REVIEW
FACILITY.EU

Name: NGI Emergency Tech Review Facility High-level assessment Immuni

Contract nr: LC-04199045

Date of signature: 15-01-2021

Name of the deliverable: Deliverable 3.2 – Penetration test report Immuni

Authors: Pierre Pronchery, Patricia Piolon, Stefan Marsiske, Marcus Bointon

Level of distribution: Confidential, only for members of the facility (including the Commission Services)

Confidential: Yes

Penetration Test Report

Extraordinary Commissioner for the COVID-19 Emergency

V 1.0
Diemen, September 21st, 2020

Document Properties

Client	Extraordinary Commissioner for the COVID-19 Emergency
Title	Penetration Test Report
Target	Immuni app
Version	1.0
Pentesters	Abhinav Mishra, Pierre Pronchery, Stefan Marsiske
Authors	Pierre Pronchery, Patricia Piolon, Stefan Marsiske, Marcus Bointon
Reviewed by	Patricia Piolon, Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	July 7th, 2020	Pierre Pronchery	Initial draft
1.0	July 10th, 2020	Patricia Piolon	Review
1.1	July 15th, 2020	Pierre Pronchery	Import a new finding (IMM-006)
1.2	July 21st, 2020	Stefan Marsiske	Added generic GAEN section
1.0	September 21st, 2020	Marcus Bointon	Review

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Overdiemerweg 28 1111 PP Diemen The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1	**Executive Summary**	**5**
1.1	Introduction	5
1.2	Scope of work	5
1.3	Project objectives	5
1.4	Timeline	5
1.5	Results In A Nutshell	6
1.6	Summary of Findings	6
1.6.1	Findings by Threat Level	6
1.6.2	Findings by Type	7
1.7	Summary of Recommendations	7
2	**The GAEN Protocol Overview**	**8**
2.1	The Protocol	8
2.1.1	The GAEN architecture	9
2.1.2	GAEN in a Nutshell	9
2.1.3	Metadata	10
2.1.4	Transmission and Reception	11
2.1.5	Reporting	11
2.1.6	Contact Tracing	11
2.2	GAEN Attacks	12
2.2.1	Implementation Level Issues, Closed-Source Vendor-Controlled Basis	12
2.2.2	False Alert Injection Attacks	13
2.2.2.1	False positive contact attacks by re(p)laying or manipulating beacons	13
2.2.2.2	Impersonating the backend	14
2.2.2.3	Access to be able to upload keys	14
2.2.3	Tracking Attacks	14
2.2.3.1	Installation and updates require non-anonymous accounts	14
2.2.3.2	De-anonymization attack of infected persons	15
2.2.3.3	Possible identification of sending device type trough TX Power Value in the Metadata	15
2.2.3.4	Contact data is stored on the phone	16
3	**Methodology**	**17**
3.1	Planning	17
3.2	Risk Classification	17
4	**Reconnaissance and Fingerprinting**	**19**
5	**Findings**	**20**

5.1 IMM-006 — (Android) The Build Is Not Reproducible 20
5.2 IMM-001 — (Android/iOS) Missing Jailbreak/Root detection checks 21

6 Non-Findings 24
6.1 NF-000 — (Android) Application Permissions Review 24
6.2 NF-002 — (iOS) Cache Database Being Created 24
6.3 NF-003 — (Android/iOS) Bypassing the client side control UploadDisabler 25

7 Future Work 26

8 Conclusion 27

9 Bibliography 28

Appendix 1 Testing team 30

1 Executive Summary

1.1 Introduction

The Immuni has received a basic quick security evaluation from Radically Open Security B.V. as part of the Emergency Tech Review Facility. The goal of the review is to provide advice and input to consider in the further development of your project. As part of this facility, selected projects are allocated 5 persondays from ROS for a quick security evaluation.

The Emergency Tech Review Facility is an initiative of the European Commission to independently review and assess technologies developed to fight COVID-19. This review facility is intended as a collaborative, community-focused effort to quickly and transparently analyse solutions brought forward for their applicability, security, and privacy characteristics, and to subsequently fast-track the maturity of the most promising technologies.

The test was carried out between June 22, 2020 and July 15, 2020.

This report contains our findings as well as detailed explanations of exactly how the test was performed.

1.2 Scope of work

The scope of the penetration test was limited to the following target:

- Immuni app

Note that this scope is very restricted. The app itself is a thin wrapper around the GAEN framework, and so contains very little code and presents only a very limited attack surface of its own. One critical omission is that the ability to test TEK uploads was deemed out of scope, as this was not available outside of production configuration, and requires OTP and health worker involvement. These scope limitations drastically reduce opportunities for finding security issues during this audit.

1.3 Project objectives

ROS performed a penetration test with the developers of the Immuni application. The test was intended to gain insight into the security of the app. To do so, ROS accessed Immuni together with the developers, attempting to find vulnerabilities and gain further access and elevated privileges by exploiting any vulnerabilities found.

1.4 Timeline

The Security Audit took place between June 22, 2020 and July 15, 2020.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Moderate issue and 1 Low-severity issue.

The lack of reproducibility in the build process makes it difficult for users of the application to verify that it was effectively built with the source code as published publicly on GitHub.

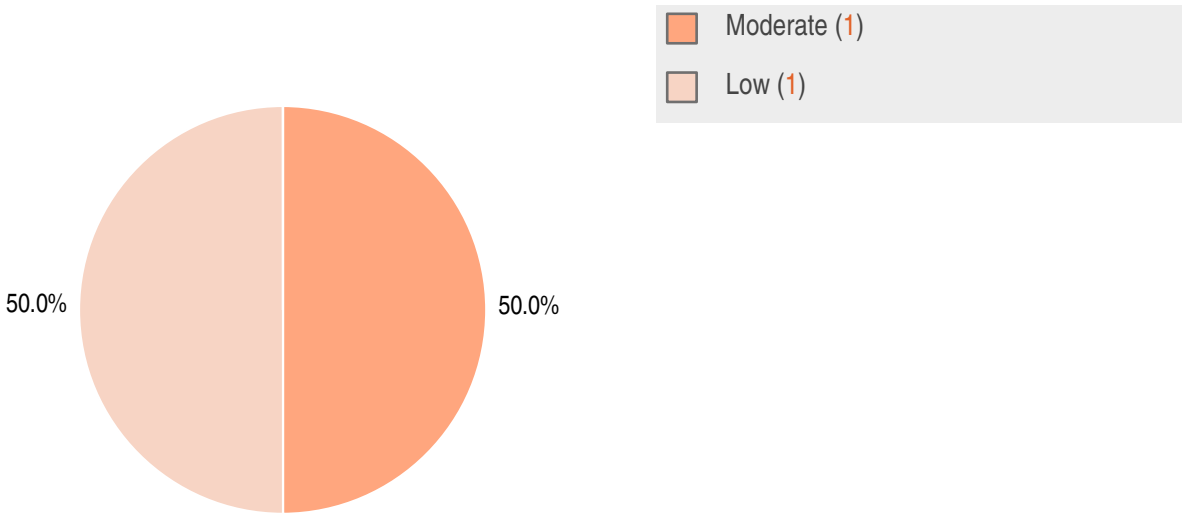
The lack of test for jailbroken devices will let the app run without modification in a potentially hostile or compromised environment.

By exploiting this issue, an attacker might be able to tamper with the data collected and with the general behaviour of the application.

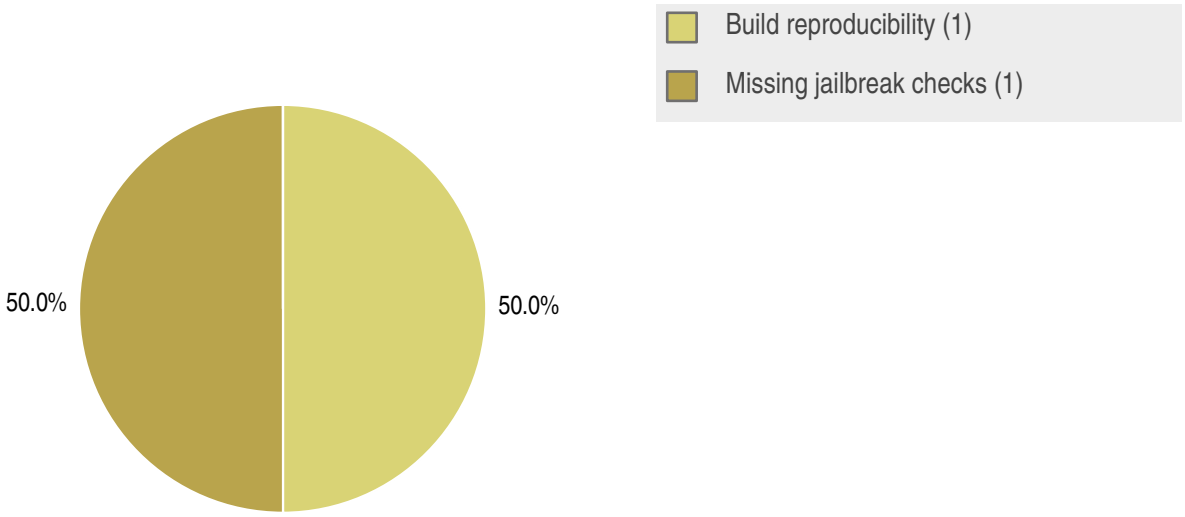
1.6 Summary of Findings

ID	Type	Description	Threat level
IMM-006	Build Reproducibility	Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code.	Moderate
IMM-001	Missing Jailbreak Checks	The apps (iOS and Android) can be installed and run on a jailbroken/rooted device.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
IMM-006	Build Reproducibility	• Complete the efforts initiated towards a completely reproducible build. (Insofar as mobile applications can be)
IMM-001	Missing Jailbreak Checks	• Implement a root/jailbreak detection mechanism on both Android & iOS apps.

2 The GAEN Protocol Overview

The Italian Covid19 notification app Immuni uses the Google/Apple Exposure Notification system as (cryptographic) basis, commonly referred to as GAEN. GAEN implements a large part of the whole solution and its security fundamentals. Therefore we review GAEN before looking at any additions made by the Italian app.

In this section we first give a detailed introduction to the GAEN protocol, with some concerns and notable details mentioned where relevant. In the second part of this section we give an overview of the issues and attacks of the GAEN protocol, illustrated by some – non-exhaustive – notable examples.

2.1 The Protocol

GAEN is a derivative of the Decentralized Privacy-Preserving Proximity Tracing (DP-3T, [12]) protocol, thus the attacks [13][15] against DP-3T likely also apply to GAEN, as would their mitigations, [14][16][17].

GAEN claims to be a decentralized/centralized contact tracing protocol. The decentralized part is the collection of contacts, which is Bluetooth Low-Energy (BLE) based (in contrast to centralised collection based on e.g. telco tracking) and the contact tracing, which happens on the smartphone at the OS level (as opposed to being calculated on a central server), isolated from any apps or third-party software. The centralized part is the app-specific backend and the implementation of the protocol on smartphones. This part is partly closed source (for example Google did not publish the code that handles received beacons), prohibiting access to for example security analysts or developers of free software, and deterring reproducible builds to ensure that the protocol is implemented as specified without backdoors or security vulnerabilities.

Specifications for the GAEN protocol can be found on the websites of both Google and Apple. They have been written independently of each other and historically there have been examples of these two variants having diverging and conflicting content. One such instance for example allowed beacons from Apple devices to be distinguished from beacons from Google devices (the Bluetooth flag `DM_ADV_TYPE_FLAGS` was set for Apple but not Google).

The current (at the time of writing) specification can be found at these locations:

- General information from Google: https://www.google.com/covid19/exposurenotifications/
- General information from Apple: https://www.apple.com/covid19/contacttracing
- Protocol specification by Apple: https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ExposureNotification-CryptographySpecificationv1.2.pdf
- Protocol specification by Google: https://blog.google/documents/69/Exposure_Notification_-_Cryptography_Specification_v1.2.1.pdf

2.1.1 The GAEN architecture

GAEN-based solutions have the following simplified architecture:

```
GAEN implementation ←[restricted API]→ Smartphone App ←{Internet}→ Backend Server(s)
```

The protocol itself only specifies the workings of the implementation behind the API. Everything else, like the app and the backend server, are out of scope and not specified at all. Most of the important mechanisms are implemented by Google/Apple. The notification app itself is (or should be) just a very slim UI wrapper between the API and the Backend Server. The functionality of the backend server is also very limited. This also means that the trusted computing base can be very minimal, and that there is not much to audit if implemented without feature creep and without access to the protocol implementation hidden behind the API.

2.1.2 GAEN in a Nutshell

GAEN operates on globally synchronized 10-minute epochs and 1 day periods (144 epochs per period). All devices must operate on this in a synchronized fashion, with a maximum of ±2 hour deviation. This means that the devices must have a more or less accurate clock, which makes it more expensive to have simple GAEN compatible embedded devices that are not smartphones and are usually not able to cheaply and automatically synchronize the time. There are other protocols which do not have an accurate time-requirement, and are thus simpler to implement on cheaper, purpose-built cheap devices that are not smartphones, however these might have other shortcomings.

At the beginning of each period (day) the devices generates a 16-byte random string called Temporary Exposure Key (TEK). The entropy of this value must be of cryptographic quality. Luckily, Bluetooth devices usually require such for secure operation, thus purpose-built embedded devices will not have a problem with this.

The protocol calculates a new Rolling Proximity Identifier Key (RPIK) for each 10 minute epoch `i`:

```
RPIK(i) = HKDF(TEK[i], NULL, "EN-RPIK", 16)
```

Note that the second parameter provided as NULL is a salt, which can be omitted in this case, as the input material is of high entropy.

When the Bluetooth Low Energy (BLE) MAC address changes, the protocol generates a new Rolling Proximity Identifier (RPI):

```
RPI(i,j)=aes128ecb(RPIK(i), "EN-RPI\x0\x0\x0\x0\x0\x0"+ENIntervalNumber(j))
```

Here, `j` is the current time in seconds since 1970-01-01 and the `ENIntervalNumber` is just the current epoch. It is worth emphasizing that, although this identifier is referred to often as anonymous, this is not true. This value is really a pseudonymous identifier and has different protections and obligations associated with it under the GDPR than non-personally identifying information (PII) data; see [22]. Also notable is that the BLE MAC address changes more than every 10 but less than 20 minutes, and thus the RPI (and the AEM, see later) also only changes on average every 15

minutes despite the epoch being 10 minutes. This mechanism does not allow for chaining MAC addresses and RPIs to track devices directly, but a timing side channel might still provide a limited chance to do so.

2.1.3 Metadata

The beacons sent out by the protocol also contain some metadata (which is an important deviation from the original DP3T protocol – see below for concerns). This metadata is encrypted with the Associated Encrypted Metadata Key (AEMK), which is derived from the current TEK as follows:

```
AEMK(i) = HDKF(TEK[i], NULL, "EN-AEMK", 16)
```

The metadata is only 4 bytes in length and is currently specified as follows:

```
Byte 0 – Versioning. - Bits 7:6 – Major version (01).
      - Bits 5:4 – Minor version (00).
      - Bits 3:0 – Reserved for future use.
Byte 1 – Transmit power level. - This is the measured radiated
      transmit power of Bluetooth Advertisement packets,
      and is used to improve distance approximation. The
      range of this field shall be -127 to +127 dBm.
Byte 2-3 – Reserved for future use.
```

The value of the Transmit power level is a device-dependant value [20]; values for Android phones are published by Google [18]. Some of these values are unique to a specific device, as in the case of the Pixel 3a, SM-A510M, SM-G610F, and SM-J510F. This means that diagnosed victims with these devices are easier to deanonymize, if they are known to be in possession of such a device. This attack can be extended to a probabilistic one when combined with statistical information on which of the aforementioned devices are in common use in a certain location.

The four bytes of metadata are encrypted like this:

```
AEM(i,j)=aes128ctr(AEMK(i),RPI[i][j],Metadata)
```

Here, `AEMK(i)` is used as the key, `RPI(i,j)` is used as the initialization vector (IV), and Metadata as the plaintext input to an AES128 in Counter Mode. Counter Mode acts as a stream cipher in this construction and only uses the first 4 bytes of its output, and thus no padding is needed. However AES in Counter Mode does not provide authenticity of the encrypted content. This means that a re(p)laying attacker can flip bits in a controlled way [23] and could possibly downgrade the version in future. Worse is that the TX power value is not uniform. It can be known for transmitting devices and can thus be manipulated through bit-flipping to make re(p)layed beacons appear closer than in reality. Even if the sending device is unknown, there is still a ~65% chance (by flipping bits 2 and 3) to make the received beacons appear closer than they really are – while the manipulated TX power value will flip to a value that is valid for some other device. In case the attacker does not care about using illegal values it is also possible to flip bit 6 of the TX power and thus generate a value that is not associated with any device, but which will with very high probability register as a very

close contact. An attacker can also send multiple attempts flipping various bits; the validation during checking might weed out the invalid ones, and keep the valid but stronger values.

2.1.4 Transmission and Reception

The beacon that is being transmitted regularly (every 200-270 milliseconds) during the protocol is a concatenation of the 16 bits of the `RPI(i,j)` and the 4 bytes of the `AEM(i,j)` with a few extra fixed bytes providing a header to the whole BLE advertisement packet.

At the same time protocol participants also listen continuously for beacons – at least every 5 minutes, but opportunistically at any Bluetooth wake-up. Any received beacon must be stored together with a timestamp and the Received Signal Strength Indicator (RSSI) in a local database for later. On smartphones this database is part of the operating system and not accessible by apps or third party components unless the phone has been rooted.

2.1.5 Reporting

In case of positive diagnosis, the user receives an authentication key to voluntarily publish a list of `{TEK[i], i}` pairs for the period of contagion – usually 14 days, one for each day. These published keys are called Diagnosis Keys and are uploaded to the backend server. This day-level granularity creates a limitation: if a user wishes to redact some values, for example in the case of an extramarital sleepover, this probably means redacting two consecutive days, limiting the usefulness of this protocol. While it must be stressed that it is a good thing that not only the use of apps but also the upload of Diagnosis Keys is voluntary, and that there should not be any discrimination, obligation or pressure on anyone to act against their will in this regard, this voluntarism does lead to two possible issues: first, that users can participate in this system while refusing entirely to upload their Diagnosis Keys for whatever reason, thus acting as a kind of parasite benefiting from other users' disclosures, but not contributing their own, while at the same time being able to demonstrate (virtue-signalling) that they do use the app. Second, that it creates an opportunity for diagnosed miscreants to sell their authorization to upload their Diagnosis Keys to others, who then can use their own TEKs – that they previously used to tag victims – as Diagnosis Keys. The economics of this as a market seem quite lucrative, and in combination with the parasitic behavior might corrupt the entire system significantly.

2.1.6 Contact Tracing

Protocol participants regularly contact the backend server and download any new Diagnosis Keys. Using the published Diagnosis Keys it is possible to recalculate the relevant RPIs stored in the local database and check if any of them have been seen. If so, the AEM part can be decrypted, this decrypted metadata must somehow be validated since there is

no authenticity ensured (see above) and a risk analysis can be calculated based on the RSSI and the TX power value. Based on the risk, the user can then be warned regarding the contact that occurred with a diagnosed user.

2.2 GAEN Attacks

The GAEN system is being studied as part of independent research and Covid19 notification app development efforts from different countries. This has already revealed some issues: [1][13][15]. In this chapter we want to highlight results from this external and own research. The number of resources on this topic is very large and we do not claim exhaustiveness. For details please refer to the cited sources.

On the protocol level, there are two major attack classes: False Alert Injections and Tracking attacks. False Alert Injection attacks attempt to trigger infection alerts for individuals or groups of users. Tracking attacks attempt to break the anonymity or confidentiality properties of the protocol, such as deanonymizing users or uncovering parts of a social network or proving that two or more individuals met at a certain time or were present at a specific location.

While some of these protocol attacks can potentially be mitigated or weakened, it is important to note that these attacks are partly inherent to the technology. A notable example here is that the GAEN protocol is Bluetooth-based, and thus permanently opens up an attack surface for remote code execution vulnerabilities like Bluefrag (CVE-2020-0022).

An detailed description of GAEN issues going beyond this document can be found here [1] and as part of this description of a the DP-3T system [12].

2.2.1 Implementation Level Issues, Closed-Source Vendor-Controlled Basis

Attacks against implementations provide the usual vulnerability classes, such as remote (Bluetooth, internet) code execution, information leaks (like devices contacting specific servers, communicating specific amounts of data with the backends, or leaking the above-mentioned distinguisher between Apple and Google devices), and other bugs and backdoors that are expensive to discover due to the (partly) closed nature of the implementation.

Most of the crypto framework and key-handling of the Covid19 notification app is based on the assumptions of the GAEN system. However these assumptions can only partially be independently verified. A complete audit that could reveal potential additional privacy and security issues is not possible, because the complete source code is not available to the public (for example Google have not published the code handling the reception of beacons).

Verification of some features *is* possible [2], and this class of implementation-level attacks can be researched through code audits for the parts that are available and reverse-engineering for the unpublished parts, although at a significantly higher cost than if the source was available.

It is worth noting that – unless the phone is rooted – not only the implementation but also all collected data is under the sole control of Google/Apple.

Implementing this at the OS level instead of in an app means that the capabilities will be always subject to the whim of its gatekeepers, and not under the control of users who just uninstall the app if they do not want or need this functionality any more. The normalisation of this feature could lead to a slippery slope where other uses could also be permitted by the vendors. Additionally, the rule-of-law being a fragile construct in our time, legal procedures might force the vendors to permit access to this information to aid in investigations. In other words: implementing this as an OS feature "creates a dormant mass-surveillance infrastructure that works world-wide on all modern smartphones." [24]

2.2.2 False Alert Injection Attacks

The most important types of injection attacks are:

- Backend Impersonation
- False Report
- Replay
- Relay

The sub-sections below list some notable examples.

2.2.2.1 False positive contact attacks by re(p)laying or manipulating beacons

It is possible to relay beacons, basically recording them and replaying them at a different locations. This allows attackers to harvest beacons at interesting locations, such as e.g. Covid19 testing centers, and replay them next to their targets. If the original beacon is then later confirmed as an infected person, the target would then be wrongfully informed to have been in contact. [1][3][8]

This attack is further aided by the fact that GAEN's Associated Encrypted Metadata (AEM), which is used for distance calculation, is not authenticated and the time window of beacon validity being ±2 hours. Manipulating the AEM through bitflips can allow an attacker to suggest less distance to the sender than was really the case. The authors of [3] further argue that these attacks can also be carried out by single-person entities who are not physically present, such as malicious app developers.

Relay attacks can further be improved by using non-standard signal amplifiers and high-gain directional antennas when sending out the relayed beacons.

2.2.2.2 Impersonating the backend

Depending on the implementation used by the entity employing the GAEN API, it might be possible to impersonate the server. This could be used for example to distribute keys belonging to beacons that were intentionally brought in contact with many devices, or with specific victim devices.

this could also be used as part of a Denial of Service attack preventing the upload or selectively preventing the upload of valid keys.

2.2.2.3 Access to be able to upload keys

The difficulty of uploading keys depends on the implementation used by the entity employing the GAEN API. Uploading might be more or less restricted.

In systems where key upload is completely open, people might upload keys even though they were not infected, which would lead to people erroneously believing they have been exposed to the virus. Depending on the follow-up steps of a possible contact, this could have a signification impact on people's lives.

If these erroneous uploads happen often enough, they could render the entire system useless.

Key upload opportunities might also be valuable, depending on the follow-up reaction. An attacker might transmit their own beacons in close proximity to a victim and then report themselves as infected. In implementations with restricted key upload this might create an incentive to sell upload access opportunities to another party.

2.2.3 Tracking Attacks

The most important types of tracking attacks are:

- Deanonymizing Known Reported Users
- Location Confirmation: Proving presence at specific locations
- Contact confirmation: Proving contact between two or more persons at a specific time.
- Triggering side-channels to disclose (partial) social networks of persons of interest
- Coercion

The following sub-sections list a number of notable examples.

2.2.3.1 Installation and updates require non-anonymous accounts

A notable implementation-level tracking issue is that the Google Play store requires a Gmail account, which is associated with some personal data, but most importantly also with a phone number and IMEI. Installation of the app is

therefore not anonymous. Circumvention of this intentional account-bound flow might only be possible for highly skilled users; the vast majority of users will not be able to do so.

Apple devices similarly require an account associated with personal data. Apple's ecosystem is even more tightly controlled, not allowing installation of apps from alternative sources. While apps from other sources may be installed using jailbreak exploits, this is in violation with Apple's terms of use. [21]

Stolen, seized and decommissioned phones likely contain this identification data. Malicious actors can then use this data in conjunction with extracted contact tracking data.

2.2.3.2 De-anonymization attack of infected persons

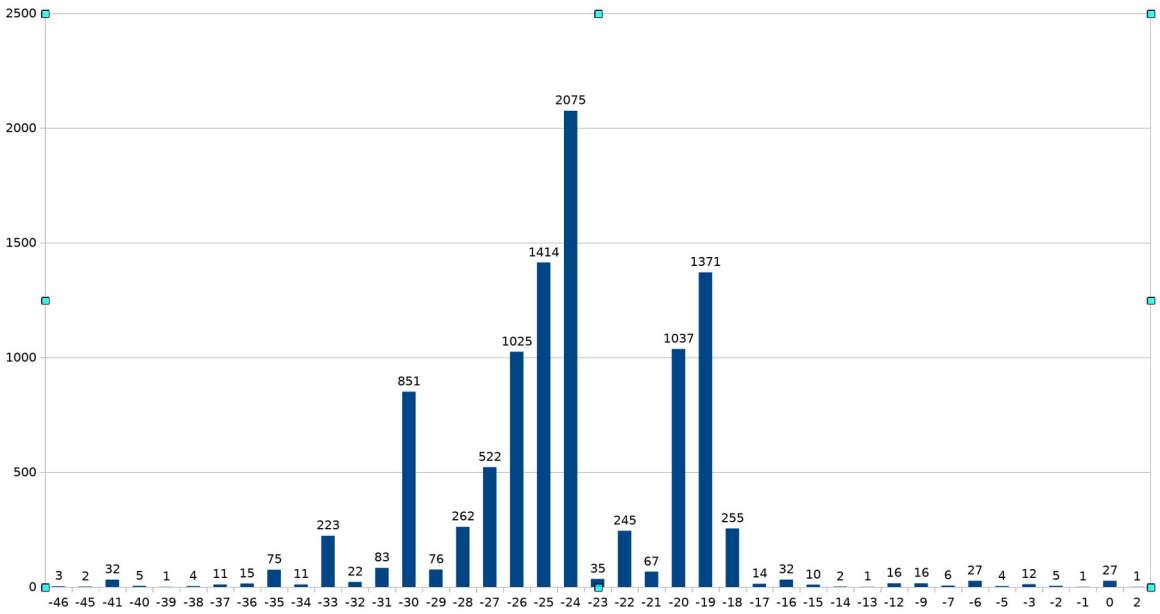
TEKs of infected person are voluntarily uploaded and distributed. The data of a network recording bluetooth signals in an area over a stretch of time can afterwards be used to track and de-anonymize infected users. This data could also be coupled with additional data sources such as CCTV [1][3] or device distinguishers like the TX power value contained in the metadata. Researchers have simulated this in a Proof-of-Concept to show the reach of this method [4]. Building tracking devices for these beacons is cheap and readily available even to hobbyists.

This is especially concerning as Bluetooth measurement networks already exist. Schiphol Airport [] and some NS stations [] do exactly this, according to news reports and their own statements. Other existing apps and technologies likewise are using Bluetooth in such a way that their data could be reused to track these infected persons: [5][3][6][7].

Among others, attackers can be geeks who have the necessary technology, vigilantes trying to harass infected people or paparazzi wanting to make a quick profit by selling information about diagnosed celebrities to the tabloids.

2.2.3.3 Possible identification of sending device type trough TX Power Value in the Metadata

The GAEN system sets the TX power based on the device and includes this information in the AEM metadata [9][10]. By combining beacons of infected persons in conjunction with an infected person's TEK it is possible to infer which device these people were using.



TX power distribution of Android devices

In the bar graph above, the X-axis shows the value of TX power sent, and in the Y-axis shows how many different devices set this value. Devices with low values on the Y-axis are TX values that should be easy to identify. Notably, the Pixel 3a, SM-A510M, SM-G610F and SM-J510F all have unique values. If these values are seen in the Metadata, the sending device can be unambiguously identified.

2.2.3.4 Contact data is stored on the phone

Phones that use contact tracing apps, e.g. those based on the GAEN framework, save the beacons for some amount of time. This information could be re-used by law enforcement, but also by malicious actors.

A user could be forced to reveal their TEKs and recorded beacons, which can be done by means of root exploits or jailbreaks on a seized or stolen phone. This data could be used to establish whether there was contact with a beacon signal, another beacon sending device or a beacon collecting device. While it is the intention that data would only be shared voluntarily by infected persons, we can imagine scenarios where people would be forced to do so, even if such coercion is illegal.

Some conceivable scenarios would be:

- A murder investigation in which law enforcement checks if and for how long there was contact between a victim and a suspect.
- An investigation to determine whether a political dissident was close to a beacon placed on a demonstration or had contact with other dissidents whose phones were also seized.

3 Methodology

3.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2017) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

3.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

4 Reconnaissance and Fingerprinting

We were able to gain information about the software through the following automated scans. Any relevant scan output will be referred to in the findings.

- Mobile Security Framework (MobSF), an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework – https://mobsf.github.io/Mobile-Security-Framework-MobSF/

5 Findings

We have identified the following issues:

5.1 IMM-006 — (Android) The Build Is Not Reproducible

Vulnerability ID: IMM-006

Vulnerability type: Build Reproducibility

Threat level: Moderate

Description:

Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code.

Technical description:

A build is reproducible if given the same source code, build environment and build instructions, any party can recreate bit-for-bit identical copies of all specified artifacts. While the developers of the Immuni application are aware of the necessity of reproducibility in this context, this process was not fully implemented at the time of the audit.

The relevant attributes of the build environment, the build instructions and the source code as well as the expected reproducible artifacts are defined by the authors or distributors. The artifacts of a build are the parts of the build results that are the desired primary output.

The build process of the Immuni app is documented at https://github.com/immuni-app/immuni-app-android/blob/development/README.md, and mentions:

When it comes to reproducible builds, we will instead open an issue explaining what we have done so far and any missing steps.

No corresponding issue was found in the tracker at the time of the audit.

The auditors built an APK matching the version tested according to the Git repository (tag `Immuni-1.2.0build1203309`) but could not compare it efficiently to the corresponding APK, as installed on the mobile phone through Google's Play store.

This was notably attempted with help from the `apkdiff.py` script from the Telegram project, as found at https://github.com/DrKLO/Telegram/blob/master/apkdiff.py:

```
$ grep -F -A 1 release.kotlin_module base.apk/META-INF/MANIFEST.MF | head -3
Name: META-INF/Immuni-1.2.0build1203309_release.kotlin_module
SHA-256-Digest: tkLXfoscYykZ1piV+CwOkFszwaqkWk0alWFD6pjglP0=
--
```

```
$ git show Immuni-1.2.0build1203309
commit 715fee70006cd80ef38aec55d4023680ce6eba83
Author: Marco Uberti <marcouberti84@gmail.com>
Date:   Thu Jun 18 17:26:10 2020 +0200

    #build-release
$ git status
HEAD detached at 715fee7
$ python apkdiff.py base.apk app/release/Immuni-1.2.0build1200000-release.apk
APKs has different amount of files (982 != 1501)
APKs are different!
```

The release tag identified above also matched the information available in `README.md`.

For more information on Reproducible Builds, see also https://reproducible-builds.org/.

Impact:

The general public does not have a tool to verify that the applications they install are provably built using their published source code.

Recommendation:

- Complete the efforts initiated towards a completely reproducible build. (Insofar as mobile applications can be)

5.2 IMM-001 — (Android/iOS) Missing Jailbreak/Root detection checks

Vulnerability ID: IMM-001

Vulnerability type: Missing Jailbreak Checks

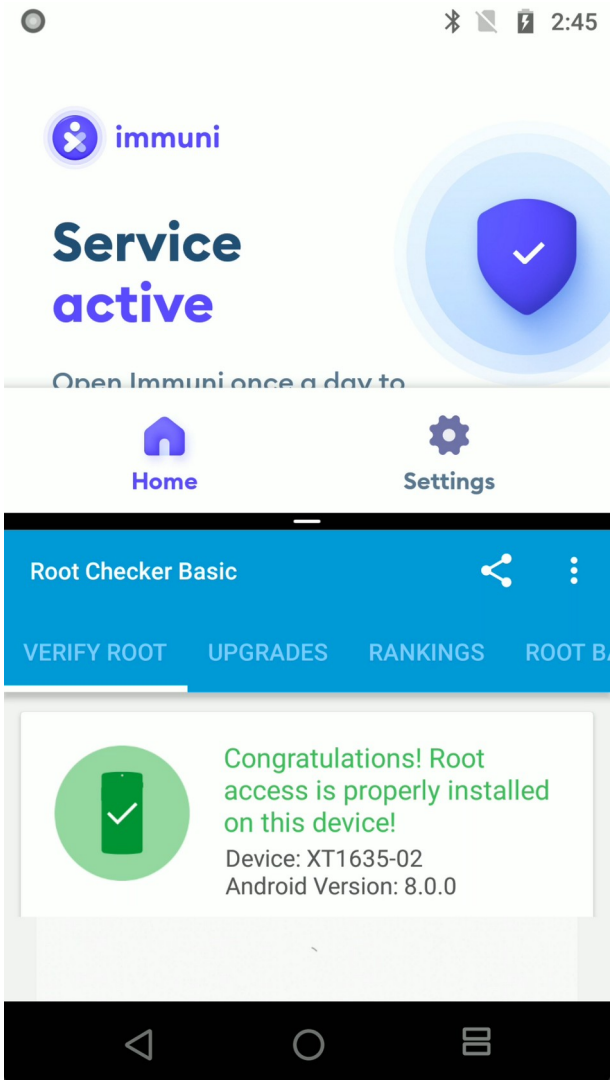
Threat level: Low

Description:

The apps (iOS and Android) can be installed and run on a jailbroken/rooted device.

Technical description:

A rooting/jailbreaking process compromises the security of the host android/iOS device. The integrity of the operating system's controls over data that an application can access is also lost. On a compromised device, a user or malicious app can disable critical security features and may compromise the integrity of the data of all applications. The Immuni app has no security controls to check whether it is running on a compromised device.



Impact:

When an application runs on a compromised device, the operating system and application's security controls, such as sandboxing, keychain access, etc. might fail to prevent the user data from being accessed by other, malicious apps on the device. An application running on a compromised device also runs higher risks of being exposed to malware, memory dumps, tampering, and other types of malicious activity that could possibly expose the user's credentials or other sensitive data. For example, in the Immuni application it is possible for an attacker to bypass internal security controls like `uploadDisabler` through runtime instrumentation.

Recommendation:

Implement a root/jailbreak detection mechanism on both Android & iOS apps. Although it is not strictly necessary to prevent the application from running on a compromised device, users should be informed of the increased risk of doing so. To attempt to detect if the application is running on a rooted/jailbroken device, use checks such as the following:

- Check for the presence of certain paths, for example:
  - `/Applications/Cydia.app`
  - `/etc/apt`
  - `/Library/MobileSubstrate/MobileSubstrate.dylib`
  - `/usr/sbin/sshd`
  - `/var/cache/apt`

- Check if the `cydia://` URL handler is in use.
- Test for write access in the `/private` directory.
- Check for root access on android devices.
- Check for `su` binary inside android.
- Check if SuperUser apk is installed.

Root/Jailbreak detection is a security control that can help applications to be made secure against running on compromised devices. However, like most other security controls, it can be bypassed using several techniques on the attacker's device.

6 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

6.1 NF-000 — (Android) Application Permissions Review

The MobSF static analysis report for the Android application listed the following permissions as required:

Permission	Status	Info	Description
android.permission.INTERNET	dangerous	full Internet access	Allows an application to create network sockets.
android.permission.BLUETOOTH	dangerous	create Bluetooth connections	Allows an application to view configuration of the local Bluetooth phone and to make and accept connections with paired devices.
android.permission.REORDER_TASKS	dangerous	reorder applications running	Allows an application to move tasks to the foreground and background. Malicious applications can force themselves to the front without your control.
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.
android.permission.WAKE_LOCK	dangerous	prevent phone from sleeping	Allows an application to prevent the phone from going to sleep.
android.permission.RECEIVE_BOOT_COMPLETED	normal	automatically start at boot	Allows an application to start itself as soon as the system has finished booting. This can make it take longer to start the phone and allow the application to slow down the overall phone by always running.
android.permission.FOREGROUND_SERVICE	normal		Allows a regular application to use Service.startForeground

While some combinations of these permissions could potentially allow abuse, they all seemed relevant and necessary for the legitimate purpose of the application.

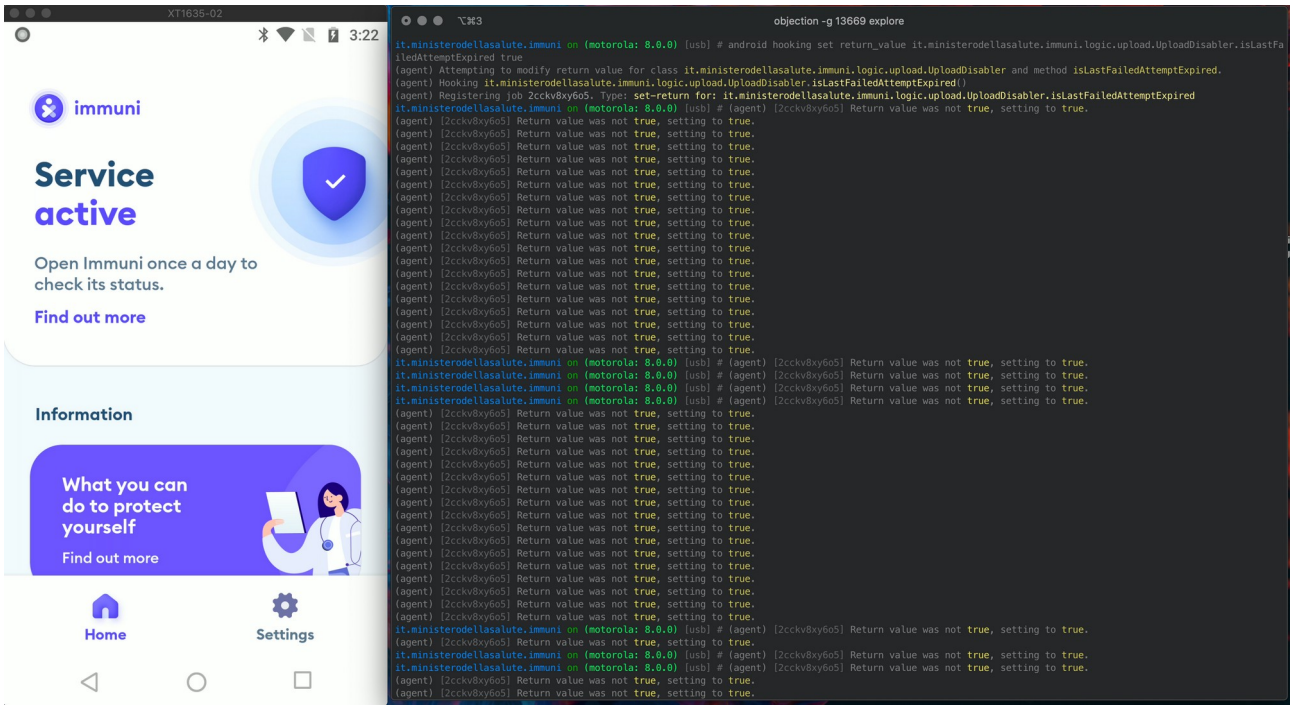
6.2 NF-002 — (iOS) Cache Database Being Created

A cache database (`Cache.db`) is created by the underlying HTTP calls `NSURLRequest.CachePolicy.useProtocolCachePolicy`. The data stored inside this database turned out not to be sensitive, however.

6.3 NF-003 — (Android/iOS) Bypassing the client side control UploadDisabler

It is possible to bypass the client-side security control of `UploadDisabler` on both Android and iOS. This would disable the timer, which disables the upload functionality in the application. This issue does however not directly impact the security of the application, as the authorisation of upload functionality happens through an OTP, which is exchanged with a healthcare operator.

Bypassing the timer can be achieved by setting the return value of `it.ministerodellasalute.immuni.logic.upload.UploadDisabler.isLastFailedAttemptExpired` to **true**.



7 Future Work

- **Audit the complete app**
  This audit was subject to severe scope limitations, dramatically reducing opportunities for finding actual or potential vulnerabilities. Specifically, the upload of TEKs requires a thorough test and analysis. A deeper test would help build greater public trust in the app and in turn, provide a greater payoff in controlling the spread of COVID-19.
- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

8 Conclusion

We discovered 1 Moderate issue and 1 Low-severity issue during this penetration test.

The development team is already aware of the moderate issue, and we were informed that they were working on resolving it in the near future.

The resolution of the low-severity issue is not critical; the missing check was a conscious decision by the development team as a trade-off to allow power users to run the application legitimately without getting in their way.

There are two reasons that this list of findings is short. Firstly, to their credit, the developers have clearly built the app with security in mind, and have followed secure (and documented) development practices. Secondly, and rather less satisfactorily, it also reflects that our ability to test the app thoroughly was hampered by scope restrictions which undermined the effectiveness of the audit. We strongly recommend a deeper test without these scope restrictions.

We do want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Radically Open Security is the world's first not-for-profit computer security consultancy. We operate under an innovative new business model whereby we use a Dutch fiscal entity, called a “Fiscaal Fondswervende Instelling” (Fiscal Fund raising Institution), as a commercial front-end to send 90% of our profits, tax-free, to a not-for-profit foundation, Stichting NLnet. The NLnet Foundation has supported open-source, digital rights, and Internet research for almost 20 years.

In contrast to other organizations, our profits do not benefit shareholders, investors, or founders. Our profits benefit society. As an organization without a profit-motive, we recruit top-name, ethical security experts and find like-minded customers that want to use their IT security budget as a "vote" to support socially responsible entrepreneurship. The rapid pace of our current growth reflects the positive response the market has to our idealistic philosophy and innovative business model.

If you have any questions about the advice in this report, please contact us at info@radicallyopensecurity.com

For more information about Radically Open Security and its services please visit our website: www.radicallyopensecurity.com.

9 Bibliography

[1] SECURITY ANALYSIS OF COVID-19 CONTACT TRACING SPECIFICATIONS. https://eprint.iacr.org/2020/428.pdf.

[2] GAEN Due Diligence: Verifying The Google/AppleCovid Exposure Notification API. https://www.scss.tcd.ie/Doug.Leith/pubs/gaen_verification.pdf.

[3] SwissCovid: a critical analysis of risk assessment by Swiss authorities. https://arxiv.org/abs/2006.10719. Accessed: 2020-07-18.

[4] BLE contact tracing sniffer PoC. https://github.com/oseiskar/corona-sniffer. Accessed: 2020-07-18.

[5] Examples of Beacon tracking. https://github.com/DP-3T/documents/issues/43. Accessed: 2020-07-18.

[6] bitsaboutme data selling app with contact tracker contact tracing sniffer PoC. https://bitsabout.me/en/ . Accessed: 2020-07-18.

[7] New York Times: As you shop, “beacons” are watching you, using hidden technology in your phone.. https://www.nytimes.com/interactive/2019/06/14/opinion/bluetooth-wireless-tracking-privacy.html. Accessed: 2020-07-18.

[8] The Dark Side of SwissCovid. https://lasec.epfl.ch/people/vaudenay/swisscovid.html. Accessed: 2020-07-18.

[9] Exposure Notifications BLE attenuations. https://developers.google.com/android/exposure-notifications/ble-attenuation-overview. Accessed: 2020-07-18.

[10] Exposure Notification Bluetooth Specification. https://blog.google/documents/70/Exposure_Notification_-_Bluetooth_Specification_v1.2.2.pdf. Accessed: 2020-07-18.

[11] CWA Datenschutz Folgeabschätzung. https://www.coronawarn.app/assets/documents/cwa-datenschutz-folgenabschaetzung.pdf. Accessed: 2020-07-18.

[12] DP3T White Paper. https://github.com/DP-3T/documents/raw/master/DP3T%20White%20Paper.pdf. Accessed: 2020-07-18.

[13] Analysis of DP3T. https://eprint.iacr.org/2020/399. Accessed: 2020-07-18.

[14] Response to "Analysis of DP3T". https://github.com/DP-3T/documents/raw/master/Security%20analysis/Response%20to%20'Analysis%20of%20DP3T'.pdf. Accessed: 2020-07-18.

[15] Centralized or Decentralized? The Contact Tracing Dilemma. https://eprint.iacr.org/2020/531. Accessed: 2020-07-18.

[16] DP3T – Best Practices for Operation Security in Proximity Tracing. https://github.com/DP-3T/documents/raw/master/DP3T%20-%20Best%20Practices%20for%20Operation%20Security%20in%20Proximity%20Tracing.pdf. Accessed: 2020-07-18.

[17] DP3T – Data Protection and Security. https://github.com/DP-3T/documents/raw/master/DP3T%20-%20Data%20Protection%20and%20Security.pdf. Accessed: 2020-07-18.

[18] . https://developers.google.com/android/exposure-notifications/files/en-calibration-2020-06-13.csv. Accessed: 2020-07-18.

[19] . https://www.schiphol.nl/en/page/data-processing-crowd-management/. Accessed: 2020-07-18.

[20] . https://developers.google.com/android/exposure-notifications/ble-attenuation-computation. Accessed: 2020-07-18.

[21] . https://support.apple.com/en-gb/HT201954. Accessed: 2020-07-18.

[22] . https://eur-lex.europa.eu/legal-content/EN/TXT/?qid=1527155581826&uri=CELEX:32016R0679. Accessed: 2020-07-18.

[23] Analysis of SwissCovid. https://lasec.epfl.ch/people/vaudenay/swisscovid/swisscovid-ana.pdf. Accessed: 2020-07-20.

[24] Google Apple Contact Tracing (GACT): a wolf in sheep's clothes.. https://blog.xot.nl/2020/04/19/google-apple-contact-tracing-gact-a-wolf-in-sheeps-clothes/. Accessed: 2020-07-20.

Appendix 1 Testing team

Abhinav Mishra	When he hacked his first application while doing his engineering degree back in 2009, Abhinav thought "this is cool". Since then Abhinav has been involved in hacking web & mobile apps and networks as a penetration tester. He has won numerous accolades from multiple organizations (including Yahoo, Coinbase, Rapid7, Jet, Mobidea, etc) for responsible disclosure of vulnerabilities . He is a member of Synack Red Team and Cobalt core team, and has performed 300+ web, 100+ mobile applications and numerous network penetration tests.
Pierre Pronchery	Pierre Pronchery is a Senior IT-Security Consultant and an accomplished Software Engineer. Freelancing for over a decade, he could be found auditing major companies in the Telecommunications, Finance, and Retail industries, or supporting Open Source Software and Hardware communities. He is currently serving as Vice-President for the NetBSD Foundation, and the founder and CEO of Defora Networks GmbH in Germany.
Stefan Marsiske	Stefan runs workshops on radare2, embedded hardware, lock-picking, soldering, gnuradio/SDR, reverse-engineering, and crypto topics. In 2015 he scored in the top 10 of the Conference on Cryptographic Hardware and Embedded Systems Challenge. He has run training courses on OPSEC for journalists and NGOs.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.