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 unwieldly 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.

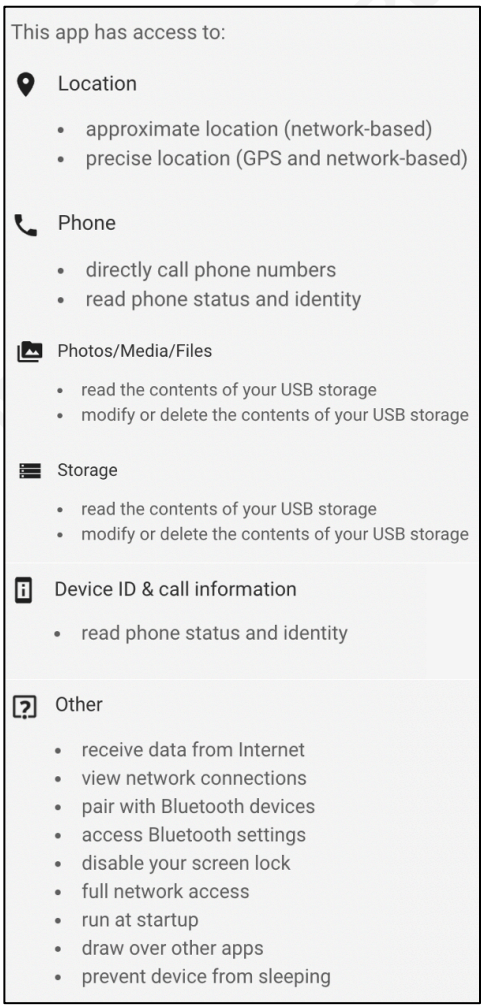

Figure 1: Ehteraz Permissions

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.

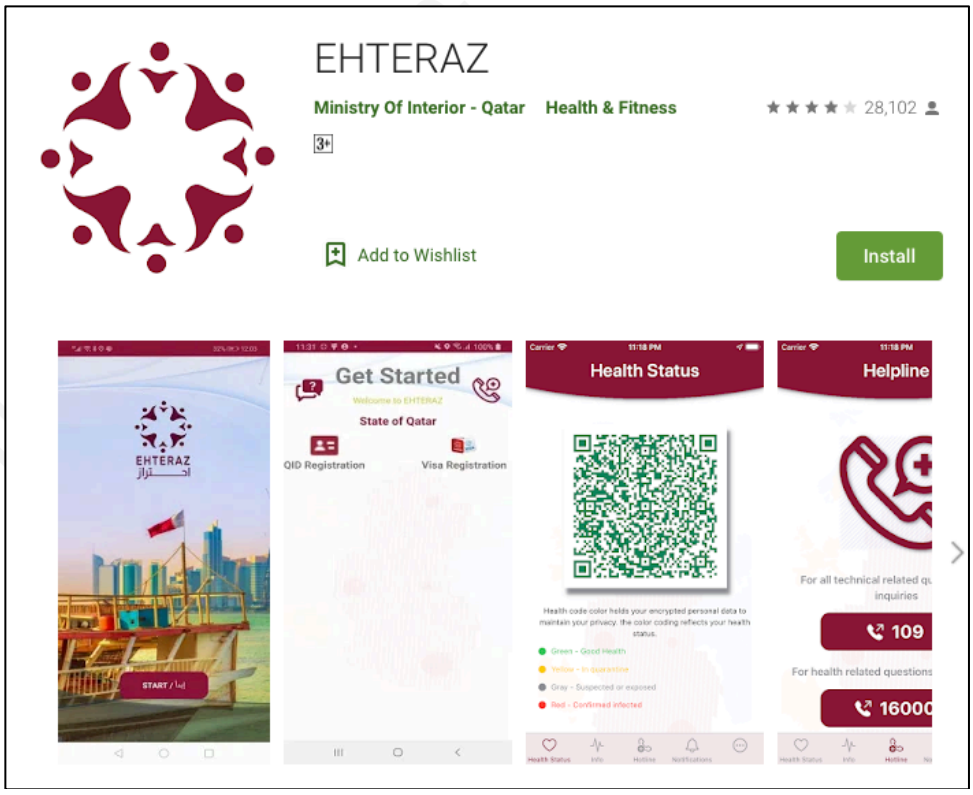

Figure 2: Ehteraz in the Google Play Store

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

```
raw2801@IT-raw2801-20 ehtraz-analysis % apktool d EHTERAZ_v9.0.2.apk
I: Using Apktool 2.4.1 on EHTERAZ_v9.0.2.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /Users/raw2801/Library/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Baksmaling classes2.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

Figure 3: Decompiling the Ehteraz APK with apktool

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 "AndroidManaifest.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.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res
/android" android:compileSdkVersion="29" android:compileSdkVersionCodename="10" package="com.moi.covid19" platfor
mBuildVersionCode="29" platformBuildVersionName="10">
    <uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
    <uses-feature android:name="android.hardware.location.gps"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.BLUETOOTH" android:required="false"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" android:required="false"/>
    <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
    <uses-feature android:glEsVersion="0x00020000" android:required="true"/>
    <permission android:name="com.moi.covid19.permission.C2D_MESSAGE" android:protectionLevel="signature"/>
    <uses-permission android:name="com.moi.covid19.permission.C2D_MESSAGE"/>
    <uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES"/>
    <uses-permission android:name="com.huawei.appmarket.service.commondata.permission.GET_COMMON_DATA"/>
```

Figure 4: Permissions listed in the AndroindManifest.xml file

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.

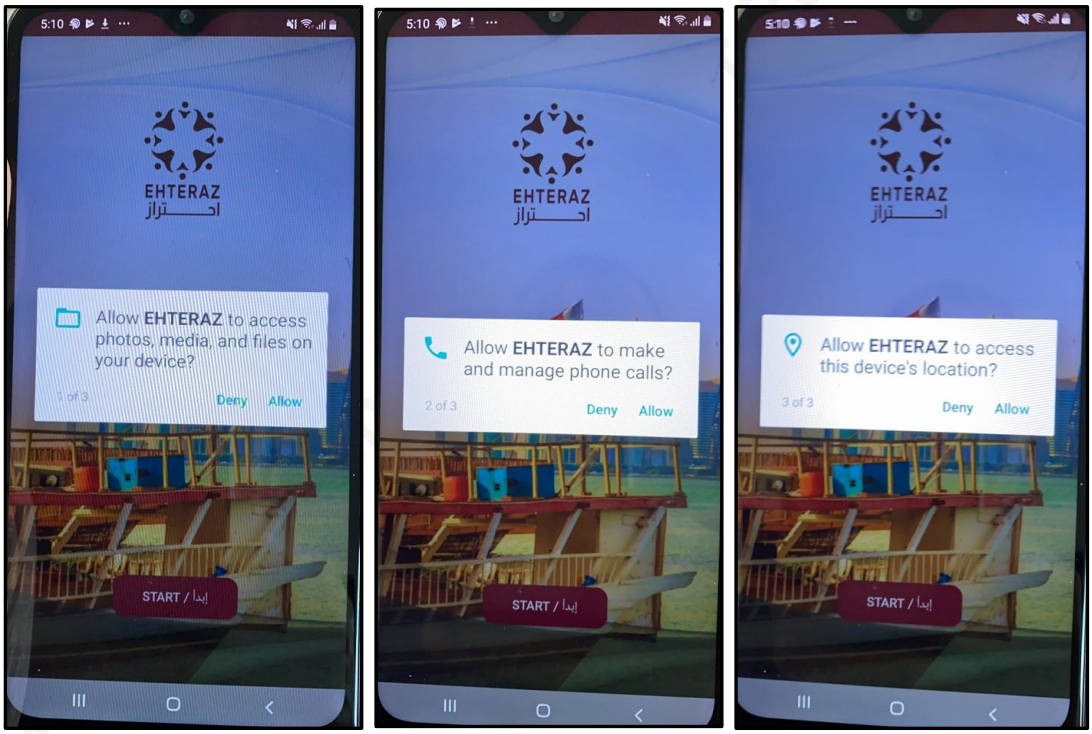

Figure 5: Permissions requests upon the first application launch

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

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”.

```
EHTERAZ_v9.0.2-direct-device-pull/smali_classes2/com/orbis/ehteraz/
```

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
│   ├── toptoche
│   └── yinglan
├── net
│   └── glxn
└── org
    ├── altbeacon
    └── apache
```

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 "toptoche" 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

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,SMART,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,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 00:50:56:c0:00:08
        inet 192.168.36.1 netmask 0xffffff00 broadcast 192.168.36.255
raw2801@IT-raw2801-20 ~ %
```

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.

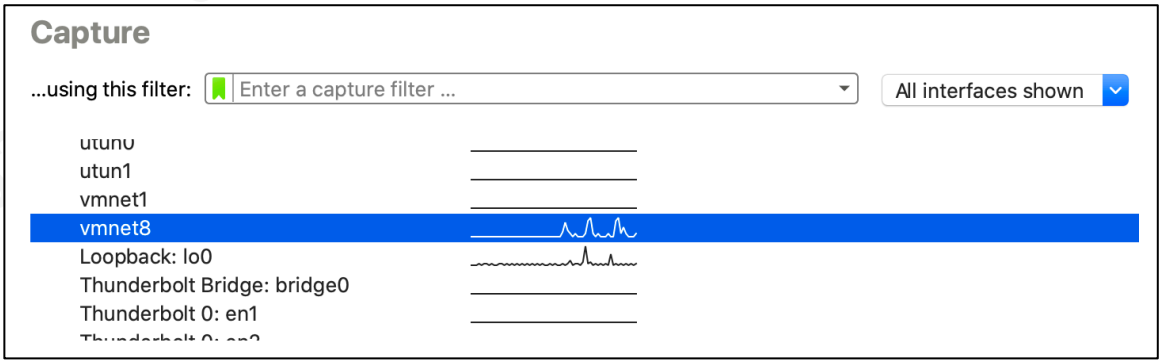

Figure 7: Wireshark indicating live traffic on the lab network

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

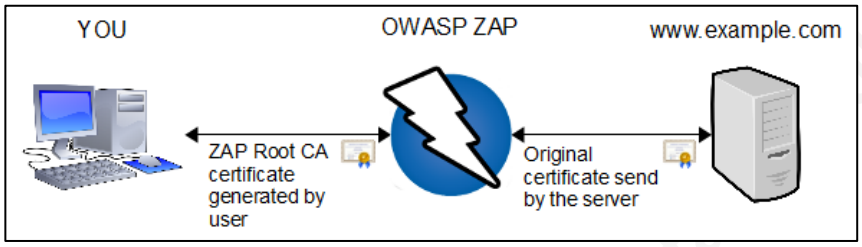

Figure 8: Zed Attack Proxy performing Miscreant-in-the-Middle

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

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
drwxrwx--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-rw---- 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 &
[1] 7629
console:/ #
```

Figure 11: Running the Frida server on the Android virtual device

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 -U
 PID  Name
----  -----------------------------------------------
7041  adbd
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.cas@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  cameraserver
3837  com.android.chrome
3896  com.android.chrome:privileged_process0
2160  com.android.inputmethod.latin
2551  com.android.launcher3
6255  com.android.mtp
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.farmerbb.taskbar.androidx86
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

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.

```
$ frida --codeshare akabe1/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.

```
[raw2801@IT-raw2801-20 EHTERAZ_v9.0.2-direct-device-pull % grep -ri cert smali_classes2/com/orbis

smali_classes2/com/orbis/ehteraz/Utils/Constants.smali:.field public static SHA_CERT_BIOTRACE:Ljava/lang/String; =
 "84Sxoh3gIOWfw3f/7sYGl+4PJVJWSJBOZJo665c8cFM="
smali_classes2/com/orbis/ehteraz/Utils/Constants.smali:.field public static SHA_CERT_EHTERAZ:Ljava/lang/String; =
"84Sxoh3gIOWfw3f/7sYGl+4PJVJWSJBOZJo665c8cFM="
```

Figure 13: "grepping" the entire folder for strings related to certificates

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).

```
[raw2801@IT-raw2801-20 android-tools % openssl x509 -in myca.pem -pubkey -noout | openssl pkey -pubin -outform der
 | openssl dgst -sha256 -binary | openssl enc -base64
5LFl0s+boJm3JnvpIxWmPf/uxvyTcgUsuE3W+jGvA8s=
```

Figure 14: A very specific command used to produce the proper encoding for the certificate

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

```
[raw2801@IT-raw2801-20 ehtraz-analysis % apktool b EHTERAZ_v9.0.2_apkpure.com
I: Using Apktool 2.4.1
I: Checking whether sources has changed...
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```

Figure 15: Using the "b" option builds a package

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.

```
127|console:/ # iptables -t nat -A OUTPUT -p tcp --dport 443 -j DNAT --to 192.168.36.1:8080
iptables -t nat -A OUTPUT -p tcp --dport 80 -j DNAT --to 192.168.36.1:8080
```

Figure 16: "iptables" used to force traffic through ZAP

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

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.

```
[raw2801@IT-raw2801-20 ~ % ifconfig bridge100
bridge100: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=3<RXCSUM,TXCSUM>
        ether 3e:22:fb:ce:52:64
        inet 192.168.2.1 netmask 0xffffff00 broadcast 192.168.2.255
        inet6 fe80::10eb:92fd:9ce5:f1f7%bridge100 prefixlen 64 secured scopeid 0x14
        Configuration:
                id 0:0:0:0:0:0 priority 0 hellotime 0 fwddelay 0
                maxage 0 holdcnt 0 proto stp maxaddr 100 timeout 1200
                root id 0:0:0:0:0:0 priority 0 ifcost 0 port 0
                ipfilter disabled flags 0x0
        member: ap1 flags=3<LEARNING,DISCOVER>
                ifmaxaddr 0 port 5 priority 0 path cost 0
        Address cache:
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
[raw2801@IT-raw2801-20 ~ % sudo tcpdump -i bridge100 -nn -w run-app-20201009-0908.pcap -v
[Password:
tcpdump: listening on bridge100, link-type EN10MB (Ethernet), capture size 262144 bytes
^C0 packets received by filter
0 packets dropped by kernel
```

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

"Address A" resolve to "ehtraz.com" at the time of publication. No other connections were observed to other resources.

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

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.

stamp	Method	URL	Code	Reason	RTT	Size Resp. Body
3:43:54 PM	GET	https://ehtraz.com/BioTrace/api/BioTrace/getConfigurationObfuscated	200	OK	655 ms	1,122 bytes
3:44:52 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/requestOTPMOIObfuscated	200	OK	5.08 s	66 bytes
3:45:55 PM	GET	https://ehtraz.com/BioTrace/api/BioTrace/getConfigurationObfuscated	200	OK	1.32 s	1,122 bytes
3:46:07 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/requestOTPMOIObfuscated	200	OK	3.57 s	61 bytes
3:46:38 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/registerPhoneMOIObfuscated	200	OK	6.02 s	116 bytes
3:46:45 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getQRCodeDetailedObfuscated	200	OK	1.1 s	193 bytes
3:46:48 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getPeronGeoStatus	200	OK	604 ms	138 bytes
3:46:48 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getPersonStatusObfuscated	200	OK	1.1 s	163 bytes
3:46:48 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getPhoneStatusObfuscated	200	OK	2.35 s	46 bytes
3:47:30 PM	GET	https://ehtraz.com/BioTrace/api/BioTrace/getMOHInformationObfuscated	200	OK	1.1 s	362 bytes
3:47:56 PM	GET	https://ehtraz.com/BioTrace/api/BioTrace/getConfigurationObfuscated	200	OK	685 ms	1,122 bytes
3:48:20 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getNotificationsObfuscated	200	OK	576 ms	4,614 bytes
3:52:36 PM	GET	https://ehtraz.com/BioTrace/api/BioTrace/getConfigurationObfuscated	200	OK	551 ms	1,122 bytes
3:52:37 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getQRCodeDetailedObfuscated	200	OK	801 ms	193 bytes
3:52:40 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getPhoneStatusObfuscated	200	OK	695 ms	46 bytes
3:52:40 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getPeronGeoStatus	200	OK	1.03 s	138 bytes
3:52:42 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/publishDeviceHeartBeatObfuscated	200	OK	755 ms	102 bytes
3:52:40 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getPersonStatusObfuscated	200	OK	7.38 s	163 bytes
3:52:50 PM	GET	https://ehtraz.com/BioTrace/api/BioTrace/getMOHInformationObfuscated	200	OK	630 ms	362 bytes
3:52:55 PM	POST	https://ehtraz.com/BioTrace/api/BioTrace/getNotificationsObfuscated	200	OK	850 ms	4,614 bytes
3:53:31 PM	GET	https://ehtraz.com/BioTrace/api/BioTrace/getConfigurationObfuscated	200	OK	1.1 s	1,122 bytes
3:54:37 PM	GET	https://ehtraz.com/BioTrace/api/BioTrace/getConfigurationObfuscated	200	OK	791 ms	1,122 bytes

Figure 18: The decrypted API calls to ehtraz.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:

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

Anthony Wallace, anthony.wallace11@gmail.com

requestOTPMOIObfuscated	One Time Password request via SMS at registration
registerPhoneMOIObfuscated	Submitting the OTP for registration

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

processing. It does bear noting that if the current permissions remain in the AndroidManaifest.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). Amnesty International. https://www.amnesty.org/en/latest/news/2020/06/bahrain-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/

Manifest.permission : Android Developers. (2020, September 9). https://developer.android.com/reference/android/Manifest.permission

Welcome Frida. (2020, November 12). https://frida.re/docs/home/

Wass, C. (2020, September 28). Four Ways to Bypass Android SSL Verification and Certificate Pinning. Retrieved September 19, 2020, from 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 ->
dec29ba902d3bd17ad69c5ceeccc3e3813582c63ed4bc68b1bff3ed66ab4c7a6

shasum -a256 com.moi.covid19-v7.0.5.apk ->
5611b46c882c475af5e592f6971f1c99d194b11a3bff18b56861324f71a8e907
```

TLS Cert for Ehtraz.com:

```
-----BEGIN CERTIFICATE-----
MIIGEzCCBPugAwIBAgIQDc40QkDb5ElWcFNoqwbJNjANBgkqhkiG9w0BAQsFADBN
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMScwJQYDVQQDEx5E
aWdpQ2VydCBTSEEyIFNlY3VyZSBTZXJ2ZXIgQ0EwHhcNMjAwNDIwMDAwMDAwWhcN
MjEwNDI4MTIwMDAwWjBQMQswCQYDVQQGEwJRQTENMAsGA1UEBxMERG9oYTEdMBsG
A1UEChMUTWluaXN0cnkgb2YgSW50ZXJpb3IxEzARBgNVBAMTCmVodHJhei5jb20w
ggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDkrDQF+ci+F6bz5qJo9A9v
DlzGxkKOjpTCASPPHy52hNotJm+JI7Tpbp91gM6izEkyxoCYPsp7AFufSQz9rLQN
rDfqXDgzCna2BHL53u+EmwVXEqdKzqSYHdclMKHVyLh+UpYwPFis8KU9A1hCLgk8
nRy7WiMdZSXMQziABAGAKi1S/C6z4apHThVi8hXNMOWg6aeYir0a9BrW2UjNsNLT
xlkUY9X3uGcIXLOqDAv/bgjDcPM1qRXpUCl5tp+U3Y7HZYqVIRdGcw9qcfns5DlJ
tztChmBH94e1x4YI6/h5ecWv3jsGWILeY7HWfrkB57wqxflrzfe7cz2BJMaLi9VL
AgMBAAGjggLqMIIC5jAfBgNVHSMEGDAWgBQPgGEcgjFh1S8o541GOLQs4cbZ4jAd
BgNVHQ4EFgQUmScT8I2D3zRQ1i0Uhb5mWnHS1Q0wJQYDVR0RBB4wHIIKZWh0cmF6
LmNvbYIOd3d3LmVodHJhei5jb20wDgYDVR0PAQH/BAQDAgWgMB0GA1UdJQQWMBQG
CCsGAQUFBwMBBggrBgEFBQcDAjBrBgNVHR8EZDBiMC+gLaArhilodHRwOi8vY3Js
My5kaWdpY2VydC5jb20vc3NjYS1zaGEyLWc2LmNybDAvoC2gK4YpaHR0cDovL2Ny
bDQuZGlnaWNlcnQuY29tL3NzY2Etc2hhMi1nNi5jcmwwTAYDVR0gBEUwQzA3Bglg
hkgBhv1sAQEwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cuZGlnaWNlcnQuY29t
L0NQUzAIBgZngQwBAgIwfAYIKwYBBQUHAQEEcDBuMCQGCCsGAQUFBzABhhhodHRw
Oi8vb2NzcC5kaWdpY2VydC5jb20wRgYIKwYBBQUHMAKGOmh0dHA6Ly9jYWNlcnRz
LmRpZ2ljZXJ0LmNvbS9EaWdpQ2VydFNIQTJTZWN1cmVTZXJ2ZXJDQS5jcnQwDAYD
VR0TAQH/BAIwADCCAQUGCisGAQQB1nkCBAIEgfYEgfMA8QB3APZclC/RdzAiFFQY
CDCUVo7jTRMZM7/fDC8gC8xO8WTjAAABcZeZdKwAAAQDAEgwRgIhAJR02n/8ig7i
TfgBEkqhjm8x8OODJ8ROTYf/E+WCUtSwAiEA9JaalmFX0oyPR/qQZeRXQr2eEHd+
lhT4nKCA8wJe7nQAdgBc3EOS/uarRUSxXprUVuYQN/vV+kfcoXOUsl7m9scOygAA
AXGXmXTaAAAEAwBHMEUCIQCTKUxh8ZTHWA9k+GAfC+YQn906GZX8fcf4866DoVyl
zAIgBR70Ur+qkrAsRue2ZZLO/6HT3JAs/bcatilyIeNqa0swDQYJKoZIhvcNAQEL
BQADggEBAD8/iutuye62EjVdDUah3mCc7Sto+j1sX4GRhbfelad4L37Y0zYPtPgb
+28jAvnVlMnx0mGTnHRjpLJdj73SiHSEa4/5PmYHeySdD2dht49PlrroRHuzix3E
KHtOb9Mjv0lYEg1Z5IqA7Ajo4+EnKc5F0crgf0+eagj0gqLKh6rURGctCe9fDepF
zwwrLCrfVNqwLfWFBDPeAS9uDg/BSIqeds2U3trRgHDBWClZkFtscMoXsRl3AHMv
FzGts6EbTmDoSGddPRkaM+feEvkyGQ/1pGUSfdx/QYlIQc0v9E7tUrsmxD/dMWQh
rM2QsNX0OdG4a2ksVG8o1KGhWa7wr8U=
-----END CERTIFICATE-----
```

Anthony Wallace, anthony.wallace11@gmail.com

Example of Embedded QR Metadata:

<imageXML> <cte> <v1>[REDACTED QID NUMBER]</v1> <v2>1</v2>
<v3>#11855C</v3> <v4>ROBERT ANTHONY WALLACE II</v4> <v5>0</v5>
<v6>10/23/2020 1:15:18 PM</v6> <v7>روبرت انتونى</v7> </cte>
<DS>AQAFAgQAAACRvn9lH8zASeexxww93JoRKYbXKmA3o34Zby5MFhvim1Tn/Cbsap4FISE
OOv5MFD+2Nb0WFuASGUyQWeEYOzQ8N8Dp00cHQv4EnQ6NyydkC6E5wYHJw6DPstZP8WvfXD
0CgoawO/bbfsCDm2QxlPgGAQjG6JHa8P0CbHOjR3MpEPttNew7kMgWQEVtLt1y3kPVchbwi
K8vnXlYqvgrMTkvdWQyWU3JA0XE7PIocLYU5SWPlGj5enO2Edwp1yEG+kD7MR8m0FK58owO
q/206f+uevks0atDKPmRVlbRfFJQOaVc2n/Pyr2KKdun6W6W9uhRtrn3bg8Z7y5uTVbiYlI
p</DS></imageXML>

Anthony Wallace, anthony.wallace11@gmail.com