Tracing the Tracer: Analysis of a Mobile Contact Tracing Application

Anthony Wallace
Tracing the Tracer: Analysis of a Mobile Contact Tracing Application

GIAC (GCIH) Gold Certification

Author: Anthony Wallace, anthony.wallace11@gmail.com
Advisor: Lenny Zeltser

Accepted: 9 December 2020

Abstract

The pandemic has led to the rapid development of applications designed to take advantage of our hyper-connected world. The Ehteraz application was developed, deployed, and mandated in the nation of Qatar. Government regulation required citizens to register with the app to enter businesses such as malls and grocery stores which forced rapid adoption among the populace. Many citizens are concerned about the range of permissions the app requires to function. Unpacking the application and finding a method of dissecting network traffic was complicated by measures developers took to prevent miscreant-in-the-middle attacks and analysis. Sharing the journey of decrypting the traffic in this application may prove useful to future engineers reversing and bypassing protections to perform analysis on mobile app traffic. Initial analysis has confirmed the application sends only location and Bluetooth data to centralized servers owned by the Ministry of Interior of the State of Qatar.
1. Introduction

The COVID-19 outbreak is the greatest health crisis in recent history and has forced many to consider how public gatherings (or gatherings of any kind) should happen. It has brought an old concept to the forefront of global conversation—contact tracing. While this has been a major tool for health organizations to help with containing and eliminating disease, 21st century technology vastly increases the volume of data collected and speed of gathering data. In the past, these activities consisted of groups of people manually working to gather data using lists of people. Those groups would call individuals who had been reported as exposed and would manually enumerate the potential circle of infection.

The ubiquity of mobile phones provides a perfect platform to build a system for tracking an infection to a granularity never experienced before. Mobile devices provide all the components necessary to track movement of potentially infected individuals as well as people with whom they have interacted. These components include:

1. Proximity information: A Bluetooth interface allows an application to detect (and identify) surrounding devices.
2. Location information: An in-built GPS provides precise coordinates of a user and history of movement.
3. Internet connectivity: An internet connection allows this information to be collected and transmitted to a centralized database for processing.

All of these factors provide opportunity to rapidly build and deploy a system to defend populations that are too unwieldy for “manual” contact tracing programs. On a national scale, governments and corporations have the ability to even use data already collected to contribute to a contract tracing program that benefits mobile devices users. However, mobile devices have no inherent concept of morality. The same tools used to help eradicate a disease can be used malevolently. Every bit of information listed above can be misused, and applications which are not properly contained by strict permissions management can collect large amounts of unnecessary data without the users’ knowledge.

Anthony Wallace, anthony.wallace11@gmail.com
1.1. Brief History of (ab)use

The major push for contact tracing applications has caused the biggest phone and software developers to consider deploying specific APIs in their operating systems. For example, an iPhone user can open their phone settings and select “Exposure Notifications” to turn on a framework made available to developers in iOS 13.5. It can be used to “inform people of potential exposure to COVID-19” (Apple, 2020). In fact, there are numerous news stories of Apple and Google, the two largest mobile phone operating system companies, teaming up to create applications for multiple states in the U.S. as well as other countries.

The problem with tracing applications is the privacy risk associated with the very nature of these systems. The applications are designed to track people and their associations. One can argue this is nothing new, and companies and governments have been doing this for a while now. However, there are major implications when other organizations with little experience in application development are attempting to tackle this problem. One particular organization concerned with this inept development and privacy overreach is Amnesty International.

The organization’s Security Lab led an investigation into a number of attempts by Middle Eastern, North African and European countries to deploy an application for contact tracing. Among the countries investigated were Bahrain, Kuwait and Qatar. Two of the apps studied in Bahrain and Kuwait “follow an invasive centralized approach, posing a great threat to privacy. These systems capture location data through GPS and upload this to a central database, tracking the movements of users in real-time” (Bahrain, Kuwait and Norway Contact Tracing Apps a Danger for Privacy, 2020).

One application mentioned in a separate report by the Security Lab was the “Ehteraz” application mandated by the government of Qatar in May 2020. The application had exposed Qatari national IDs, names and confinement locations for users of the application (Qatar: Contact tracing app security flaw exposed sensitive personal details of more than one million, 2020). The developers have since repaired this vulnerability in subsequent releases, but this incident highlights a perfect example of reasons a user might be concerned about their data.

Anthony Wallace, anthony.wallace11@gmail.com
Additionally, how can a user of this application be sure that personal, non-pertinent information is not sent from their devices to a central repository? The Ehteraz developers are clear that the company collects location information and surrounding device information. Reading the permissions required by the application in the Google Play Store is disconcerting to say the least.

![Ehteraz Permissions](image)

It is understandably difficult for users to trust these sweeping permissions to any government or organization, especially when the application is overtly designed to track users. Before delving into network traffic analysis, a review of the history of the development group and static analysis of the package helps establish an expectation for functionality and basic architecture.

Anthony Wallace, anthony.wallace11@gmail.com
2. Application Overview

The word Ehteraz means “caution” in Arabic. The Ministry of Interior (MOI) in the State of Qatar was searching for a means to contain or track infections across the country in early 2020. Little information is publicly available about the developers behind the application. The MOI is listed as the author in the Play store, and every Qatari news source in English regarding the app only refers to the MOI for comment on development. To find more information, the APK itself was needed. Using a stock Android device, the app was downloaded from Google Play, and then transferred to an Apple Mac for further analysis via an ADB bridge.

Unpacking the APK using Connor Tumbleson’s “apktool” (https://ibotpeaches.github.io/Apktool/) was trivial. This highly effective tool takes the dex files used by dalvik, Android’s Java VM, and translates them into samli code, defined as human-readable commands and syntax.

Anthony Wallace, anthony.wallace11@gmail.com
The human-readable code can be inspected, and the code can be manipulated and re-compiled into an APK for installation on an Android device. The repacking and re-compiling functions are especially useful when attempting to reverse engineer security features in the code designed to prevent analysis. Bypassing the certificate pinning functions in the application, which prevent some critical network analysis, was possible using this unpacking tool. It allows a user to modify the smali code and repackage the code into dex executables.

One of the most important files in this package is the “AndroidManifest.xml” which contains the permissions and other metadata about the structure of the app including additional resources and libraries for various functionality. The invasive permissions required by Ehteraz are a key concern for those who have reported on this app. There are location permissions (e.g. ACCESS_FINE_LOCATION) which can be expected from an app designed to openly track your location, but reading and writing to external storage or reading the phone state might be troublesome.

Anthony Wallace, anthony.wallace11@gmail.com
The permissions given in the manifest allow the app to manipulate data elsewhere on the phone (“external storage”) or read “the current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device” (Manifest.permission, 2020). In the first run of Ehteraz, a user is prompted to allow access to potentially unnecessary data.

If the user selects “Deny” on any prompt, the app refuses to run, leaving the user without the ability to be “labeled green” or given an uninfected status. The compulsory nature of the application for citizens and expatriates in Qatar presents a problem for security-conscious users.

2.1. Unidentified Development Contractor

An essential part of understanding any program includes looking at the authors or development group. These details provide vital context for an application. They give an idea about the experience of the team, and what to expect in terms of quality. The Play Store indicates the MOI of Qatar itself as the sole developer, and all the contact information seems to go to the ministry. The website listed is “https://moi.gov.qa/”, and

Anthony Wallace, anthony.wallace11@gmail.com
the contact email address is “ideveloper@moi.gov.qa”. No other public references exist for a contractor or other entity that could have built the app. From this vantage point, it would seem the ministry has hired developers directly to run this initiative, but unpacking the application indicates a group outside of the MOI has built the majority (if not all) of the codebase. The folder containing the main application code for Ehteraz is called “orbis”.

Searching for this term turned up very little information. Adding the search term “Qatar” to the Google query finally yielded results for a company in Doha called Orbis Systems W.L.L. It was listed as a contractor on Milipol Qatar, “the international event for homeland security and civil defense in the Middle East” (About Milipol Qatar, 2020). The company boasts its focus on biometric and cryptographic technology solutions for government and corporate clients. The company linked to a website (orbisholdings.com), which makes extensive reference to smart card and “epassport” projects in Qatar and the UAE including NFC and OCR recognition for ID cards. The reasons for the vague to non-existent references regarding the developer are unclear, and only raise more questions about transparency.

2.2. Basic Application Architecture

After unpacking the APK within the smali_classes2 folder, the major components of the application are plainly seen:

```
│── com
│   │── google
│   │── huawei
│   │── orbis
│   │── toptoch
│   └── yinglan
├── net
│   │── glxn
└── org
    ├── altbeacon
    └── apache
```

Anthony Wallace, anthony.wallace11@gmail.com
A majority of the research focused on the orbis folder as it contains all the main functionality of the application. The other folders contain code for different portions of the UI. For example, the “toptache” folder contains graphics for spinning loops used in search interfaces, and the “yinglan” folder contains code for rendering shadows behind pictures display in the UI. The “altbeacon” folder contains resources for the Bluetooth beaconing functionality which also uses the Eddystone beaconing profile developed by Google, but this falls outside of the scope of research. Future research would be useful regarding interactions between other devices and how much of that data is sent via the network connection back to the central database.

3. Research Method

The application is developed for both iOS and Android platforms. Arguments are plenty regarding the ease of use for both these environments, but the mature development platforms, the versatility, readily available instrumentation, and open-source nature of the Android platform are significant drivers behind using it to run experiments. In October 2020, the Play store indicated the Ehteraz app was downloaded more than one million times. The download count indicates the APK provides a good sample of user experience.

Various challenges were encountered when attempting to sniff the network traffic of the APK, so a combination of testing environments was used to gather more data for analysis. An unrooted physical Samsung a10s phone was used to gather some of the data, and an Android VM was used to provide a “rooted” operating system for network traffic modification and manipulation. The VM allowed data from encrypted connections to be decrypted and analyzed.

3.1. Android Virtual Device Testing Environment

Creating the proper environment for testing can be challenging on a stock Android device. Some of the techniques used to observe app behavior and monitor traffic require that the device be “rooted” (having root access to the phone). To facilitate the root access needed to some of the system components, an x86 Android Open Source Project (AOSP) image was downloaded from the Android-x86.org. The image is built for usage of Android on Intel and AMD-based platforms instead of RISC ARM Processors. It was

Anthony Wallace, anthony.wallace11@gmail.com
installed on VMware Fusion Pro for easy snapshot capability (although the free VMware Player could easily be substituted). The image also has a root shell built in.

Additionally, the setup in a virtual environment allows for easy network packet capture via the VMware virtual interface. On the host system (MacBook Pro – macOS 10.15.7), listing the network interfaces with the “ifconfig” command shows there are two virtual network interfaces created by VMware. These are private networks created by VMware for network address translation (NAT) using the host’s physical ethernet/Wi-Fi interface for public internet connections.

```
vmnet1: flags=8863<UP,BROADCAST,SSMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
ether 00:50:56:c0:00:01
inet 172.16.16.1 netmask 0xffffff00 broadcast 172.16.16.255
vmnet8: flags=8863<UP,BROADCAST,SSMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
ether 00:50:56:c0:00:08
inet 192.168.36.1 netmask 0xffffff00 broadcast 192.168.36.255
```

*Figure 6: Virtual network adapters created by VMware on the physical host*

Simply opening Wireshark and clicking on the vmnet8 interface will capture all the traffic to and from the virtual Android device.

```
Capture

...using this filter: [Enter a capture filter ...]

vuunu
utun1
vmnet1

vmnet8

Loopback: lo0
Thunderbolt Bridge: bridge0
Thunderbolt 0: en1
Thunderbolt 0: en2

```

*Figure 7: Wireshark indicating live traffic on the lab network*

All network communication from the app was TLS encrypted except for some DNS queries for “ehtraz.com” therefore simply sniffing the traffic produced very little context about the nature of communication with the application. The open-source Zed Attack Proxy (ZAP) was chosen to intercept the traffic and decrypt the HTTPS streams. The proxy was able to dynamically produce a certificate authority for a miscreant-in-the-middle attack.

Anthony Wallace, anthony.wallace11@gmail.com
Using openssl, the certificate was renamed as the hash of its subject.

```
raw2801@IT-raw2801-20 android-tools % openssl x509 -inform PEM -subject_hash_old -in myca.pem | head -1
844a9dac
raw2801@IT-raw2801-20 android-tools % cp myca.pem 844a9dac.0
```

Figure 9: A specific hash of the certificate is used as a name in the trusted authorities folder

From here, the ZAP certificate was copied to the “trusted authorities” folder on the virtual device, and its permissions, owner, and group were modified so the device would recognize the certificate as legitimate. This addition to the virtual device Certificate Authority folder allowed the TLS connection to proceed and not throw an error for an invalid certificate which would have halted the HTTPS connection. Next, the application was utilizing certificate pinning to defend against the very attack outlined above. The app has certain routines to search for a specific hashed value of the certificate with which to establish connections to the central database/server. The check must be stopped or manipulated with the new certificate hash values from the trusted certificates folder. A code injection tool was deployed to find and “hook” the function while the program is running.

Frida “lets you inject snippets of JavaScript or your own library into native apps on Windows, macOS, GNU/Linux, iOS, Android, and QNX” (Welcome Frida, 2020). The program is able to step through running processes and intercept system calls so they can be examined and potentially manipulated. Using the x86 version of Frida server, the process list from the phone can be listed, and the PID of the Ehteraz application is obtained.

```
[raw2801@IT-raw2801-20 android-tools % /adb push frida-server-12.8.9-android-x86_64 sdcard/Download/
frida-server-12.8.9-android-x86_64: 1 file pushed, 0 skipped. 116.4 MB/s (53372168 bytes in 0.437s)
```

Figure 10: Pushing the Frida server to the simulated sdcard on the Android virtual device

Anthony Wallace, anthony.wallace11@gmail.com
The executable must be copied from the sdcard storage to a tmp folder in the data file so that it will execute. Running the server with the “&” allows it to run in the background and frees the command-line.

```
console: # ls -lah /data/local/tmp/
total 25M
drwxr-x-x 2 shell shell 4.0K 2020-10-22 20:59 .
drwxr-x-x 4 root root 4.0K 2020-10-14 17:57 ..
-rw-r--r-- 1 root sdcard_rw 51M 2020-10-22 20:59 frida-server-12.8.9-android-x86_64
console: # chmod 755 /data/local/tmp/frida-server-12.8.9-android-x86_64
console: /data/local/tmp/frida-server-12.8.9-android-x86_64
```

![Figure 11: Running the Frida server on the Android virtual device](image)

The server creates a listen service on the device designed to interact with Frida to allow a user to pause a running application and “step through” its process flow. By doing this, the user can manipulate the way the program executes.

```
raw2801@IT-raw2801-20 android-tools % frida-ps -O
PID  Name
---- ---------------
 7041  adb
 1902 android.hardware.audio@2.0-service
 1903 android.hardware.bluetooth@1.0-service.btlinux
 1904 android.hardware.camera.provider@2.4-service
 1905 android.hardware.cam@1.0-service
 1906 android.hardware.configstore@1.1-service
 1907 android.hardware.dumpstate@1.0-service
 1908 android.hardware.light@2.0-service
 1909 android.hardware.memtrack@1.0-service
 1910 android.hardware.power@1.0-service
 1911 android.hardware.usb@1.0-service
 1912 android.hardware.wifi@1.0-service
 1900 android.hidl.allocator@1.0-service
 7228 android.process.media
 1913 audioserver
 1920 camerarserver
 3837 com.android.chrome
 3896 com.android.chrome:privileged_process0
 2160 com.android.inputmethod.latin
 2551 com.android.launcher3
 6255 com.android.tp
 2243 com.android.phone
 6965 com.android.printspooler
 4298 com.android.settings
 4388 com.android.settings.intelligence
 2173 com.android.systemui
 5885 com.android.vending
 2604 com.farmerbob.taskbar.androidx6
 7546 com.google.android.gms
 2615 com.google.android.gms.persistent
 2534 com.google.android.googlequicksearchbox:interactor
 5413 com.google.android.partnersetup
 2568 com.google.android.setupwizard
 6815 com.google.process.gapps
 2663 com.google.process.gservices
 6846 com.moi.covid19
```

![Figure 12: "frida-ps" lists all the running processes on the virtual device](image)

Anthony Wallace, anthony.wallace11@gmail.com
Frida then attaches to that process (in this case PID 6846 – “com.moi.covid19”) and can watch for calls to SSL encryption components. Multiple plugins written by the Frida developer community (in JavaScript) can be used to hook specific SSL libraries used for pinning. Deploying some of these scripts is incredibly simple. Built into the Frida tool is the option to pass scripts located in “codeshare.frida.re” — a repository for scripts of other Frida users. Running the following command will load a script that attempts to bypass multiple SSL libraries that enable certificate pinning.

```bash
$ frida --codeshare akabel/frida-multiple-unpinning -f Ehteraz_v9.0.2.apk
```

In previous versions of the application (specifically v7.0.5), the developers used the “okhttp3” library to implement certificate pinning in the APK. One of these JS codes was able to hook this function, and the ZAP proxy certificate hash could be substituted in place of the expected value. Unfortunately, the most recent version used in the experiments (v9.0.2) ceased using okhttp3, and the developers began to use native SSL libs in the Android platform to accomplish the SSL/TLS connections, which hindered the ability to hook the function. Lack of JS experience rendered the Frida tool useless against the new version’s certificate pinning methods. However, there are other methods to bypass this certificate-checking mechanism. Following an example from blogger Cody Wass, a more direct approach was taken.

Wass explains the smali code in the unpacked APK needs to be directly modified with hashes of the new ZAP certificate and then repackaged (Wass, 2020). He states: “Overwriting the...certificate with [the] custom CA should allow us to trick the application into accepting our certificate” (Wass, 2020). In order to find the smali code which contained the variable for the pinned hash, grep ran through the entire APK for specific keywords (not case-sensitive): cert, sha, pinning, pinned. Finally, one search string worked, producing the folder with the file containing constants the app relied on for this function.

Anthony Wallace, anthony.wallace11@gmail.com
After finding the value location in the “Constants.smali” file, inserting the proper value of the ZAP certificate is trivial. A thread on stackoverflow was able to quickly assist with the proper format for the new string to be inserted (Stackoverflow, 2020).

Using vi, this string was inserted in place of the other constants, and the apktool was used to repack the APK.

The next obstacle, as a part of the app’s design, was a refusal to follow system proxy settings. The app ignored any settings in the Wi-Fi proxy fields for the Android, so via the terminal, iptables was used to steer all HTTPS and HTTP traffic to our ZAP proxy.

These modifications finally allowed unencrypted data to flow through ZAP to the Ehteraz server.

3.2. Samsung Galaxy A10s

While much of the research was done on a VM, a standard Samsung with factory defaults purchased off the shelf in Qatar was used for testing on an unrooted physical device. According to some comments from users, the application was

Anthony Wallace, anthony.wallace11@gmail.com
equipped with some defenses against analysis such as “jailbreaking” detection for iOS. In order to observe “normal” behavior, this physical device was unrooted and unmodified. The drawback to this testing mode was the network traffic from the phone to the backend servers was TLS encrypted. However, valuable metadata such as connection frequency and upload/download sizes was obtained.

A dedicated Wi-Fi access point was created by enabling internet sharing on a MacBook Pro. The Mac shared an ethernet connection to the Wi-Fi adapter for internet access, and the wireless device on the network was the test Samsung device. The app was opened on the device and left connected to Wi-Fi overnight (approximately 10 hours). The tcpdump tool was used for packet capture.

![tcpdump output](image)

**Figure 17:** Using tcpdump to capture traffic moving across the isolated lab network

4. Analysis

Wireshark is a recognized and well-created tool used to dissect the information gathered in the long-term capture session. While the session was encrypted, there is valuable data to be gleaned from the metadata. The tool has the ability to break down the conversation between the physical testing device and the rest of the internet. Once the individual conversations are extracted, users can see the amount of data transferred and with whom. Additionally, timing information like how often the app checks in with the centralized server is recorded for analysis.

The following statistics were obtained running the app on the physical Samsung device for approximately 9 hours and 48 minutes. All four of the IP addresses labeled

Anthony Wallace, anthony.wallace11@gmail.com
“Address A” resolve to “ehtraz.com” at the time of publication. No other connections were observed to other resources.

<table>
<thead>
<tr>
<th>Address A</th>
<th>Address B</th>
<th>Packets</th>
<th>Bytes</th>
<th>Packets A -&gt; B</th>
<th>Bytes A -&gt; B</th>
<th>Packets B -&gt; A</th>
<th>Bytes B -&gt; A</th>
<th>Rel Start</th>
<th>Duration (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>51.136.74.166</td>
<td>192.168.2.2</td>
<td>753</td>
<td>211639</td>
<td>390</td>
<td>162859</td>
<td>363</td>
<td>48780</td>
<td>28.66</td>
<td>23669.48</td>
</tr>
<tr>
<td>20.50.171.191</td>
<td>192.168.2.2</td>
<td>555</td>
<td>157804</td>
<td>287</td>
<td>122655</td>
<td>268</td>
<td>35149</td>
<td>1231.79</td>
<td>21223.06</td>
</tr>
<tr>
<td>20.50.9.15</td>
<td>192.168.2.2</td>
<td>518</td>
<td>150495</td>
<td>270</td>
<td>117226</td>
<td>248</td>
<td>33269</td>
<td>631.13</td>
<td>6775.70</td>
</tr>
<tr>
<td>20.50.170.60</td>
<td>192.168.2.2</td>
<td>96</td>
<td>26062</td>
<td>48</td>
<td>20452</td>
<td>48</td>
<td>5610</td>
<td>2493.18</td>
<td>3711.85</td>
</tr>
</tbody>
</table>

4.1. **Experiment Data from v7.0.5**

In order to provide context for the encrypted data that was captured in the table above, decrypted data gathered from previous experiments and testing on v7.0.5 of the app should be presented and discussed. This older version was using the “okhttp3” library for cert pinning and was able to be hooked by the Frida server (see Section 3.1). Using a similar setup with a rooted Android VM revealed more information because the okhttp3 library was easier to “unpin” and capture successful handshakes via the ZAP proxy. Again, the only domain to which the application attempts connections is “ehtraz.com.” These appear to be application servers hosted in the Azure cloud based on basic WHOIS queries of the returned IP addresses. The URLs include API calls to the “BioTrace” application. There seems to be no public reporting connected to this app, and initial searches regarding “biotrace” yield no data.

The following figure shows a sample that represents the traffic of the decrypted information passed to the application server via ZAP. Specifically, the registration process is observed while using the VM in the virtual testing environment. Similarities between the packet sizes and the encrypted metadata indicate the functionality does not appear to have changed significantly.

Anthony Wallace, anthony.wallace11@gmail.com
After careful study of the decrypted traffic captured by ZAP, the following table postulates the purpose of each API call based on observed behavior:

<table>
<thead>
<tr>
<th>API Call</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>getConfigurationObfuscated</td>
<td>Pulls an app config from server like heartbeat timing and beacon intervals</td>
</tr>
<tr>
<td>getQRCodeDetailedObfuscated</td>
<td>Pulls QR Code information unique to a user (See Appendix for QR data example)</td>
</tr>
<tr>
<td>getPhoneStatusObfuscated</td>
<td>Looks up if phone is “Active in the Database”</td>
</tr>
<tr>
<td>getPeronGeoStatus</td>
<td>Possible misspelling of “person”</td>
</tr>
<tr>
<td>getPersonStatusObfuscated</td>
<td>Appears to display “GeoTracing” status</td>
</tr>
<tr>
<td>publishDeviceHeartBeatObfuscated</td>
<td>Creates an interval for the device’s “heartbeat” based on a timestamp and a “major” and “minor” variable</td>
</tr>
<tr>
<td>getMOHInformationObfuscated</td>
<td>Statistics populated on a stats menu in app</td>
</tr>
<tr>
<td>getNotificationsObfuscated</td>
<td>MOPH Notifications and news bulletins</td>
</tr>
</tbody>
</table>
4.2. Implications for Future Research

The research process was interrupted by a number of complications in addition to the issues previously explained in section 3.1. When attempting to re-register other devices for testing, a lockout policy for requests using a single mobile phone number caused registration to error out. Fortunately, using a different phone number with the same Qatar ID and expiration date allowed the registration process to continue. However, this required the purchase of additional SIM cards in Qatar which may raise suspicion for a tester in that country. Each phone number must be tied to a Qatar ID number. Perhaps engaging the developer (Orbis Holdings) and the proper government authorities can create an opportunity for registration of testing accounts. This would prevent lockout mechanisms or rate limiting that could hinder future research.

Furthermore, closer inspection of the Bluetooth beaconing protocol (specifically Eddystone) could yield additional vectors for information leaks. Using a tool such as BluSee on macOS was explored during the research period but is beyond the scope of the network traffic analysis. Watching this Bluetooth interaction with another Ehteraz device may reveal more interesting functionality in both the beaconing protocol and network protocols, as none of the experiments tested to see what network traffic was potentially triggered by Bluetooth interaction. It would be a better test of the full range of the application’s capabilities if this interface was monitored on a physical device. The VM-centric nature of the experiments limited that particular line of observation.

5. Conclusion

While the Ehteraz application has sweeping permissions that may rile some privacy conscious users, tests indicate only telemetry, location data, and basic identity information are the types of data uploaded to the central database for registration and

Anthony Wallace, anthony.wallace11@gmail.com
processing. It does bear noting that if the current permissions remain in the AndroidManifest.xml file that future updates to the app could be abused to gather other data from the phone in addition to the data already gathered. Continuous monitoring of each update should be performed to keep the government and application developers honest about the nature of its function and purpose.

Further testing of the server-side infrastructure would be required to ensure that it was not leaking any data, as in the previously cited example of the Amnesty report. Overall, the app seems to be carrying out the functions described by the public pronouncements of the MOI in Qatar. For now, users can rest assured this version of the application (at least at the time of writing) is not sending their photos or personal data to the Ministry of Interior in Qatar.
References

*Bahrain, Kuwait and Norway contact tracing apps a danger for privacy.* (2020, June 16).

kuwait-norway-contact-tracing-apps-danger-for-privacy/

*Qatar: Contact tracing app security flaw exposed sensitive personal details of more than one million.* (2020, May 26). Amnesty International.

https://www.amnesty.org/en/latest/news/2020/05/qatar-covid19-contact-tracing-
app-security-flaw/


https://developer.android.com/reference/android/Manifest.permission


https://blog.netspi.com/four-ways-bypass-android-ssl-verification-certificate-
pinning/

Anthony Wallace, anthony.wallace11@gmail.com
Appendix

SHA Hash of the tested APKs:
shasum -a256 EHTERAZ_v9.0.2.apk -> dec29ba902d3bd17ad69c5ceccc3e3813582c63ed4ec8bb1bf3ed66ab4c7a6
shasum -a256 com.moi.covid19-v7.0.5.apk -> 5611b46c882c475af5e592f6971f1c99d194b11a3bf8b56861324f71a8e907

TLS Cert for Ehtraz.com:

-----BEGIN CERTIFICATE-----
MIIGezCBBpugAwIBAgIQCDCzDQYJKoZIhvcNAQEFBQADggEBAD8+iutuye62EjVdDuah3mCc7Sto+j1sX4GRhbfelad4L37Y0zYPtPgb+28jAvnVlMnx0M7nHRpJLd73iSHE4a5/PmHyeSd2dht49P1rroRHuzix3EKhto9Mb9j1v1yEg125i7a7jo4+enKc5F0crqgfo+eagyjggqLkhrURGcte9fDedPzwrrLcrfVNgwLwFfDPeAs9udq/BisIgeds2UlrRg1HDBC112kFtscMoXsR13AhMvFzGtse6Eb7mDoSgddPKkaM+mFeEvkYQG/1pGUSdfx/QYlIc0c0v9E7utUrsmdX/dMWQhrM2QsNX06dG4a2ksVg8olKhwa7r8wUr-----END CERTIFICATE-----

Anthony Wallace, anthony.wallace11@gmail.com
Example of Embedded QR Metadata:

<!--Embedded QR Metadata-->

Anthony Wallace, anthony.wallace11@gmail.com