Radically Open Security

Cryptographic Framework and Back-end Security Evaluation

Ministerie van Volksgezondheid, Welzijn en Sport

Dutch COVID-19 Notification App

V 1.1
Diemen, August 28th, 2020
Confidential

Document Properties

Client	Ministerie van Volksgezondheid, Welzijn en Sport
Title	Cryptographic Framework and Back-end Security Evaluation
Targets	The cryptographic framework of the Dutch COVID-19 notification app The code of the Dutch COVID-19 notification app back-end
Version	1.1
Pentesters	Tim Hummel, Tillmann Weidinger, Stefan Marsiske
Authors	Tim Hummel, Stefan Marsiske, Patricia Piolon, Marcus Bointon, Melanie Rieback
Reviewed by	Marcus Bointon, Patricia Piolon, Melanie Rieback
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	July 15th, 2020	Tim Hummel	Initial draft
0.2	July 17th, 2020	Stefan Marsiske	Added generic GAEN section
0.3	July 21st, 2020	Tim Hummel	Conclude initial crypto framework evaluation
0.4	July 23rd, 2020	Patricia Piolon	Review
0.5	August 5th, 2020	Tim Hummel	New Dutch crypto framework documents
0.6	August 11th, 2020	Tim Hummel	Initial back-end evaluation
0.7	August 14th, 2020	Tim Hummel	Continued back-end evaluation
0.8	August 19th, 2020	Patricia Piolon	Review
0.9	August 19th, 2020	Tim Hummel	New updates
0.10	August 20th, 2020	Marcus Bointon	Review
0.11	August 20th, 2020	Patricia Piolon	Review
0.12	August 21st, 2020	Marcus Bointon, Patricia Piolon, Tim Hummel	Expected final Review
0.13	August 26th, 2020	Tim Hummel	Additional code and documentation updates
1.0	August 28th, 2020	Marcus Bointon, Patricia Piolon, Melanie Rieback	Review

1.1	August 28th, 2020	Patricia Piolon	Fixed status error

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

Radically Open Security B.V. is registered at the trade register of the Dutch Chamber of Commerce under number 60628081.

Table of Contents

1 Executive Summary

1.1 Introduction

Radically Open Security has been contracted by the Ministry of Health, Welfare and Sport (Ministerie van Volksgezondheid, Welzijn en Sport) to peer-review the Dutch cryptographic framework and perform a code review on the CoronaMelder back-end code. Both reviews were time-boxed.

This report contains our analysis as well as detailed explanations of any findings discovered.

1.2 Scope of Work

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

The cryptographic framework of the Dutch COVID-19 notification app, specifically the "Covid-19 Exposure Notification Cryptografie raamwerk" document retrieved from [30] (hereafter called 'the Dutch Crypto Framework document').

The code of the Dutch COVID-19 notification app back-end, retrieved from [32]

1.3 Project Objectives

There were two objectives for this evaluation:

Peer review of the cryptographic basis for the Dutch COVID-19 notification app. We checked if the claims made in the Dutch Crypto Framework document can be supported by the document itself and the supporting documentation. We point out potential gaps.

Code review of the Dutch COVID-19 notification app back-end. We carried out a code and documentation review to verify the implementation against the design. We point out relevant security differences, potential back doors, and any other security findings.

1.4 Timeline

The security audit took place between July 9th and August 25th, 2020.

Both activities were initially time-boxed to a combined 16 days including reporting. We were allocated an extension of maximum 12.5 days, which we only partially used. This was needed because we were required to spend a significant amount of time accommodating the ongoing documentation and code changes occurring during our review.

1.5 Results In A Nutshell

We began our audit with a peer-review of the Dutch Crypto Framework document. It builds upon the Google/Apple Exposure Notification (GAEN) system, which has well established attack vectors and risks, some of which are inherent to the underlying technology. These have not been fully addressed by Google or Apple - and to the best of our knowledge, GAEN's security has never been fully audited or verified by an independent party.

The Dutch Crypto Framework document is a work in progress, and there are several shortcomings and points that are not (yet) defined. A clear choice has been made for a key upload process that, in theory, allows the health authority to link uploaded diagnosis keys to other personal data. This is a trade-off in order to prevent non-infected people from uploading their keys maliciously. The recently added international key exchange documentation also contains additional privacy trade-offs, leaving some open questions.

During the back-end code review we found 1 High, 1 Elevated, 11 Moderate, and 13 Low-severity findings.

Of these, 1 High, 1 Elevated, 6 Moderate, and 7 Low-severity findings remain unresolved in the latest evaluated version.

The most severe issues are:

- Usage of a third-party authentication provider, which if compromised or acting maliciously, can give attackers the role of health authority employees, able to authorize the upload of diagnosis keys. We recommend checking that this party is regularly pentested, audited, or verified to be trustworthy, or otherwise to not use a third party for authentication at all.
- A feature that is not part of the Dutch Crypto Framework document requirements allows health authority employees who just authorized a key upload to check if the de-anonymized user actually uploaded their keys. Key upload is supposed to be voluntary. Knowledge of whether a user did could be used legitimately for feedback, or could be abused to e.g. coerce users and monitor which users (identifiable by name and phone number) uploaded their keys to help de-anonymize the keys.
- Decoy keys could be distinguished from real keys by their properties. This finding has been partially resolved, and now it only allows estimating the total number of uploaded valid keys.

This evaluation is a code review only. It's essential to perform a penetration test on the deployed system to provide additional assurance on the security of the system as a whole, but this is beyond the scope of a code review.

The table below gives an overview of the code review findings that were NOT resolved in the newest evaluated version:

ID	Type	Description	Threat level
NLC-019	Authentication issue	The solution uses The Identity Hub (https://theidentityhub.com) as its OAuth provider to allow users access to the healthcare API.	High
NLC-013	Privacy issue	An API allows health authority employees who have authorized a key upload to check if the user really did upload the keys. This reduces anonymity and could be abused to coerce users.	Elevated

NLC-007	Privacy issue	The decoy traffic analysis is not in line with the design and does not seem to be able to successfully mimic real traffic.	Moderate
NLC-014	Authentication issue	The JWT for authentication against the health authority back-end is part of a GET parameter.	Moderate
NLC-020	Authentication issue	The JWT used for authentication is valid for several hours. There is no rate limiting nor any way to revoke individual authentications.	Moderate
NLC-022	HSTS	A HTTP Strict-Transport-Security response header (HSTS) is recommended for all APIs that only use SSL, in order to prevent adversaries from using SSL stripping attacks in a MitM scenario.	Moderate
NLC-028	Authentication issue	The signature validation used during diagnosis key upload contains a timing oracle.	Moderate
NLC-029	Authentication issue	The authentication against the health authority back-end contains a weak single-use component.	Moderate
NLC-001	Inconsistent documentation	The architecture and design deviates from the implementation. Inconsistent documentation can be a source of errors and confusion.	Low
NLC-008	Privacy issue	The database stores time values relating to diagnosis keys. In combination with other information, this can aid de-anonymization efforts.	Low
NLC-012	Authentication issue	Any valid JWT token is accepted as valid regardless of its properties. As this token needs to match the one stored in the database, this does not currently result in an issue. It could however result in confusion or issues in future updates.	Low
NLC-016	Privacy issue	The decoy keys added to increase anonymity can mostly be distinguished from real diagnosis keys by their properties.	Low
NLC-025	Privacy issue	Uploaded diagnosis keys are not deleted, only marked as published.	Low
NLC-026	Privacy issue	There is no intentional delay between the time of uploading and the time of publishing for diagnosis keys.	Low
NLC-027	Privacy issue	The Microsoft SQL database used in the background can retain data for longer than strictly required.	Low

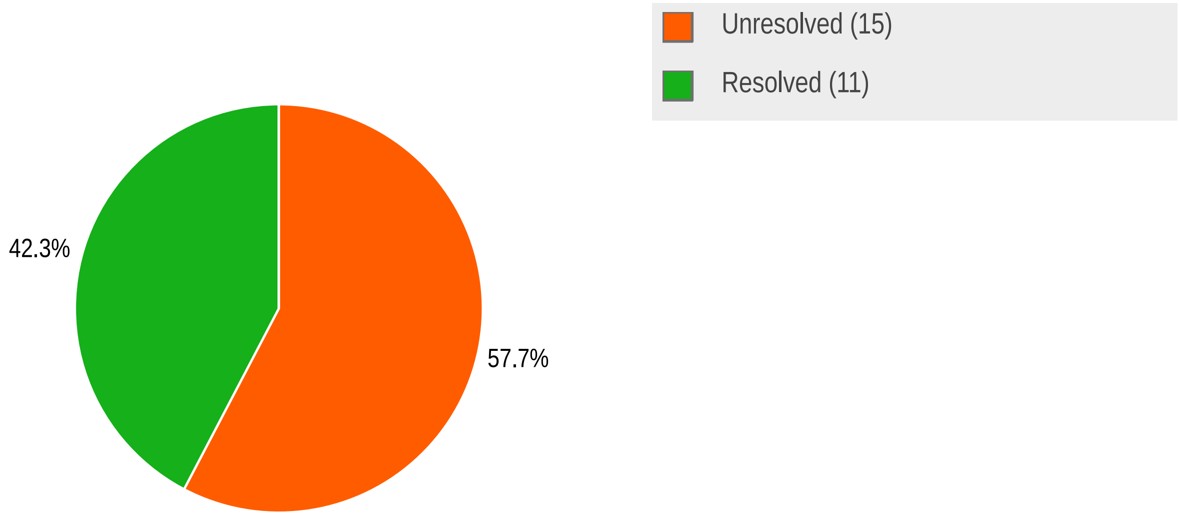
The table below gives an overview of the findings that ARE resolved in the latest evaluated version:

ID	Type	Description	Threat level
NLC-002	Misconfiguration potential	The software contains options to switch off security features for development.	Moderate
NLC-004	Data sanitization	The RollingPeriod of uploaded keys is not properly checked. Uploaded keys can continue into the following days.	Moderate

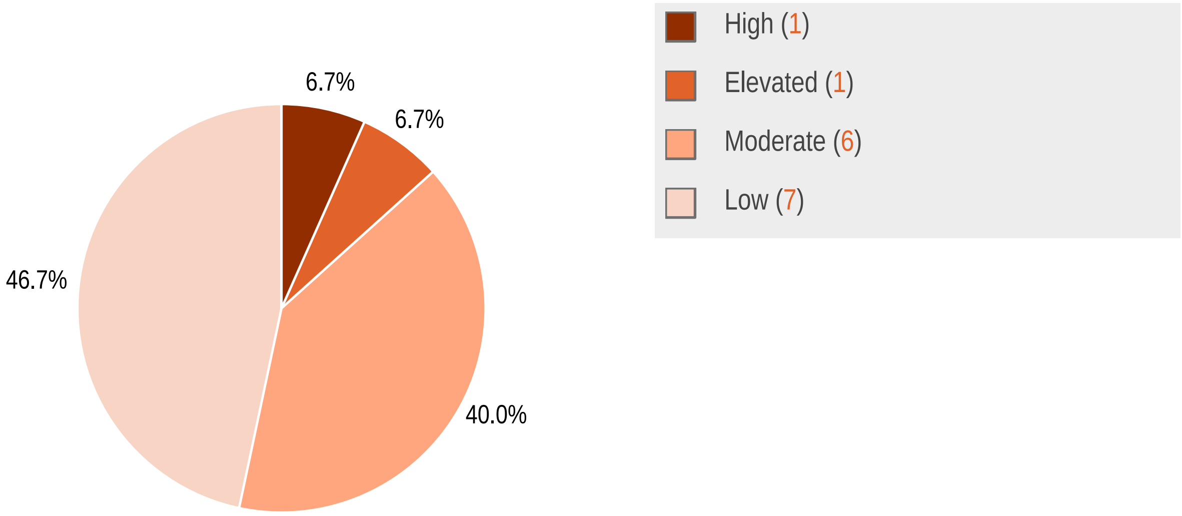
NLC-005	Data sanitization	MobileAppApi does not check if an uploaded TEK is associated with a future date.	Moderate
NLC-017	Privilege separation	Even though the databases are accessed by different programs with different needs, there is limited privilege separation.	Moderate
NLC-018	Privilege separation	Some components imply they need to run as local administrator. If they do, their compromise would occur in the context of the administrator role instead of in that of an isolated process.	Moderate
NLC-003	Privilege separation	Connection strings containing database credentials are included in the appsettings.json of components/servers that do not require them.	Low
NLC-009	Privacy issue	The log messages store information which, together with log timestamps, can aid de-anonymization efforts.	Low
NLC-010	Information disclosure	Developer exception page function is activated in production.	Low
NLC-015	Misconfiguration potential	The default configuration is to add example keys to the database.	Low
NLC-021	CSRF	The Cross-Origin Resource Sharing (CORS) policy facilitates Cross-Site Request Forgery (CSRF) attacks	Low
NLC-023	Privacy issue	The health authority employee authentication token contains the employee's email address and name, which are not used by the back-end.	Low

1.5.1 Visual Overview

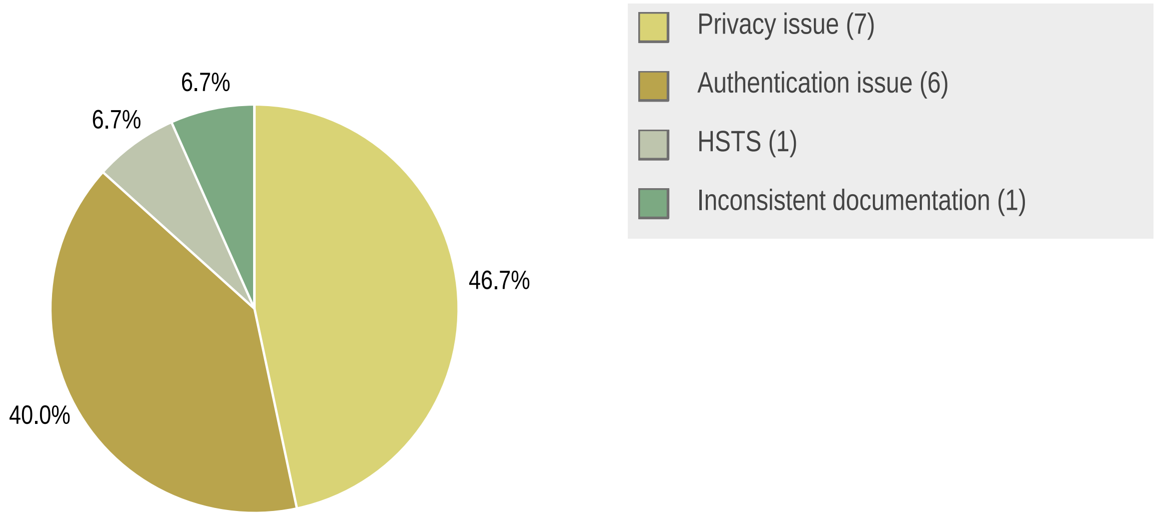
1.5.1.1 Code Review Findings by Status



1.5.1.2 Unresolved Code Review Findings by Threat Level



1.5.1.3 Unresolved Code Review Findings by Type



2 Abbreviations and Terms Used in This Document

Abbreviation/Term	Definition
AEMK	Associated Encrypted Metadata Key
AES	Advanced Encryption Standard
API	Application programming interface
BLE	Bluetooth Low Energy
CDN	Content Delivery Network
CK	ConfirmationKey
CORS	Cross-Origin Resource Sharing
CSRF	Cross-Site Request Forgery
CTR	Counter, a mode of AES
Diagnosis Key	GEAN terminology: TEK of an confirmed infected person
DP-3T	Decentralized Privacy-Preserving Proximity Tracing
GAEN	Google/Apple Exposure Notification (system)
HMAC	Hash-based Message Authentication Code
HSTS	HTTP Strict Transport Security
IV	Initialization Vector
JWT	JSON Web Token
LLC	LabConfirmationCode
RPI	Rolling Proximity Identifier
RPIK	Rolling Proximity Identifier Key
RSSI	Received Signal Strength Indicator
TEK	Temporary Exposure Key
TX	transmission

3 The GAEN Protocol Overview

The Dutch COVID-19 notification app uses the Google/Apple Exposure Notification system as its (cryptographic) basis, commonly referred to as GAEN. GAEN constitutes a large part of the overall solution and provides its security fundamentals, so we need to review GAEN before looking at any additions made by the Dutch Crypto Framework document.

In this section we first give a detailed introduction to the GAEN protocol, with 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. This section was written for GAEN version 1.4; a quick scan did not produce any notable differences compared to the current 1.6 version.

3.1 The Protocol

GAEN is a derivative of the Decentralized Privacy-Preserving Proximity Tracing (DP-3T, [13]) protocol, thus attacks [14][16] on DP-3T likely also apply to GAEN, as would their mitigations, [15][17][18].

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

Specifications for the GAEN protocol can be found on both Google and Apple web sites. They were written independently of each other and historically there have been examples of these two variants having diverging and conflicting content. One such difference 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 GAEN specification can be found at these locations (current at the time of writing):

- 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

3.1.1 The GAEN Architecture

GAEN-based solutions have the following simplified architecture:

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

The protocol itself specifies only the workings of the implementation behind the API. Everything else, like the app and the back-end 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 back-end server. The functionality of the back-end server is also very limited. This also means that the trusted computing base can be minimal, and that there is not much to audit if tha app is implemented without feature creep and without access to the protocol implementation hidden behind the API.

3.1.2 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 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 this for secure operation anyway, so 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 because the input material has 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 often referred to 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 *personal data*; see [24]. 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 using a timing side channel there might still provide a limited chance of doing so.

3.1.3 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:

```
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 [22]; values for Android phones are published by Google [19]. 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 (`ctr`). Counter mode acts as a stream cipher in this construction and only uses the first 4 bytes of its output, so 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 [28] 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.

3.1.4 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 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.

3.1.5 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 back-end 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 both the use of apps and 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 any reason, thus parasitically 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.

3.1.6 Contact Tracing

Protocol participants regularly contact the back-end server and download any new diagnosis keys. Using these 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.

3.2 GAEN Attacks

The GAEN system is being studied as part of independent research and COVID-19 notification app development efforts across different countries. This has already revealed some issues: [1][14][16]. In this chapter we want to highlight results from this external research alongside our own. The number of resources for this topic is very large and we do not claim exhaustiveness. For further details please refer to the cited sources.

At 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 to some extent these attacks are 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).

A 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 [13].

3.2.1 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 back-ends, or leaking the above-mentioned distinguisher between Apple and Google devices), and other bugs and back doors that are expensive to discover due to the (partly) closed nature of the implementation.

Most of the crypto framework and key-handling of the COVID-19 notification app is based on 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 has 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 on 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 no longer need this functionality. 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." [29]

3.2.2 False Alert Injection Attacks

The most important types of injection attacks are:

- Back-end Impersonation
- False Report
- Replay
- Relay

The sub-sections below list some notable examples.

3.2.2.1 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 location. This allows attackers to harvest beacons at interesting locations, such as COVID-19 testing centers, and replay them near their targets. If the original beacon is later confirmed as an infected person, the target would then be informed incorrectly to have been in contact with them. [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 that the time window of beacon validity may vary by ±2 hours. Manipulating the AEM through bitflips can allow an attacker to suggest closer proximity 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 be improved by using non-standard signal amplifiers and high-gain directional antennas when sending out the relayed beacons.

3.2.2.2 Impersonating the Back-end

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 uploads altogether, or by selectively preventing the upload of valid keys.

3.2.2.3 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 after a possible contact, this could have a significant 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 his own beacons in close proximity of a victim and then report himself as infected. In implementations with restricted key upload this might create an incentive to sell upload access opportunities to another party.

3.2.3 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 people at a specific time.
- Triggering side-channels to disclose (partial) social networks of people of interest
- Coercion

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

3.2.3.1 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 also require an account associated with personal data. Apple's ecosystem is even more tightly controlled, prohibiting 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 [23].

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

3.2.3.2 De-anonymization Attack of Infected People

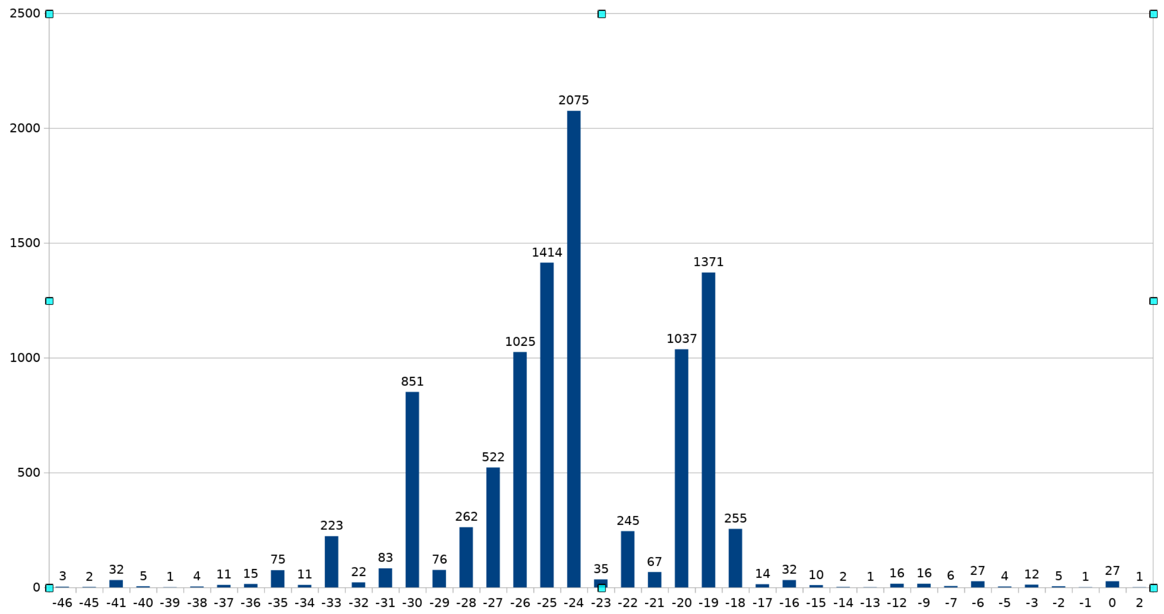
TEKs of infected person are voluntarily uploaded and distributed. The data of a network recording Bluetooth signals in an area over a period of time can afterwards be used to track and de-anonymize infected users. This data could also be coupled with additional data sources such as CCTV [1][3] or device 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]. Constructing tracking devices for these beacons is cheap and readily accessible even to hobbyists.

This is especially concerning as Bluetooth measurement networks already exist. Schiphol Airport [21] and some NS stations [20] do exactly this, according to news reports and their own statements. Other existing apps and technologies use Bluetooth in such a way that their data could be reused to track these infected people: [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.

3.2.3.3 Possible Identification of Sending Device Type Through 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 people 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.

3.2.3.4 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 people, we can image 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 at a protest, or had contact with other dissidents whose phones were also seized.

4 Dutch COVID-19 Notification App Cryptographic Framework

In this section we analyze the Dutch COVID-19 notification app's cryptographic framework, specifically the document "Covid-19 Exposure Notification Cryptografie raamwerk", retrieved from [30] and supporting documentation. We analyzed version v0.65, and then version v0.90, taking into account the accompanying documentation in the documentation repository [31] up to and including commit **443fb932d19697f114d1de167839895e13415c58**, which was downloaded on August 25th 2020 at the end of the day.

The Dutch Crypto Framework document is mostly a description of the TEK[1] life cycle, taking into account the following 3 threat scenarios:

- Upload of TEKs which do not belong to a phone of an infected person.
- Download of TEKs that do not originate from the central server or have been manipulated during transport.
- De-anonymization, except for a temporary non-anonymity during the upload for TEKs.

The Dutch Crypto Framework document then attempts to mitigate these threats.

Most of the important mechanisms and TEK life cycle are implemented by the GAEN (Google/Apple) framework. The app and back-end that are supposed to be implemented on top of it are only part of the overall solution.

This analysis is based on a work-in-progress version of the Dutch Crypto Framework document. We will list potential points of improvement in the following sub-sections. These can be weaknesses in existing mitigations or points that have not (yet) been defined, even though we would expect this from such a cryptographic framework document.

4.1 Underlying Technology GAEN Contact Tracing

The Dutch Crypto Framework document follows the key life cycle. Most of the key life cycle is implemented by the GAEN system. For this portion, the Dutch Crypto Framework document elaborates on any relevant points, such as key randomness and storage. We got the impression that the author is limited by a lack of available information about the GAEN framework. On some points, the author asks questions and presents answers that might have been received from Apple and Google.

The GAEN framework is presented as a closed-source, trusted third-party component. Even though the GAEN framework has recently become partially open source, to the best of our knowledge it has so far not been completely independently verified, so the present level of assurance is questionable. This is also something other countries developing similar apps are struggling with [28]. The shortcomings of and potential attacks on the GAEN system have already been elaborated on in the previous section. It is worth repeating that while some of its associated risks could potentially be mitigated or reduced, some risks are inherent to the chosen decentralized BLE contract-tracing approach.

The Dutch Crypto Framework document lists risks taken from external literature [1] and [13] in an appendix, but marks all of these as not addressed by cryptographic means. Mitigations and explanations are part of the out-of-scope Data

[1] TEKs are explained in section 3.1.2 (page 14).

Protection Impact Assessment (DPIA) [25]. Plenty of mitigations for shortcomings of GAEN are of a non-technical nature, such as laws, fines, and disallowing abuse of the data.

4.2 Non-GAEN TEK Life Cycle

The Dutch COVID-19 notification app has to implement certain features (such as the uploading and downloading of keys) on top of the GAEN framework.

The Dutch Crypto Framework document describes properties of a key upload flow 1 – refer to [11] for the full description. A flow 2 is also mentioned, but it is not discussed any further and the provided link to it does not lead to any public document. We will therefore only analyze flow 1.

Information is spread between the Dutch Crypto Framework document and the solution architecture document [11], which is both confusing and error-prone. Ideally, the cryptographic properties would be defined in one place, and this could then be referenced by the solution. Things should not be defined in multiple locations.

A short overview of the key uploading procedure: Submission of diagnosis keys (TEKs of infected people) is done via an upload to a data bucket in the back-end server. This server later forwards them to a content delivery network (CDN) responsible for distributing these Diagnosis Keys to all user devices, which can then compare them against their received beacons. The app can at any time call the back-end server to receive a LabConfirmationCode (LCC) and a 256 bit ConfirmationKey (CK), which are both linked to an empty data bucket on the server. A key upload into the data bucket can occur when a person is called by the health services and notified of a positive COVID-19 test result. The user can then choose to tell the operator their LCC, which allows the operator to mark the data bucket as positive. The user can then start sharing their keys signed by their CK over a TLS-protected connection.

A short overview of the key downloading procedure: Both the diagnosis keys and the configuration for contact analysis are hosted on the CDN. They are both signed and downloaded over a TLS-protected connection.

As stated in the Dutch Crypto Framework document, the upload is not entirely privacy-preserving; it is clearly possible for entities that have access to the server and the health authority call logs to link the called user to the subsequent upload of TEKs. This is by choice and by design; other implementation options were available. Multiple options are compared in the Dutch Crypto Framework document's appendix. The designers have opted for a solution that can be linked to a person, to mitigate the increased risk of TEKs not belonging to infected people being uploaded. Germany for example has chosen an upload flow which runs a greater risk of this occurring, but is less privacy-invasive, as a user can call a hotline and is provided with a code to upload his keys after a mere verbal plausibility check. [12]

According to the Dutch Crypto Framework document, this potential coupling of patient information to key upload is supposed to be temporary. We note that it might be technically hard to actually strip all of this information away. Additionally it might also be hard to provide guarantees to citizens that this information is not linked, as they are not able to audit the server or network infrastructure. It is also hard to give any guarantees that this information wouldn't be linked to CCTV and existing Bluetooth-tracking networks such as on NS stations [20] or the Schiphol Airport [21]. This server infrastructure can also be changed, which is likewise not transparent to the end user.

There is a remaining risk of malicious uploads of TEKs of non-infected people as mentioned in section 3.2.2.3 (page 19). There is also potential to abuse the infection status to bring the user device into proximity of a victim before performing an upload. There is also no guarantee that an uploader is using their own phone and not the phone of e.g a housemate. This can be used for attack scenarios described in section 3.2.2 (page 18). Once a data bucket is activated, it stays valid for at least ~28 hrs after LCC generation, according to the "Data cleanup" section in [26]. A confirmed infected user who has transmitted his LCC to the operator can use this window to sell his access to a malicious actor or to perform these types of attack himself.

This is a trade-off between user privacy and the risk of fake keys being uploaded.

We find that the risk of back-end impersonation as mentioned in section 3.2.2.2 (page 19) is state-of-the-art mitigated by providing certificate pinning functionality for both the uploading and downloading of keys.

We further find:

- The Dutch Crypto Framework document describes the upload of TEKs in batches into a data bucket on the back-end. It is not clear if TEKs are kept in these batches or are mixed around. If TEKs are also kept in batches during distribution to user devices or the CDN, it would be easier to discover which TEKs belong to a specific person, and to track people using bluetooth tracking networks beyond the 1-day TEK usage period. If TEKs are properly mixed, potentially with decoy keys, and do not contain additional data, finding out which keys belong together would be close to impossible. This is however described in the supporting documentation [33].
- The Dutch Crypto Framework document says that decoy keys are added if necessary to limit the privacy risk in case the number of infected people is very low. The Dutch Crypto Framework document does not explain how many fake keys may be added, with which properties, and under which conditions. This should be properly dimensioned and defined, so that the risk of identifying which TEKs belong to one person during downloads is minimized. Depending on the data connected to the TEKs, such as the infection risk, there might also be a need to add similar information to the fake keys so as not reveal their origin. This is partially described in the supporting documentation [33], but does not specify how key properties are made indistinguishable
- Neither the Dutch Crypto Framework document nor the architecture description [11] describe into detail how the communication between the app back-end for key upload and the CDN for key download is secured. This would also be a critical part of the key transport path.
- The maximum number of TEKs uploaded per LCC/CK is not defined; this should be a reasonable number to prevent people from abusing the system to upload additional (false) keys, keys of other people, etc.

4.3 Other Keys

The TEK upload flow [11] mentions additional keys, namely LCC and CK. The Dutch Crypto Framework document also mentioned using OTP/ICC for authentication in the back-end in version v0.65; these keys were replaced with two-factor authentication in v0.90. This two-factor authentication makes use of a user name and password that are not explicitly defined.

We find:

- The Dutch Crypto Framework document does not define all key properties, such as lifetimes of the LCC and CK. Some key properties are described in the architecture description [11]. Key properties are something we would expect to find in a crypto document.
- It is not defined/unclear when the LCC and CK are first created. They could exist for a long time before the actual key upload. This could give potential attackers a longer time window in which to steal keys from a user and e.g. upload keys in their name. The architecture description [26] however does seem to limit the lifetime of the key. We would expect both the motivations for and strict definitions of LCC and CK to appear in a crypto document.
- The uniqueness of the LCC is not explicitly defined anywhere. The LLC should be unique, so as to not create conflicts between two users (when a user is called and provides their LCC, this should not activate the data bucket of any other user). LCC collision is however statistically unlikely, and also depends on the LCC's lifetime.
- The LCC and CK should only be used for their intended purpose and not for any additional communication, because they are unique identifiers. If they are e.g. used during regular communication, they can be used to identify users.
- There is no definition of how these keys are stored when at rest. Safe storage is required, because a stolen/ extracted LCC and CK could be abused to upload the keys on another device.
- There is no minimum strength given for the username/password combination used with the two-factor authentication.
- In the supporting documentation [11], the component for health authority employee login is said to be The Identity Hub, which is a 3rd party solution, which is not further defined.

4.4 International Diagnosis Key Exchange

The Dutch Crypto Framework document states that agreements for international key exchange are made in the document "20200603_CWA_Interop_Architecture.doc". We did not review this document; it is not part of the repository. There is a summary given which only explains how communication with the Federation Gateway is secured. The Federation Gateway is described in the supporting documentation as a diagnosis key exchange platform for European countries using the GAEN framework. [27]

The Dutch Crypto Framework document does not mention further points such as:

- Prevention of fake/false-positive keys from other countries. This could happen, because other countries have other (weaker) processes to upload keys into the system. The federation gateway architecture [27] states that countries can filter or tag these keys before distributing them, but this is not described any further here. Therefore it is hard for us to judge if the Dutch choice of upload method (reducing fake/false-positive keys at a cost of reduced user anonymity) is still a valid trade-off given that fake/false-positive keys could end up in the system through international transfers anyway.

- Storage duration and conditions of diagnosis keys. Keys can be deleted from Dutch servers, but might be retained for longer on international servers. This is a minor concern because in theory any user could just download and store all keys forever.
- The Dutch Crypto Framework document does not state whether decoy keys are also uploaded to the federation gateway.
- The supporting documentation [34] states that keys will get an additional 'region of interest' tag. The supporting documentation correctly mentions that this has an effect on decoy keys, but does not provide enough details to enable us to validate that decoy keys will be indistinguishable from real ones. This new key property could require a significantly larger number of decoy keys to successfully hide these additional properties.
- The supporting documentation [34] states that users can use the app to select the regions they have travelled in, in order to only download diagnosis keys from those regions. The supporting documentation correctly states hat this could be abused by the server to check which user (identified by IP address) travelled where. Instead of choosing an approach that would make traffic from individual app users indistinguishable, the designers have opted for a solution that requires the back-end to not log IP addresses. This means that privacy might be hard to guarantee to end-users. Other approaches are available which would make it easier to guarantee privacy e.g. by requiring downloading all diagnosis keys of all countries all the time.
- The supporting documentation [34] states that the health authority will enter the countries a user has travelled to into the portal, probably during a call in which the user's name and phone number are known to the health authority. There is no technical solution preventing them from keeping hold of this information. Other approaches are possible: a users could e.g. select the countries he has travelled to himself and upload this information without needing to use the health authority as an intermediary. This is a again a trade-off between in this case false key properties and privacy.

5 Back-end Code Review

We partially reviewed several code-bases in the repository for the Dutch COVID-19 notification app back-end, retrieved from [32]. We occasionally stopped the ongoing evaluation and continued on a newer version due to major updates. We looked at commits:

- c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4
- f7b160c5ca6f8f8f428d0b83ed28ef5c63893191
- 1813b997e5ba7085abf1ea043a6d8b6020618aea
- 601afa149dd631593589b383266e6b4185ab2524
- 601afa149dd631593589b383266e6b4185ab2524
- 8e6e22fce98f7914e1f43aa51e5d17ad127dae93
- c180ff31c5d451486e39f6fb54175adab73a8074
- 3765936e4a6df5320a5d3593d952d27b19fe8f92
- eddf6131ecafc2227155f3b3325b8e9f512d9b51

eddf6131ecafc2227155f3b3325b8e9f512d9b51 is the latest evaluated version, which was downloaded on August 25th 2020, at the end of the day. The back-end is still in development as it still misses the international key exchange features; We do not expect this to be the final version.

The back-end repository contains several separate applications. We only evaluated the following folders:

- **DbProvision**: a tool to provision the database, e.g. during deployment
- **ICCBackend**: the API used by health authority employees to authorize uploaded keys
- **MobileAppApi**: the API used by the Mobile app to upload keys
- **EksEngine**: a tool that takes keys submitted to the MobileAppApi and publishes them in the ContentApi
- **ContentApi**: the API facing the CDN to deliver diagnosis keys and other content to the app
- **DbFillExampleContent**: a tool to fill the database with example content
- **ManifestEngine**: a tool to generate a manifest file for the CDN
- **Components**: a collection of help components. We only looked at components used by other applications in this list, not the entire collection.
- **Database**: database structure and settings

We did not evaluate the test/development applications in the folders:

- BatchJobsApi (later removed from repository)
- Components.Tests
- docker
- EksParser
- elevated

- ForceTekAuth
- GaenFormat
- GenTeks
- ManagementPortal (later removed from repository)
- MobileAppApi.Tests
- ServerStandAlone (later removed from repository)
- SigTestFileCreator
- Testautomation

5.1 Limitations of a Code Review

The scope of this project is limited to a review of the code in the back-end repository. This means that this project could not test the actual deployment or any external components, including:

- The stripping of metadata such as IP addresses from incoming connections
- The actual back-end configuration and secrets used on the server
- The server configuration to prevent e.g. telemetry leaking data
- Separation of privileges in the databases, such as per-service and per-table read/write permissions
- The configuration of the database and whether it stores metadata or securely deletes data
- The security and configuration of other services on the servers, for example ssh
- Certificates, private keys, and their storage e.g. for signing diagnosis key batches
- Whether decoy traffic responses from the MobileAppApi look sufficiently similar to non-decoy traffic.

Two points are of critical importance and must be checked during a penetration test:

It is important that any private key material is stored and used securely. This is required for the correct functioning of certificate pinning, which is a major security feature to prevent impersonation of the server facing the app. It is also important for signing diagnosis key packages and other materials distributed to the apps. To comply with the Dutch Crypto Framework document and security best practices, HSMs must be used. We were told by the developers on August 24th 2020 that the HSMs are still being deployed, so this is something important to verify.

The code we have seen does not implement IP stripping or removal of other connection metadata. This is something that would have to be implemented by the infrastructure on which these applications are running. This feature is crucial to fulfilling the promise of user anonymity.

The `deployment.md` document in the `IccBackend` folder shows the reverse proxy path as HTTP instead of HTTPS. This was discussed during a call with the developers, who confirmed that HTTPS is indeed terminated at the load balancers. This means that the load balancers see both the entire unencrypted traffic content such as Diagnosis Keys and its metadata such as IP addresses and timestamps. This would not limit access to the keys to the machines that absolutely need it.

This could be reimplemented so that the load balancers are not exposed to unencrypted data and back-end servers never see user IP addresses. Verifying proper infrastructure configuration has to be covered by a penetration test of the deployed systems, and is unfortunately not something we can do in a code-only review.

5.2 Looking for Potential Back-doors

We were explicitly asked to look for any potential back-door functionality.

By design this solution holds relatively little user data. Essentially it only uses the TEKs/diagnosis keys, including associated data such as country of origin, and the user-specific uploaded data such as LCC, CK, and related data. There is little data to gain from exploiting the service, but it could be used exploited for e.g. denial of service or other scenarios as described in section 3.2 (page 17).

The overall implementation is roughly in line with the documentation/design/architecture in [31]. The implementation does deviate from the specification in multiple places, and there are undocumented features, but this is to be expected given the urgency and development speed of this project.

We have not found any code that might represent an intentionally-placed back door to control the server. These could of course still be put in place during deployment.

Inadvertent back doors could also hide behind bugs or innocent implementation choices. Therefore we want to highlight a few of our findings:

NLC-013 (page 31) is an API that allows the health authority employees to check if a user really uploaded a TEK. This could be considered sensitive information and is not technically necessary.

We found options to deactivate or manipulate security features for testing purposes in NLC-002 (page 32); these should not be present in production in a system of this sensitivity. This has since been resolved.

There is more data collected than strictly necessary, or it is retained for longer than necessary when compared to a complete privacy-first approach; see NLC-009 (page 46), NLC-008 (page 45), NLC-025 (page 54), and NLC-027 (page 56). Only finding NLC-009 has since been resolved.

A third party is used for authenticating the health authority employees. When acting as a malicious agent or when hacked, this third party could impersonate health care authority employees, see NLC-019 (page 30).

5.3 List of Findings

We identified the following issues during the code review.

Findings are sorted by severity. Findings that are resolved in the latest evaluated version are marked as resolved. Findings that are only partially resolved are still listed with their original impact rating- The software version in which the issue was encountered is listed for each finding; this does not mean the issue might not have already existed in earlier versions of the software.

5.3.1 NLC-019 — Third Party Authentication Provider Used

Vulnerability ID: NLC-019	**Status:** Unresolved
Vulnerability type: Authentication issue	
Threat level: High	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

The solution uses The Identity Hub (https://theidentityhub.com) as its OAuth provider to allow users access to the healthcare API.

Technical description:

`Startup.cs` in IccBackend:

```
           services
               .AddAuthentication(auth =>
               {
                   auth.DefaultChallengeScheme = TheIdentityHubDefaults.AuthenticationScheme;
                   auth.DefaultAuthenticateScheme =
CookieAuthenticationDefaults.AuthenticationScheme;
                   auth.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
               })
```

The IccBackend uses The Identity Hub as its OAuth provider. Successful authentication through The Identity Hub is needed to generate a valid JSON Web Token (JWT), which is in turn required to authorize diagnosis key uploads.

This means that if The Identity Hub acts maliciously or gets compromised, the IccBackend is compromised as well.

Using third party authentication components can be a valid choice, but from a code review perspective it adds another attack path, so we included it as a finding.

Impact:

If The Identity Hub gets compromised or acts maliciously, IccBackend is compromised as well.

Recommendation:

Implement user management without a third party or verify that The Identity Hub can be trusted. Check that this party is trustworthy and regularly tested and audited.

5.3.2 NLC-013 — Health Authority Employees Can Check If Keys Were Uploaded

Vulnerability ID: NLC-013	**Status:** Unresolved
Vulnerability type: Privacy issue	
Threat level: Elevated	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

An API allows health authority employees who have authorized a key upload to check if the user really did upload the keys. This reduces anonymity and could be abused to coerce users.

Technical description:

`HttpPostLabVerifyCommand.cs` contains a function that allows health authority employees who just authorized a key upload to check if the user actually uploaded the keys. During the call, the user is known to the health authority (by name and phone number). This feature is probably intended to be used during the call, as it allows the employee to give the user feedback.

Unfortunately this also means that the API gives health authority employees a way to coerce or persuade the user. Without this function, the health authority employees wouldn't know if a user voluntarily uploaded his key. The feature therefore casts doubts on how voluntarily an upload is. Even if it is not abused now, it could be in the future.

It also reduces anonymity, because the health authority could note down which user uploaded a key, including name and phone number, to make sure the key is part of the next published batch.

Only the employee who authorized the key upload can check the status, but nothing prevents the health authority from letting all employees use one and the same account.

The feature can also be used for statistics, e.g. to find our how many people really did upload their keys based on which call.

Impact:

Health authority workers know if a user uploaded their keys, resulting in reduced anonymity and potential scenarios in which people might be coerced.

Recommendation:

Remove this feature.

5.3.3 NLC-002 — Developer Options To Switch Off Security Features

Vulnerability ID: NLC-002	**Status:** Resolved
Vulnerability type: Misconfiguration potential	
Threat level: Moderate	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

The software contains options to switch off security features for development.

Technical description:

`DevelopmentOnlySettings.cs` contains options to switch off security features such as the back-end's user authentication and use of HTTPS. The default is to have these security features switched on, which is good. As long as the option to switch off significant parts of the security features is present, however, these features might conceivably be turned off accidentally.

Impact:

Some security features might get switched off due to misconfiguration.

Recommendation:

Do not offer the option to switch off security features.

Retest update:

Developer options removed.

5.3.4 NLC-004 — RollingPeriod Not Limited To One Day

Vulnerability ID: NLC-004	**Status:** Resolved
Vulnerability type: Data sanitization	
Threat level: Moderate	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

The RollingPeriod of uploaded keys is not properly checked. Uploaded keys can continue into the following days.

Technical description:

`appsettings.release.json`:

```
"RollingPeriod" : {
    "Min": 1, //Coded default
    "Max": 2147483647
}
```

`DefaultGaenTekValidatorConfig.cs`:

```
public int RollingPeriodMax => int.MaxValue;
```

Each TEK is valid for RollingPeriod intervals of 10 minutes. A TEK is valid for a maximum of 144 intervals (1440 minutes, 24 hours) until it is changed as defined by GAEN.

This missing check allows keys with RollingPeriods longer than one day to be uploaded to the system.

Impact:

An attacker could upload keys to the server database with a lifetime that stretches longer than one day. The impact depends on whether such keys are rejected by the mobile phone clients, however, which is out of our scope.

Recommendation:

Implement a check against the `RollingPeriod` maximum of 144.

Retest update:

A check has been implemented; the `RollingPeriod` is now limited to 144.

5.3.5 NLC-005 — It Is Possible to Update Diagnosis Keys With Future Dates

Vulnerability ID: NLC-005	**Status:** Resolved
Vulnerability type: Data sanitization	
Threat level: Moderate	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

MobileAppApi does not check if an uploaded TEK is associated with a future date.

Technical description:

`TemporaryExposureKeyValidator.cs` validates the uploaded TEKs, but does not implement a check on the `RollingStartNumber` that indicates when a key became active.

The developers are apparently aware of this issue:

```
//TODO valid values epoch size, currently 10mins, for value.RollingStartNumber
```

Impact:

Attackers could upload keys to the database that would be activated at a later date.

Recommendation:

Implement a check to validate the TEK `RollingStartNumber`.

Retest update:

A check has been implemented.

5.3.6 NLC-007 — Decoy Traffic Is Not Efficient

Vulnerability ID: NLC-007	**Status:** Unresolved
Vulnerability type: Privacy issue	
Threat level: Moderate	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

The decoy traffic analysis is not in line with the design and does not seem to be able to successfully mimic real traffic.

Technical description:

Note that this section is not based on actual traffic analysis but solely on the code review.

The decoy traffic design [35] states that the intention is to mitigate network traffic analysis by systematically sending 'decoy' traffic that is indistinguishable from real TEK uploads and key upload registration API calls.

The back-end has to support this process by making the 3 MobileAppApi commands indistinguishable from a traffic perspective, by having a similar answer size and response size.

We find that:

- the response size is not similar.
- the response time is not similar.

There is no padding implemented.

There is a response delay implemented, but only in the `HttpPostDecoyKeysCommand` command.

```
var delayInMilliseconds = _RandomNumberGenerator.Next(_Config.MinimumDelayInMilliseconds,
 _Config.MaximumDelayInMilliseconds);
              _Logger.LogDebug("Delaying for {DelayInMilliseconds} seconds", delayInMilliseconds);
        await Task.Delay(delayInMilliseconds);
```

Partial fix in version 1813b997e5ba7085abf1ea043a6d8b6020618aea:

Version 1813b997e5ba7085abf1ea043a6d8b6020618aea adds padding to the responses. This creates responses with similar randomized lengths.

Impact:

Entities observing the network traffic can find out if a user has uploaded keys and infer that the user is likely infected.

Recommendation:

Implement the required features and test them with a Man-in-the-Middle setup on a deployed system.

5.3.7 NLC-014 — Authorization Token Included in GET Parameter

Vulnerability ID: NLC-014	**Status:** Unresolved
Vulnerability type: Authentication issue	
Threat level: Moderate	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

The JWT for authentication against the health authority back-end is part of a `GET` parameter.

Technical description:

`HttpGetAuthorisationRedirectCommand.cs` uses:

```
return new RedirectResult(_Configuration.FrontendBaseUrl + "/auth?token=" + jwtToken);
```

This means that the token gets transmitted as a `GET` parameter. This an unsafe practice, as a `GET` parameter might for example end up in log files, which is not recommended for non-single use tokens such as this one.

Partial fix in version eddf6131ecafc2227155f3b3325b8e9f512d9b51:

There is now a single-use `authorizationCode` generated by `AuthCodeService.cs` transmitted in the GET parameter instead. This code can be used to retrieve a `jwtToken`. It is an improvement that the code transmitted is single-use, but the `authorizationCode` itself does not expire and the token is of limited complexity compared to the security provided by 2-factor OAuth and JWT.

Impact:

JWT might end up in log files. If others have access to this log files, they gain authorization.

Recommendation:

Transmit JWT as part of e.g. a `POST` parameter or make the JWT token single-use and have it expire immediately.

5.3.8 NLC-017 — Databases have Insufficient Privileges Separation

Vulnerability ID: NLC-017	**Status:** Resolved
Vulnerability type: Privilege separation	
Threat level: Moderate	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

Even though the databases are accessed by different programs with different needs, there is limited privilege separation.

Technical description:

The back-end manages its data in one Microsoft SQL Server. Different services need access to different databases: `Content`, `PublishingJob` and `Workflow`

- MobileAppApi puts the received content in `Workflow`
- EksEngine creates content in `Content` from `Workflow` using `PublishingJob` for help
- ContentApi takes its content from `Content`

These programs are supposed to run on two different servers. There is no need for all programs to be able to access all the data, but they can. We did not find any user read/write privilege separation between the 3 databases. We would expect to find this in the Database folder, but it might also be added later during deployment.

Impact:

The compromise of one service leads to more access to the database than necessary.

Recommendation:

Implement database read/write access permissions on a per-program basis.

Retest update:

Read/write access permissions are now implemented on a per-program basis.

5.3.9 NLC-018 — Signing Components Imply Running as Local Administrator

Vulnerability ID: NLC-018	**Status:** Resolved
Vulnerability type: Privilege separation	
Threat level: Moderate	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

Some components imply they need to run as local administrator. If they do, their compromise would occur in the context of the administrator role instead of in that of an isolated process.

Technical description:

`CmsSignerEnhanced.cs`: The signing components check for admin privileges. If they do not have access to these privileges, they continue running but present an error message. We concluded that they are supposed to run as local administrator, which was confirmed by the developers, who stated that this is necessary since elevated permissions are needed to access the certificates. Typically it is recommended that processes run in their own isolated context instead of having full access to the system. If an attacker obtains run-time control of a vulnerable program, he only gains rights within that limited context. We find that the programs containing the `CmsSignerEnhanced.cs` also contain functionality that does not require admin privileges.

Impact:

An attacker compromising a component running as administrator can gain control of the entire system instead of just the process context.

Recommendation:

Do not run any components with administrator rights. If strictly required, isolate this specific functionality and pack it into a separate program with a minimal attack surface.

Retest update:

The developers have corrected their statement: the code is not intended to be run with admin privileges. They now state that this is just for development purposes.

We consider this resolved, because the code indeed never forces the users to run anything as admin and the finding was only included because a developer initially confirmed our conclusion.

5.3.10 NLC-020 — Health Worker Authentication Token Is Valid For Several Hours and Not Rate-Limited

Vulnerability ID: NLC-020	**Status:** Unresolved
Vulnerability type: Authentication issue	
Threat level: Moderate	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

The JWT used for authentication is valid for several hours. There is no rate limiting nor any way to revoke individual authentications.

Technical description:

A JWT token is used for authentication in the IccBackend.

In the default configuration this token is valid for 3 hours:

`JwtService.cs`:

```
builder.AddClaim("exp",
 _DateTimeProvider.Snapshot.AddHours(_IccPortalConfig.ClaimLifetimeHours).ToUnixTime());
```

The server does not keep track of tokens, but can verify that they are legitimate and still valid using the individual properties.

The lifetime has to be selected carefully, because if a token gets stolen, it stays valid and cannot be revoked unless an administrator changes the secret key for all tokens. There is no logout function to deactivate a token. An API user can only choose to forget the token. No rate limiting is implemented, so the token could be used to authenticate an infinite amount of diagnosis keys. The actual configuration is not known yet and anyone deploying the code might configure a very long lifetime for reasons of convenience.

Partial fix in version eddf6131ecafc2227155f3b3325b8e9f512d9b51:

Whenever the JWT is used for authentication, the system checks that the authorization by the OAuth provider is still valid. This can be used for revocation. There is still no rate limiting, however, which means that the number of actions an attacker could carry out with a compromised login is only limited by network speed and potential DoS protections in the infrastructure. It is not limited to a reasonable number of user interactions.

Impact:

A compromised authentication token is usable without limitation until it expires.

Recommendation:

- Implement rate limiting for requests that use the same token
- Consider limiting the token lifetime to a reasonably secure value, or implement token revocation.

5.3.11 NLC-022 — HSTS Header Not Set

Vulnerability ID: NLC-022	**Status:** Unresolved
Vulnerability type: HSTS	
Threat level: Moderate	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

A HTTP Strict-Transport-Security response header (HSTS) is recommended for all APIs that only use SSL, in order to prevent adversaries from using SSL stripping attacks in a MitM scenario.

Technical description:

For more information, visit https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security

A discussion with the developers revealed that HTTPS is terminated at the load balancer. This could mean this finding is invalid; we cannot confirm this in a code-only review. We do not recommend terminating HTTPS here, to prevent information being available in plain text on the infrastructure network.

Impact:

Mitigation against SSL stripping attacks is limited.

Recommendation:

Add HSTS to the relevant APIs and set a proper lifetime.

5.3.12 NLC-028 — Timing Oracle in Signature Comparison

Vulnerability ID: NLC-028	**Status:** Unresolved
Vulnerability type: Authentication issue	
Threat level: Moderate	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

The signature validation used during diagnosis key upload contains a timing oracle.

Technical description:

`SignatureValidator.cs`:

```
using var hmac = new HMACSHA256(confirmationKey);
var hash = hmac.ComputeHash(data);
return hash.SequenceEqual(signature);
```

The Hash-based Message Authentication Code (HMAC) signature provided by the uploader of a diagnosis key is compared against an HMAC created by the server from the provided data. The comparison function `SequenceEqual` has different execution times depending on which byte in a sequence is not matching. The earlier, the shorter the execution time. This allows attackers who are able to carry out a large number of attempts to find the correct HMAC signature for a provided message, with significantly less effort than it would take to brute-force the complete HMAC.

This might not be applicable in practice, because the time difference is in the microsecond/nanosecond range, which might not be measurable over a network connection.

Impact:

Attackers could upload diagnosis keys for a user without knowledge of the proper confirmation key.

Recommendation:

Use a time-constant compare function.

5.3.13 NLC-029 — AuthorizationCode is weak

Vulnerability ID: NLC-029	**Status:** Unresolved
Vulnerability type: Authentication issue	
Threat level: Moderate	**Software version:** eddf6131ecafc2227155f3b3325b8e9f512d9b51

Description:

The authentication against the health authority back-end contains a weak single-use component.

Technical description:

`AuthCodeService.cs` creates a `authorizationCode` which is a 12-character string containing only base64 characters. There is an API endpoint that does not require authentication, which allows exchanging this single-use `authorizationCode` for a time-limited-use JWT token. A health authority employee receives such an `authorizationCode` after successfully authenticating with the The Identity Hub OAuth provider.

There are multiple issues with this approach:

- The complexity of the `authorizationCode` token is very limited compared to the `HMACSHA256` JWT and the 2-factor authentication that are used by the rest of the authentication flow.
- The function to check whether the `authorizationCode` exists is `ConcurrentDictionary.TryGetValue()` which is not a secure comparison function and mights contain timing oracles. This might be hard to exploit over a network connection.
- If the `authorizationCode` is not used immediately, e.g. because of an error or because the user aborts the process, it stays in the database and could be attacked by third parties.

We rate this only as a moderate threat, because guessing 12 base64 characters is still a complex task and these codes only continue to exist if they are not used immediately.

Impact:

External parties might gain authorization in the ICCBackend.

Recommendation:

Replace the `authorizationCode` with an immediately expiring single-use JWT token with at least `HMACSHA256`

Optional: consider simplifying the authentication, so that this redirect with a GET parameter and the `authorizationCode` is no longer required.

5.3.14 NLC-001 — Deviation Between (API) Design and Implementation

Vulnerability ID: NLC-001	**Status:** Unresolved
Vulnerability type: Inconsistent documentation	
Threat level: Low	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

The architecture and design deviates from the implementation. Inconsistent documentation can be a source of errors and confusion.

Technical description:

We see several endpoints that are not defined in

- https://github.com/minvws/nl-covid19-notification-app-coordination
- https://editor.swagger.io/?url=https://raw.githubusercontent.com/minvws/nl-covid19-notification-app-coordination/master/architecture/api/apispec.yaml

There are also other inconsistencies between the implementation and the architecture; the latter is updated less frequently.

Impact:

Inconsistent documentation can be a source of errors and confusion. Security features defined in the design are not implemented in the application. We have listed dedicated findings for those cases where there is an impact on the solution's security.

Recommendation:

Update the documentation, designs, and architecture consistently when updating the product.

5.3.15 NLC-003 — Database Credentials in Components That Do Not Require Them

Vulnerability ID: NLC-003	**Status:** Resolved
Vulnerability type: Privilege separation	
Threat level: Low	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

Connection strings containing database credentials are included in the `appsettings.json` of components/servers that do not require them.

Technical description:

The files `appsettings.json` and `appsettings.release.json` are present in all sub-projects of the back-end. They contain the (placeholder) connection strings for the database connections. In version 1813b997e5ba7085abf1ea043a6d8b6020618aea the `appsettings.release.json` of e.g. MobileAppApi contains:

```
 "ConnectionStrings": {
   "Content": "Data Source=#{Common.DatabaseServer}#;Initial
Catalog=#{Common.Content.DatabaseName}#;Integrated Security=True",
   "WorkFlow": "Data Source=#{Common.DatabaseServer}#;Initial
Catalog=#{Common.WorkFlow.DatabaseName}#;Integrated Security=True",
   "PublishingJob": "Data Source=#{Common.DatabaseServer}#;Initial
Catalog=#{Common.PublishingJob.DatabaseName}#;Integrated Security=True"
 }
```

This implies that this component, which does not use any database access other than "WorkFlow", would contain other credentials as well, needlessly spreading them. This finding is not valid if all components access the database through only one user. In that case however it might be even easier for an attacker to access all data when a server is compromised.

Impact:

The compromise of one component could lead to exposure of more credentials than necessary.

Recommendation:

Remove connection strings from components that do not require them.

Retest update:

The connection strings were removed from components that do not require them in `appsettings.json`.

5.3.16 NLC-008 — Database Stores Upload and Phone Call Times Etc.

Vulnerability ID: NLC-008	**Status:** Unresolved
Vulnerability type: Privacy issue	
Threat level: Low	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

The database stores time values relating to diagnosis keys. In combination with other information, this can aid de-anonymization efforts.

Technical description:

`TekReleaseWorkflowStateCreate.cs`:

```
Created = _DateTimeProvider.Snapshot
```

This stores the precise time at which a diagnosis key upload data structure is created. This information is never used.

`AuthorisationWriterCommand.cs`:

```
wf.AuthorisedByCaregiver = _DateTimeProvider.Snapshot;
```

This is used to limit the time in which a user is allowed to upload a key after a call.

`HttpPostReleaseTeksCommand2.cs`:

```
i.PublishAfter = _DateTimeProvider.Snapshot;
```

This value is used to indicate the time after which a key is allowed to be published.

Storing any time information that relates to the time of uploading, phone call, etc. can aid de-anonymization efforts.

Version eddf6131ecafc2227155f3b3325b8e9f512d9b51:

The example in`TekReleaseWorkflowStateCreate.cs` now only stores a date:

```
Created = _DateTimeProvider.Snapshot.Date
```

The remaining examples reaming unchanged.

We find one additional example:

`WriteNewPollTokenWriter.cd`:

```
wf.PollToken = _PollTokenService.Next();
```

The poll token contains a time value which is created during the call with a user. The time is required for checking the token validity, but the token is not deleted after it expires so the time stays in the database.

Impact:

Stored time information can aid de-anonymization efforts.

Recommendation:

Do not store unnecessary time information. If time information is required, keep it as imprecise as possible, e.g. by increasing its granularity to day or hour.

5.3.17 NLC-009 — Log Messages Store Sensitive Information

Vulnerability ID: NLC-009	**Status:** Resolved
Vulnerability type: Privacy issue	
Threat level: Low	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

The log messages store information which, together with log timestamps, can aid de-anonymization efforts.

Technical description:

`HttpPostReleaseTeksCommand2.cs`:

```
_Logger.LogDebug("Body -\n{ArgsJson}.", argsJson);
```

This saves all user uploaded diagnosis keys in the log file. Logs are typically stored with timestamps. This is against the goal of storing data only when necessary and for as short a period as possible.

`JwtService.cs`:

```
_Logger.LogError("Invalid jwt token, Other error - Token:{Token} Exception:{e}", token, e);
```

This stores the authentication token which can be used to log into the IccBackend. The log message is luckily only generated in case the token decoding fails, which might mean the token was invalid anyway.

`WriteNewPollTokenWriter.cs`:

```
_Logger.LogInformation("Committed - new PollToken:{PollToken}.", wf.PollToken);
```

This stores the `PollToken` which can be used to check if an diagnosis key upload was performed. This is not critical as somebody who is able to read this log file likely has access to the server anyway. It is however still bad practice to store key material in logs.

Whether this information is really stored depends on the configured `LogLevel`.

Impact:

Data is stored for longer than necessary and ends up in the log files.

Recommendation:

Do not include sensitive data in log files.

Retest update:

The mentioned instances have been removed. We did not find additional instances.

5.3.18 NLC-010 — API Exceptions Are Too Verbose

Vulnerability ID: NLC-010	**Status:** Resolved
Vulnerability type: Information disclosure	
Threat level: Low	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

Developer exception page function is activated in production.

Technical description:

`Startup.cs` in MobileAppApi:

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, IServiceProvider services)
{
    app.UseDeveloperExceptionPage();
```

Citing the official Microsoft documentation: "Enable the Developer Exception Page only when the app is running in the Development environment. You don't want to share detailed exception information publicly when the app runs in production."

Impact:

Information disclosure; developer exception page is active.

Recommendation:

Remove this feature.

Retest update:

A check was added so that this feature is only activated in the development environment.

5.3.19 NLC-012 — JWT Token Not Properly Validated

Vulnerability ID: NLC-012	**Status:** Unresolved
Vulnerability type: Authentication issue	
Threat level: Low	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

Any valid JWT token is accepted as valid regardless of its properties. As this token needs to match the one stored in the database, this does not currently result in an issue. It could however result in confusion or issues in future updates.

Technical description:

In `PollTokenService.cs`, function `Validate` does not validate the JWT token properly. It accepts any currently valid JWT regardless of its embedded properties. The JWT generator is also used for authentication with the back-end, so the user already has a token that would pass this check.

This does not however cause a vulnerability, because the token also has to match the token stored in the database exactly.

Version eddf6131ecafc2227155f3b3325b8e9f512d9b51:

Function renamed to `ValidateToken`

Impact:

None at this time, but the function naming could give the developers a false idea of token validity.

Recommendation:

Check the presence of the poll token's embedded properties during validation or use a different secret than for the authentication token.

5.3.20 NLC-015 — Database Provision Default Setting Is To Generate Example Content

Vulnerability ID: NLC-015	**Status:** Resolved
Vulnerability type: Misconfiguration potential	
Threat level: Low	**Software version:** c2a8ec8c9c2ca86d6a2099d1864ad78c7c7ceab4

Description:

The default configuration is to add example keys to the database.

Technical description:

`ProvisionDatabasesCommand.cs` in DbProvision: The default configuration is to delete the database and add example content. This could lead to example content (in this case false keys) showing up in the live environment. This should ultimately not have any impact on proper operation, however, as fake keys get added during normal operation as well.

Impact:

The database provision default setting could lead to example content (in this case false keys) showing up in the live environment.

Recommendation:

We recommend setting the default to not adding example content.

Retest update:

Example content functionality has been removed from the DbProvision program and moved to the separate DbFillExampleContent program. We consider this fixed, because the functionality is now clearly isolated.

5.3.21 NLC-016 — Decoy Keys Can Largely Be Distinguished From Real Diagnosis Keys

Vulnerability ID: NLC-016	**Status:** Unresolved
Vulnerability type: Privacy issue	
Threat level: Low	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

The decoy keys added to increase anonymity can mostly be distinguished from real diagnosis keys by their properties.

Technical description:

`EksStuffingGenerator.cs`: When low numbers of TEKs are published, extra TEKs are added. Extra TEKs are added to prevent linking the published TEKs to other TEKs by the same user. Random TEKs always have the property `TransmissionRiskLevel = TransmissionRiskLevel.Low`. Other TEKs that do not have a low transmission risk are therefore clearly real TEKs.

This means e.g. that:

- if only one person uploads diagnosis keys in one time interval, these could immediately be identified.
- if multiple people upload diagnosis keys in one time interval, one could count the number of real TEKs and thus infer the number of uploaders.

Knowing which TEKs belong together in combination with a beacon tracking network allows for the cross-RollingPeriod tracking of individuals.

Partial fix in version 601afa149dd631593589b383266e6b4185ab2524:

The code now creates decoy keys that imitate all used transmission risk levels

`EksStuffingGenerator.cs`: `TransmissionRiskLevel = (TransmissionRiskLevel) (i % 3) + 1`

This creates a roughly equal number of keys of all risk levels. This might still hint at how many users uploaded their keys in total, as real keys do not have completely equal distribution of risk levels. This finding's threat level was originally Elevated; after this partial fix, however, it has been set to Low.

Impact:

Decoy keys can be identified, which makes it possible to filter them out.

Recommendation:

Implement indistinguishable decoy keys.

5.3.22 NLC-021 — IccBackend API Allows CORS Requests From Any Source

Vulnerability ID: NLC-021	**Status:** Resolved
Vulnerability type: CSRF	
Threat level: Low	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

The Cross-Origin Resource Sharing (CORS) policy facilitates Cross-Site Request Forgery (CSRF) attacks

Technical description:

`Startup.cs` in IccBackend:

```
            app.UseCors(options =>
                options.AllowAnyOrigin().AllowAnyHeader().WithExposedHeaders("Content-
Disposition")); // TODO: Fix CORS asap
```

This allows CORS requests from any source, which can facilitate e.g. CSRF attacks. There is no need for this API to be available publicly.

Impact:

Potentially none, because CORS is configured to not allow credentials and all APIs require credentials.

Recommendation:

Limit CORS requests to intended origins.

Retest update:

CORS requests have been limited to intended origins.

5.3.23 NLC-023 — JWT Token Needlessly Contains Employee Information

Vulnerability ID: NLC-023	**Status:** Resolved
Vulnerability type: Privacy issue	
Threat level: Low	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

The health authority employee authentication token contains the employee's email address and name, which are not used by the back-end.

Technical description:

`JwtService.cs`:

```
        public string Generate(ClaimsPrincipal claimsPrincipal)
        {
            if (claimsPrincipal == null)
                throw new ArgumentNullException(nameof(claimsPrincipal));

            var builder = CreateBuilder();
            builder.AddClaim("exp",
 _DateTimeProvider.Snapshot.AddHours(_IccPortalConfig.ClaimLifetimeHours).ToUnixTime());
            builder.AddClaim("id", GetClaimValue(claimsPrincipal, ClaimTypes.NameIdentifier));
            builder.AddClaim("email", GetClaimValue(claimsPrincipal, ClaimTypes.Email));
            builder.AddClaim("access_token",
                GetClaimValue(claimsPrincipal, "http://schemas.u2uconsult.com/ws/2014/03/identity/
claims/accesstoken"));
            builder.AddClaim("name",
                GetClaimValue(claimsPrincipal, "http://schemas.u2uconsult.com/ws/2014/04/identity/
claims/displayname"));
            return builder.Encode();
        }
```

The health authority employee email address and name are included in the plaintext JWT token. This data is not used by the back-end. We do not see any reason why this data is included in the token, unless it is required by the front-end.

Impact:

Employee data is included in a plaintext token.

Recommendation:

Do not include sensitive or superfluous data in the token.

Retest update:

The email address has been removed from the token. The name stays in the token, as according to the developers it is displayed in the front-end. We deem this to be an acceptable practice.

5.3.24 NLC-025 — Old Diagnosis Keys Are Not Deleted

Vulnerability ID: NLC-025	**Status:** Unresolved
Vulnerability type: Privacy issue	
Threat level: Low	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

Uploaded diagnosis keys are not deleted, only marked as published.

Technical description:

We did not manage to find a function that would handle the deletion of diagnosis keys once they are not required anymore; we only see them being marked as published.

`ExposureKeySetBatchJobMk3.cs`:

```
 foreach (var i in zap)
{
    i.PublishingState = PublishingState.Published;
}
```

According to the privacy-first principle all data should be deleted as quickly as possible.

Impact:

Diagnosis keys stay on the server for longer than necessary.

Recommendation:

Delete the keys instead of marking them as published.

5.3.25 NLC-026 — Diagnosis Key Publishing Time Can Reveal the Time of the Phone Call

Vulnerability ID: NLC-026	**Status:** Unresolved
Vulnerability type: Privacy issue	
Threat level: Low	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

There is no intentional delay between the time of uploading and the time of publishing for diagnosis keys.

Technical description:

An infected user can upload their TEKs to the server as soon as a health authority employee calls them with a positive result. The moment of this call is potentially known to the network operator, people who are in the vicinity when the call is made, and the health authorities.

A regularly executed task - EksEngine - publishes the keys and adds decoy keys. Depending on how frequently the EksEngine job is run, there is little delay between the phone call and the diagnosis key becoming available to the public.

This means that the downloaded keys then have a high likelihood of being either decoys or related to a call that just occurred.

There are a few scenarios for which this produces a de-anonymization advantage. One example could be that the telephone company knows somebody was called by the GGD and that the call was long enough to make it likely that it was about a positive test result. They can then assume that the key of this person will be in the next batch.

This is a risk that can never be fully mitigated; it is only possible to lower the likelihood of a key being in or being alone in a certain batch, immediately after a phone call.

Impact:

The approximate time of the call to an infected person can be derived from the time when the diagnosis key is published.

Recommendation:

- Introduce a significant random delay between the time of uploading and the time of publishing, so that a key can end up in one of several batches.
- Collect a minimum amount of real diagnosis keys before publishing, so that a key is never the only one in a batch.
- Run EksEngine only a few times a day, so that plenty of calls can be made in between.

5.3.26 NLC-027 — Database Can Retain Data after Deletion

Vulnerability ID: NLC-027	**Status:** Unresolved
Vulnerability type: Privacy issue	
Threat level: Low	**Software version:** 1813b997e5ba7085abf1ea043a6d8b6020618aea

Description:

The Microsoft SQL database used in the background can retain data for longer than strictly required.

Technical description:

Using a database such as in this case MS SQL creates metadata and log files. Additionally, data is often just marked as free, but not immediately deleted from all positions in the database. Data might also be kept on the hard drive in unused blocks instead of being deleted.

This means that information such as the uploaded diagnosis keys and associated time values could be stored for longer than necessary.

This risk is inherent to these technologies. With some effort, however, data could be stored in such a way that it can be deleted securely.

Impact:

Data such as diagnosis keys, upload times etc. might be available for longer than required.

Recommendation:

Configure the database to not store metadata and securely delete data if possible.

Use a storage method that does not collect metadata and can wipe data securely.

6 Conclusion

Peer-review of the cryptographic basis

We analysed the Dutch Crypto Framework document, the cryptographic framework document of the Dutch COVID-19 notification app, which at the time of writing is still a work in progress.

The Dutch COVID-19 notification app is based on the GAEN (Google/Apple Exposure Notification system). GAEN implements a large portion of the cryptographic basis and is a partially closed-source component. GAEN has known attack paths and risks; Some of these risks and issues are inherent in the technology and there are many trade-offs in its approach, but some are implementation-dependent and can be improved upon. Outstanding risks of the GAEN system are not mitigated by the Dutch Crypto Framework document. However, the separate DPIA (Data Protection Impact Assessment) document contains an extensive list of risks and shortcomings, along with a mention of its mostly non-technical mitigations.

The Dutch Crypto Framework document follows the TEK life cycle, but we got the impression that the author was limited by a lack of information about the GAEN framework. While they seem to have tried to address this by asking relevant critical questions, the GAEN framework is nonetheless used as a partially closed-source, trusted third-party component. To the best of our knowledge, GAEN has not been completely independently verified, so the level of trust placed in it seems questionable.

The remainder of the Dutch Crypto Framework document describes how to protect key uploads and downloads. A clear choice has been made for a flow that allows for de-anonymization, at least theoretically, and makes it possible to link keys to an infected person. This approach was explicitly chosen in order to prevent non-infected people from uploading their keys. There is no transparency issue as the Dutch Crypto Framework document states this clearly. This potential coupling of patient information to key uploads is supposed to be temporary, but we note that it might be technically hard to actually strip all of this information away, and it may be correspondingly difficult to give citizens a plausible guarantee that this information is not linked. Meanwhile, some risk of malicious uploads of TEKs by non-infected people remains. GDPR's privacy by design principles recommend avoiding zero-sum trade-offs with privacy, but it's difficult to mitigate the risk of fake keys being uploaded without doing so.

The section about international key exchange in the Dutch Crypto Framework document does not elaborate on its possible implications, which leaves some open questions. The supporting documentation explains the intended implementation and describes additional privacy trade-offs related to the exposure of download size and countries visited.

Finally, we found that a number of points required for a secure solution were not defined in the Dutch Crypto Framework document, such as some key properties, information on the randomization of key upload batches, etc. We recommend extending the Dutch Crypto Framework document to elaborate on the points raised in this report.

Back-end code review

We reviewed several versions of the back-end code. The code is still work in progress and we do not expect this to be the final version – readers have to be aware that new code may introduce new bugs in components that have already been examined.

This evaluation is a code review only. It's essential to perform a penetration test on the deployed system to provide additional assurance on the security of the system as a whole, to check and verify deployment configuration, that the infrastructure is secure, and that the features promised by the Dutch Crypto Framework document are present and correct. During conversations with the developers we learned that HSMs are not yet used. We also learned that HTTPS is terminated at the load balancers; depending on the configuration, this could mean that these load balancers have access to the IP addresses, metadata and diagnosis key information. This would not limit access to the keys to the machines that absolutely need it. These are not things we can confirm in a code review, but they are of critical importance to the solution's overall security.

During the back-end code review we found 1 High, 1 Elevated, 11 Moderate, and 13 Low-severity findings.

Of which 1 High, 1 Elevated, 6 Moderate, and 7 Low-severity findings remain unresolved in the latest evaluated version.

The most severe issues are:

- The use of a third-party authentication provider, which, if compromised or acting maliciously, could allow attackers to assume the role of health authority employees able to authorize the upload of diagnosis keys. We recommend checking that this external service is regularly pen-tested, audited, and is trustworthy, or not using a third party for authentication at all.
- There is a feature that allows health authority employees who just authorized a key upload to check if the user actually uploaded their keys. Key upload is supposed to be voluntary; knowledge of whether it was done could be used legitimately for feedback, but also abused to e.g. coerce users and record which users (identifiable by name and phone number) uploaded their keys to help de-anonymize the keys.
- Decoy keys can be distinguished from real keys by their properties. This has been partially resolved and now only allows estimating the total number of uploaded valid keys.

All remaining issues are of moderate or lower severity and are mostly categorised as privacy issues where more data is collected than is strictly needed, or it is retained for longer than necessary, and authentication issues, where authentication processes for health authority employees can be hardened.

Given the sensitivity of this project, we recommend fixing all of the remaining findings and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Overall, the general design forms an adequate basis for a privacy-preserving COVID-19 tracking system, despite some deliberate trade-offs and undefined elements. However, the implementation is currently let down by numerous deviations from an optimal, secure, privacy-preserving implementation, and the design itself is some way adrift from the implemented solution; the two need to be kept in sync. Though the project's urgency is understandable, given its critical importance, a failure to address these issues appropriately could undermine it completely.

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

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

7 Bibliography

[1] Security Analysis of COVID-19 Contact Tracing Specifications. https://eprint.iacr.org/2020/428.pdf.

[2] GAEN Due Diligence: Verifying The Google/Apple COVID-19 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] Lab result validation flow. https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/Solution%20Architecture.md#lab-result-validation-flow commit 02d7fa69ef337298b448f1a82954a83223d90caa. Accessed: 2020-07-20.

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

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

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

[15] 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.

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

[17] 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.

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

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

[20] . https://tweakers.net/nieuws/125967/autoriteit-persoonsgegevens-gaat-vragen-stellen-over-wifi-tracking-op-ns-station.html. Accessed: 2020-07-18.

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

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

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

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

[25] . https://www.rijksoverheid.nl/regering/bewindspersonen/hugo-de-jonge/documenten/rapporten/2020/07/07/gegevensbeschermingseffectbeoordeling-dpia-covid-19-notificatie-app. Accessed: 2020-07-18.

[26] . https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/Solution%20Architecture.md#backend. Accessed: 2020-07-18.

[27] . https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/Federated-Gateway-1.00.pdf. Accessed: 2020-08-06.

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

[29] 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.

[30] Dutch COVID-19 Notification App cryptographic framework document. https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/Crypto Raamwerk.docx. Accessed: 2020-08-25.

[31] Design documents Dutch COVID-19 Notification App. https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/. Accessed: 2020-08-25.

[32] Back-end repository Dutch COVID-19 Notification App. https://github.com/minvws/nl-covid19-notification-app-backend. Accessed: 2020-08-25.

[33] Key Stuffing. https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/Linkability%20Mitigation%20With%20Stuffing.md. Accessed: 2020-08-25.

[34] International Interoperability. https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/International%20Interoperability.md. Accessed: 2020-08-25.

[35] Traffic analysis mitigation with decoy traffic. https://github.com/minvws/nl-covid19-notification-app-coordination/blob/master/architecture/Traffic%20Analysis%20Mitigation%20With%20Decoys.md. Accessed: 2020-08-25.

Appendix 1 Testing team

Tim Hummel	Tim Hummel is a senior IT-security analyst, consultant, developer and trainer. His speciality is hardware, crypto, and related software security. In his work he tests everything from apps, car components, payment solutions, white-box crypto, pay TV, mobile devices, IoT, TPMs, TEEs, bootloaders, entertainment systems to transport cards.
Tillmann Weidinger	Tillmann Weidinger is a student of Computer Science and full-stack developer with a strong emphasis on security. He has multiple years of experience in engineering, software architecture, and breaking things. Due to his curiosity he has a wide range of knowledge in different fields and is always happy to learn new topics.
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.

Front page image by Slava (https://secure.flickr.com/photos/slava/496607907/), "Mango HaX0ring",
Image styling by Patricia Piolon, https://creativecommons.org/licenses/by-sa/2.0/legalcode.