamed from: g.a.a.b */
/* loaded from: classes.dex */
public class C2309b extends Exception {
    private int errorCode;
```

```
    public C2309b(String str, int i) {
        super(str);
        this.errorCode = i;
    }

    public C2309b(String str, int i, Throwable th) {
        super(str, th);
        this.errorCode = i;
    }

    /* renamed from: a */
    public int m10173a() {
        return this.errorCode;
    }
}
```

Decompilation error examples:

```
WARN  - Code restructure failed: missing block: B:116:0x0234, code lost:
    r6 = true;
 in method: g.f.a.c.i.e.p2.p(T, byte[], int, int, int, int, int, int, long, int, long,
g.f.a.c.i.e.w):int, file: class
es.dex
WARN  - Code restructure failed: missing block: B:117:0x0236, code lost:
    r6 = false;
 in method: g.f.a.c.i.e.p2.p(T, byte[], int, int, int, int, int, int, long, int, long,
g.f.a.c.i.e.w):int, file: class
es.dex
WARN  - Code restructure failed: missing block: B:118:0x0237, code lost:
    r11.m5828h(r6);
 in method: g.f.a.c.i.e.p2.p(T, byte[], int, int, int, int, int, int, long, int, long,
g.f.a.c.i.e.w):int, file: class
es.dex
```

Similarly, on iOS the JavaScript code of the React Native application is minified but not encrypted:

Affected File:
Payload/LeaveHomeSafe.app/main.jsbundle

Affected Code:

```
var
__BUNDLE_START_TIME__=this.nativePerformanceNow?nativePerformanceNow():Date.now(),__DE
V__=false,process=this.process||{};process.env=process.env||{};process.env.NODE_ENV=pr
ocess.env.NODE_ENV||"production";
```

```
!(function(r){"use strict";r.__r=o,r.__d=function(r,i,n){if(null!=e[i])return;var
o={dependencyMap:n,factory:r,hasError:!1,importedAll:t,importedDefault:t,isInitialized
:!1,publicModule:{exports:{}}};e[i]=o},r.__c=n,r.__registerSegment=function(r,e){s[r]=
e};var e=n(),t={},i={}.hasOwnProperty;function n(){return e=Object.create(null)}[...]
```

On Android, while not directly considered as a code obfuscation technique, the Hermes JavaScript engine is used to run the compiled React Native application. The binary contains more than 21k functions and over 1M lines of hardly readable disassembled Hermes Bytecode, with automatically removed identifier names. Many third-party React libraries are embedded inside, making the analysis considerably harder because of the inherited complexity. These statements can be confirmed as follows:

Affected File:
resources/assets/index.android.bundle

Command (disassemble binary):

```
hbctool disasm "resources/assets/index.android.bundle" "/tmp/hermes"
```

Output:

```
[*] Disassemble 'resources/assets/index.android.bundle' to '/tmp/hermes' path
[*] Hermes Bytecode [ Source Hash: 9e5ed8d62eae2a3bf69181e4de71af46863fe442, HBC
Version: 74 ]
[*] Done
```

Commands (retrieve instruction metrics):

```
cd "/tmp/hermes"; ls -sh instruction.hasm; wc -l instruction.hasm
```

Output (size and count of instructions):

```
39M instruction.hasm
1143657 instruction.hasm
```

Commands (count functions):

```
cat instruction.hasm | grep EndFunction | wc -l
```

Output (number of functions):

```
21352
```

Command (show start of file):

```
head instruction.hasm
```

Output (decompiled function example):

```
Function<global>0(1 params, 19 registers, 0 symbols):
    DeclareGlobalVar           UInt32:18173
    ; Oper[0]: String(18173) '__BUNDLE_START_TIME__'

    DeclareGlobalVar           UInt32:18161
    ; Oper[0]: String(18161) '__DEV__'

    DeclareGlobalVar           UInt32:105
    ; Oper[0]: String(105) 'process'
```

LHS-01-Q13 WP2: Identification of Companies behind LeaveHomeSafe (Evident)

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q13: Which external companies help build and maintain these apps?

While checking the downloaded APK file, it was found that the digital signer of it is the *Hong Kong SAR Government (HKSARG) / Office of the Government Chief Information Officer (OGCIO)*. This was confirmed as follows:

Command:

```
jarsigner -verify -certs -verbose LeaveHomeSafe_3.1.0.apk | grep X.509 | sort | uniq
```

Output:

```
X.509, CN=LeaveHomeSafe (APK Download), OU=OGCIO, O=HKSARG, L=HK, ST=HK, C=CN
```

Similarly, the iOS page for *LeaveHomeSafe* reveals the same developer:

URL:
https://apps.apple.com/app/leavehomesafe/id1536377801

Contents:
App Privacy
The developer, ***Office of the Government Chief Information Officer, Hong Kong SAR Government***

Seller
Office of the Government CIO of Hong Kong *Special Administrative Region*

The above can also be found on the latest iOS app at the time of writing (3.2.3):

Affected File:
Payload/LeaveHomeSafe.app/main.jsbundle

Affected Contents:
This mobile application and related system [...] is developed by the Office of the Government Chief Information Officer of the Government of the Hong Kong Special Administrative Region

The same developer account seems to be responsible for a number of HK government related applications on Android[84] and iOS[85]**:** *iAM Smart, LeaveHomeSafe, Tell me@1823, StayHomeSafe, GovHK Apps, GovHK Notifications, EventHK*.

Nevertheless, at the Google Play page for the *LeaveHomeSafe* application, the developer contact information can be seen:

URL:
https://play.google.com/store/apps/details?id=hk.gov.ogcio.leavehomesafe

Output:
```
Developer
info@cherrypicks.com
```

This email address is related to a company called *Cherrypicks*, which defines itself as follows[86]:
"Established in 2000, Cherrypicks is a home-grown technology startup headquartered in Hong Kong and ***now a subsidiary of NetDragon Websoft Holdings Limited (HKSE: 777)****. The company is a regional Mobile Technology and Mobile eCommerce leader specializing in smart city, augmented reality, artificial intelligence, eWallet and location intelligence."*

Even though Cherrypicks is headquartered in Hong Kong, a quick search for *NetDragon Websoft Holdings Ltd*, reveals that the parent company is in China:

URL:
http://www.netdragon.com/about/overview.shtml

[84] https://play.google.com/store/apps/developer?id=GovHK,+OGCIO,+HKSARG&hl=en
[85] https://apps.apple.com/app/leavehomesafe/id1536377801
[86] https://www.cherrypicks.com/about-us/

*"NetDragon Websoft Holdings Limited (Stock Code: 00777.HK), established in 1999, is a global leading creator of internet communities. Its headquarters are **located in Fuzhou, Fujian, China**."*

At the same time, there is no mention of that same company inside the decompiled Android or iOS files.

LHS-01-Q14 WP2: LeaveHomeSafe vs. Study the Great Nation Relation *(Unclear)*

This ticket summarizes the *7ASecurity* attempts to answer the following question during the audit:

Q14: What is the relationship between the LeaveHomeSafe apps and the "Study the Great Nation" app from mainland China if any?

7ASecurity participated in the *"Study the Great Nation"* app analysis together with Cure53 in 2019[87], and delivered talks at multiple security conferences about it afterwards[88][89]. During this assignment no relationship could be found between these apps. In particular, all major characteristics and fingerprints were cross-checked without yielding any clear match, among other attempts, the audit team failed to spot any connection between the apps while checking for:

- Reporting URLs
- Domains
- Characteristic DES based encryption
- Checking for sudo
- Execution of arbitrary commands
- Base64 obfuscated strings
- Package existence checks
- As well as other matching attempts

In short, *LeaveHomeSafe* and *"Study the Great Nation"* were not found to share any similarities.

[87] https://7asecurity.com/reports/analysis-report_sgn.pdf
[88] https://www.youtube.com/watch?v=kuJJ1Jjwn50
[89] https://www.youtube.com/watch?v=chBky3M70KE

Conclusion

This exercise involved both a privacy audit and a security audit of the *LeaveHomeSafe* Android and iOS apps. The privacy audit could not conclusively prove malicious intent or unauthorized tracking of Hong Kong citizens. However, the security audit demonstrated that these applications have not been professionally audited by any competent security firm before, and that significant flaws exist in the current software security development lifecycle.

The *7ASecurity* team had significant limitations during this assignment. This made certain areas of the application simply untestable at runtime, hence it was not possible for the test team to validate the security or privacy promises of the application in certain functional areas. The most significant shortcomings during this assignment were:

1. Lack of *Hong Kong Health Code System* credentials
2. Lack of valid Hong Kong COVID Vaccination QR codes
3. Lack of valid Hong Kong COVID Test QR codes

Regarding the security audit, despite the poor results, a number of positive impressions deserve a mention:

- The Android & iOS apps generally do not leak sensitive information in the filesystem (where data is encrypted) or in logs. It was particularly interesting that *Hong Kong Health Code System* credentials are not leaked in HTTP caching artifacts on Android or iOS, notwithstanding other not-security-relevant data being present in such locations.
- The Android app explicitly disables backups and clear-text HTTP traffic in the Android manifest. Which improves security by eliminating backup leak attack vectors as well as clear-text MitM attacks.
- The iOS app does not implement insecure custom URL schemes or ATS exceptions to weaken TLS protections. Hence avoiding well-known URL hijacking attacks and potential MiTM attacks.
- The hardcoded Google API keys on the Android and iOS apps were found to be correctly restricted to prevent abuse.
- The Android and iOS apps generally protect application secrets well, leveraging the appropriate hardware-backed security enclave for the platform. Namely, the Android KeyStore and the iOS KeyChain. Furthermore, all user information and the visit record were found to be encrypted at rest.
- The Firebase device registration implementation appears to be a good balance to provide *COVID-19* contact tracing capabilities without compromising user

privacy[90]. The application appears to use this functionality to subscribe users to notifications.

The *LeaveHomeSafe* mobile applications were found to be affected by a number of common misconfigurations. Their security posture will improve significantly with a focus on the following areas:

- **Protection of Network Communications:** A concerning weakness identified during this exercise had to do with a faulty hostname verifier which allows interception of TLS traffic between the Android app and the backend servers without any warnings, allowing the capture of *Hong Kong Health Code System* credentials, among other possibilities (LHS-01-001). This type of vulnerability suggests a complete lack of internal security code reviews or third party penetration testing of the mobile applications before each release. Please note this finding was verified on all Android versions between 3.2.0-3.2.3, more versions are likely affected.
- **Authentication Implementation**: The Android and iOS apps were found to implement an authentication feature to protect the COVID vaccination and test status data, however, this can be currently bypassed on both platforms by simply tapping through screens (LHS-01-008). This issue currently provides users with a false sense of security. Furthermore, biometric protection should be considered for the entire app instead of the COVID status alone. For example, the *Hong Kong Health Code System* or the Visit Record are currently unprotected for attackers with access to an unlocked device.
- **Protection of Data at Rest**: The most concerning finding in this regard was the leakage of COVID data in the Android SD Card (LHS-01-007), this issue should be resolved as soon as possible. Generally, the mobile apps correctly make use of the *Android KeyStore* and *iOS Keychain* for storing sensitive information. However, the *iOS* app could improve its *iOS Keychain* and Backup implementation to avoid leaks in backups and against attackers with physical access (LHS-01-009, LHS-01-011), the iOS app should then additionally protect its files at rest through the *iOS Data Protection* features (LHS-01-010).
- **Mitigation of Task Hijacking Attacks**: The Android app should mitigate well-known Task Hijacking attacks (LHS-01-002).
- **Avoidance of Screenshot Leaks**: The *Android* and *iOS* apps would both benefit from implementing a security screen to avoid leaks through screenshots and app backgrounding (LHS-01-003). This is a common security feature in targeted mobile apps such as banking applications.
- **General Hardening**: Other less important hardening recommendations include

[90] https://firebase.google.com/docs/cloud-messaging

implementing a root/jailbreak detection mechanism to alert users about security risks prior to using the application (LHS-01-004), a number of settings that could be improved to better protect users on older supported devices (LHS-01-005, LHS-01-006), and usage of dependencies with known vulnerabilities that implement insecure cryptographic mechanisms (LHS-01-012)

Regarding the privacy audit, the concerns expressed by the media are reasonable, especially as the application continues to be mandated in a number of Hong Kong locations. For background, *LeaveHomeSafe* is mandated in all government venues, hospitals, markets, shopping malls, supermarkets and places of worship, and the number of places where the app is mandated is increasing[91].

While no clear privacy violation could be conclusively proven during the audit at runtime, a number of application artifacts, likely inherited from underlying dependencies or simply security vulnerabilities introduced by mistake, were found during this exercise and could be summarized as follows:

- **Usage of Obfuscation**: Usage of obfuscation techniques was proven during this assignment (LHS-01-Q12). While this is common in commercial applications as well, it only decreases citizen trust in the government mandating this application. The ideal solution, to help solve this trust issue, would be to make the entire *LeaveHomeSafe* application open source and remove obfuscation completely. This way the government can prove that it has nothing to hide, as the code is fully open to third-party scrutiny.
- **Usage of Libraries with Traces of User Tracking**: Given the natural reluctance of the population to use mandated software, together with the use of obfuscation (LHS-01-Q12), traces of Face Recognition functionality identified in the Android application (LHS-01-Q06), only makes this trust problem worse. Please note that these privacy concerns appear to be artifacts inherited from underlying libraries and could not be proven to be used by the application at runtime. It is recommended to completely remove all device tracking and face recognition libraries from the codebase to help resolve the obvious question of *"Why is this in the application in the first place?"*.
- **Data Gathering**: Data gathered by the application did not appear to be sensitive during this assignment, however a number of artifacts inherited from underlying libraries complicate the aforementioned trust problem (LHS-01-Q02). Removing all dependencies that contain any code that gathers device data will substantially improve the way in which the application is perceived.
- **Protection of Data in Transit and at Rest**: The *LeaveHomeSafe* Android and iOS applications contain a number of security vulnerabilities that appear to be

[91] https://zh.wikipedia.org/wiki/%E5%AE%89%E5...

introduced due to lack of security training for the developer team, as well as a complete lack of regular penetration testing of the mobile applications prior to releasing them to the public. In short, *LeaveHomeSafe* fails to properly protect user data at rest and in transit (LHS-01-Q04), a number of weaknesses exist to expose the Visit Record both at rest and in transit (LHS-01-Q05), at least some Android users are exposed to MitM attacks whereby their *Hong Kong Health Code System* credentials could be intercepted (LHS-01-Q07), and COVID information can be accessed from the SD card in at least some scenarios (LHS-01-Q08).

- **Data Sending:** No clear evidence could be identified where the mobile applications send sensitive data from users to the backend servers (LHS-01-Q03). However, a major limitation for the audit team in this regard was the complete lack of valid *Hong Kong Health Code System* credentials.
- **Potential Malicious Behavior**: This is perhaps the most positive privacy analysis aspect of the engagement, as no backdoors (LHS-01-Q10), RCE exploits (LHS-01-Q09) or root privilege escalation (LHS-01-Q11) code could be identified during the test window.
- **Company Behind *LeaveHomeSafe***: The company behind *LeaveHomeSafe* appears to be successfully identified as *CherryPicks* (LHS-01-Q13), a Hong Kong based subsidiary of *NetDragon Websoft Holdings Limited*, which is based in mainland China.
- **Similarities with Study the Great Nation**: *7ASecurity* was unable to find any similarities between the "Study the Great Nation" app analyzed in 2019[92] and *LeaveHomeSafe*. Thus, the apps appear to be completely unrelated.

It is advised to address all issues identified in this report, including informational and low severity tickets where possible. This will not just strengthen the security posture of the platform significantly, but also reduce the number of tickets in future audits.

Once all issues in this report are addressed and verified, a more thorough review, including a full code audit, is highly recommended to ensure adequate security coverage of the platform. This provides auditors with an edge over possible malicious adversaries that do not have significant time or budget constraints. Please note that future audits should ideally allow for a greater budget so that test teams are able to deep dive into more complex attack scenarios.

It is advised to test the platform regularly, at least once a year or when substantial changes are going to be deployed, to make sure new features do not introduce undesired security vulnerabilities. This proven strategy will reduce the number of security

[92] https://7asecurity.com/reports/analysis-report_sgn.pdf

issues consistently and make the platform highly resilient against online attacks overtime.

7ASecurity would like to thank the *Hong Kong Democracy Council* (HKDC) for their project coordination, support and assistance, both before and during this assignment. Last but not least, appreciation must be extended to the *Open Technology Fund (OTF)* for sponsoring this project.