Security Report Proximity Scanning

Author: csirt@bit.admin.ch & outreach@govcert.ch
Topic: Security Report for Proximity Tracing Project – Appendix to the Risk Estimation
Date: 28th of May 2020
Classification: **TLP WHITE**

Content

Discussion of findings

Overview

The following appendix gives an overview of the various vulnerabilities we found and passed to the project for fixing.

The most important concerns we found were as mentioned in the risk estimation:

- Demasking a user based on traffic monitoring / DNS requests. This has been solved by using fake POST requests. However, the residual risk that remains is if a user enters the code incorrectly, a distinguishable traffic patterns gets visible.
- Man-in-the-middle attacks using TLS interception, either in transit or on CDN level which would allow an attacker to identify persons that send in the information that they are infected. This issue has been solved by implementing a certificate pinning. The suggested, additional end-to-end encryption has not been implemented.
- We found various small to medium vulnerabilities and bugs which have been addressed by the project. These are all documented in the following document.
- We have found various vulnerabilities in supporting systems that area not in the scope of the public security test. Most of these have been addressed by the respective system owners, some were fixed, other mitigated. For the system with most vulnerabilities, an internal project has been started for a complete code refactoring.

There are a few security concerns that lie outside the scope of this document but should be mentioned, nevertheless. One is the overall security of the smartphone, such as revealing the identity by the name of the device (“Max Muster’s iPhone) or by outdated OS versions with known vulnerabilities, especially in the Bluetooth stack. Another noteworthy risk are the devices of medical staff. If such a device gets infected, an attacker might generate authentication codes and could potentially flood the system with wrong infection data.

Protocol

CSIRT FOITT / GovCERT has reviewed the protocol from EPFL: It is very robust and respects privacy to a high degree. We have made various inputs to further strengthen the protocol, most of which have been received and implemented. The protocol has also been reviewed by various other groups. We are convinced that the risk at the protocol level is very low as long as the relevant framework conditions are taken into account during the implementation.

How we tested

Testing the protocol did happen only by reviewing the published specification and discussing it with Carmela Troncoso and her team.

Ensuring the privacy of seed uploads

The most serious issue we have found during our test reflects that in the first iteration, the low-cost protocol version has been implemented and that the environment was not properly considered.

- There were different DNS endpoints to route the traffic to the various URIs.
- Anyone controlling DNS servers (e.g. Google, Cloudflare but also ISPs or companies) can tell who queried a certain DNS at a certain time.
- Uploading a seed is involved with a known sequence of DNS requests and can therefore be detected by monitoring DNS alone.
- The same applies to an organization that can monitor the traffic, e.g., a company or an ISP.

This may lead to an unmasking of the user that uploads his/hers seed in the case of a positive test.

The protocol addressed this fact in the second variant by proposing the usage of dummy POST requests. We suggested implementing these into the current version which has been done by the project. An interesting question was, how often such POST request needs to be done to ensure anonymity without creating an overload on the backends. Both EPFL and GovCERT/CSIRT came to the conclusion that only few POST requests are needed to ensure a high probability that no one can differentiate a real infection POST request from fake requests.

Ensuring the privacy of end users distributing Bluetooth EphIDs

We looked at how the creation of the EphIDs is proposed by the protocol and did not find any issues as long as the pseudo random number generator is good and used correctly.

One inherent problem is the exposure of Bluetooth itself as this might lead to some degradation of the user's privacy, e.g. by exposing the name of the phone ("Mikes iPhone"). This however is neither the fault of the protocol nor the implementation.

The usage of Bluetooth is the best method in our opinion to have a commonly available protocol without sacrificing too much privacy, e.g. compared to using geolocation information from the phone. However, it must be clearly stated, that Bluetooth had and is going to have vulnerabilities which may expose a device. Especially worrisome are older, non-patchable Android devices.

Replay attacks in order to poison the system

Replay attack is the only real sabotage possibility we could find in the protocol: An attacker can collect EphIDs of people with a high probability of future positives with a very sensitive receiver, e.g., near a drive-in test center or a hospital in general, send these via internet to a very different location where a lot of non-infected people are expected (like in residential areas), and replay them there using a very strong Bluetooth signal. This would cause a lot of false detections.

Eavesdropping the EphIDs

An attacker who is in proximity of the victim can eavesdrop the EphIDs. Currently, no defense against that is implemented. EPFL proposed spreading the EphID across low-energy beacons using k-out-of-n secret sharing. However, this is not implemented, and it is not clear if this can be done using the Google/Apple APIs in the future.

Technical implementation architecture

We did analyze the technical architecture and had a few discussions about certain elements, especially about the usage of a CDN. Basically, three approaches were discussed:

1. All traffic passes a CDN.
2. Only (the privacy irrelevant) GET requests are passed through the CDN.
3. No CDN is used at all.

The project decided to go with the 2nd approach using the CDN for the GET requests only.

How we tested

We reviewed the architectural specifications / drawings together with the architects at FOITT.

Usage of CDN

The usage of a CDN (in that case Amazon AWS) has advantages and disadvantages:

- It makes the whole system much more resilient against overloads as well as DDoS attacks.
- It gives additional flexibility by easily adding resources upon need and at very short notice.
- It has some privacy drawbacks that are larger or smaller depending on the chosen implementation.
- It adds an additional layer of complexity.

In order to combat a DDoS attack efficiently, especially on the application layer, TLS termination must be in place on the CDN, which raised privacy issues, especially for the transfer of the POST requests of a user telling the system that he/she is infected and uploads the seeds.

End-to-end encryption of POST parameters

We proposed using a simple asymmetric encryption of all POST parameters in order to gain the following advantages:

- Being able to use a CDN in case of performance issues / DDoS attacks against the backend.
- Adding an additional layer of security if a problem with the certificate pinning should occur.

The project decided against this implementation in the first version but instead chose to rely on certificate pinning only. If the pinning is done properly this should not lead to an additional exposure of the data being transmitted as any TLS inspection would fail and lead to a termination of the TLS session.

Introduction to Code and Configuration Tests

How we tested

We chose a greybox approach whenever possible to ensure as much testing as possible in the shortest timeframe. While this approach requires having both code, documentation and services running to fully perform the planned activities, it ensures that the live system indeed matches the expected concept and protocol. Implementation and operational aspects are often error prone and thus security relevant, while being hard to audit from a blackbox perspective. Furthermore, the system's management is one of the key aspects where the public needs to have a high trust as the publication of the source code will not answer questions on how the code is actually installed and operated.

Backends – Code and Configuration

Black Backend

VULNERABILITY	EXPLANATION (IF NEEDED)	STATUS
UNHANDLED ERROR CONDITIONS	Unhandled error conditions lead to info leaking on https://www.codegen-d.bag.admin.ch showing the actual hostname.	Fixed
INFORMATION LEAKAGE	Calling the URL (https://codegen-service-a.bag.admin.ch/actuator/health) leads to information disclosure.	Fixed
EXTERNALLY HOSTED WEBSITE	www.covidcode.ch is hosted externally, changing Name Resolution does not follow the same authentication and authorization principles as for “normal” admin.ch domains	Open, migration is planned

Red Backend

VULNERABILITY	EXPLANATION (IF NEEDED)	STATUS

INFO LEAKAGE	Calling the URL https://www.pt-a.bfs.admin.ch/v1/config results in information leakage as the server responses behind the CDN show its actual hostname:
	`Response:`
	`404 Not Found: Requested route (REDACTED) does not exist.`
		Fixed

INFO LEAKAGE	Calling the URL https://d27mnw3eab46ei.cloudfront.net/v1/config results in information leakage as the server responses behind the CDN show its actual hostname: Response: `404 Not Found: Requested route (REDACTED) does not exist.`	Fixed
LARGE KEYS	When reporting keys (Post to /v1/exposed), a Key is normally 32 bytes long (b64 encoded 44 bytes). The endpoint allows for reporting keys with sizes up to 2000 bytes. If the key size is bigger then the request results in a database error. An attacker who is able to generate valid JWT or bypass the Token validation could generate large buckets which could lead to massive data consumptions on server and customer side. While this would be also possible with 32 byte keys, the number of requests needed for the same outcome would be 60 times higher. Overall the risk of someone abusing this technique is minimal as he needs to be able to generate a lot of valid JWT Tokens (or bypass the validity check) in a short amount of time. Recommendations Validate the key size before writing it to the database.	Fixed
EXTERNALLY ACCESSIBLE INTERNAL URL	The URL https://identity-a.bit.admin.ch/realms/bag-pts-test should only be accessible from within the federal network.	Fixed

Further tests without findings

JSON Web Token validation

The JWT validation is an essential part of the red backend. If the token validation can be bypassed or if a token can be use multiple times an attacker could send fake infections to the backend to create false positive infection alerts.

Multi-Use
To ensure that a JSON Web Token can only be use once, the backend stores the UUID present (and signed) in the token to the database. If the UUID is already present, the request is denied (401 Unauthorized). Additionally, the tokens are short lived and expire after a short timeframe.

Invalid signature
If the signature is invalid, the server responds with 401 Unauthorized and the reported key will not be stored in the database.

None algorithm
A well-known issue with JWT is the usage of "none" algorithm to bypass the signature check (https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries/)
Using https://github.com/ticarpi/jwt_tool we generated various requests and confirmed that the backend is not vulnerable to this attack. We did not find a way to bypass the signature check this way.

Fuzzing
We did a “fuzzing” of the backend to find potential undocumented accessible files/paths on the server without any unexpected results.

Source-Code analysis
While testing we did a source-code analysis of the backend, which did not yield in any new findings.

Smartphone App Code: iOS (pre-gaen version)

Version tested in this document: 200504.2152.338, commit: `457270a` (May 5, 2020). Note that several new updates occurred meanwhile (until May 8, 2020). According to comments, the changes are only affecting the Bluetooth proximity algorithm itself, which most probably don’t have an impact on these results, as this algorithm is not security relevant.

Cryptography and TLS

Note: As soon as the Apple API will become public, cryptographic functions will be done inside this API. So the remarks in this section only refer to the version used before this API is published.

- Random number generator used to generate EphIDs: The SDK calls https://developer.apple.com/documentation/security/1399291-secrandomcopybytes from the standard library, which, according to docs, generates an array of cryptographically secure random bytes.
- SHA256 (`CC_SHA256`), AES and HMAC are all used from the standard library as well and as such out of scope of this analysis and considered as correct.
- The code implements the protocol as described in the low cost design of DP3T.

As the project decided against an end-to-end encryption of the sensitive POST parameters, it was decided to do a *certificate pinning*:

- Pinning the certificate for the configuration downloads and the GET downloads to Quovadis Intermediate certificate.
- Pinning the certificate for the POST requests to the SubjectPublicKeyInfo.

Certificate pinning could be verified in our lab:

- When trying to be run over an SSL proxy (verified with Burp professional suite), no SSL connection happened.
- When certificate pinning was turned off in the code (https://github.com/DP-3T/dp3t-app-ios-ch/blob/develop/DP3TApp/Logic/Networking/Base/URLSession%2Bpinning.swift, setting useCertificatePinning to false on lines 27 and 29), the connections worked, which serves as a negative control test
- Certificate pinning worked for all types of requests.

One small issue was forwarded to Ubique; in case of a WLAN SSL proxy that breaks open SSL, the user gets a cryptic error message in case he tries to upload a key ("Fehler – Abgebrochen – UNKNW") which should probably be made clearer; it will probably be fixed in one of the next releases.

Storage

The **daily secret keys** used to generate the EphIDs are stored the iOS keychain as an array named `org.dpppt.keylist` of structures, each containing a date `day`, the key `keyData`, and a `description` string. iOS Keychain is considered as a secure and encrypted storage.

The **EphIDs for one day** are also stored in the iOS keychain as a structure `org.dpppt.ephsIDs` containing a date `day`, and an array `EphIDs` of data blobs. iOS Keychain is considered as a secure and encrypted storage.

Whenever a new list of EphIDs is created at UTC midnight, the previous list is overwritten. Daily secret keys older than 21 days are removed.

Contacts collected during normal operation (i.e. EphIDs broadcast by other devices and recorded by our device) are stored in an *SQLite database* table named `contacts` with columns `id`, `date`, `EphID`, `windowsCount` (representing the estimated contact time), and `associated_known_case` (initially set to `nil` and updated with an id in the `known_cases` table as soon as a match with an infected person was actually found).

The *timestamp of the contact is stored with the granularity of currently 2 hours*, see DP3TSDK/utils/ContactFactory.swift:

```
let bucketTimestamp = timestamp -
timestamp.truncatingRemainder(dividingBy:
Default.shared.parameters.networking.batchLength)
```

This is in a slight contradiction to the DP3T Whitepaper, which states:

> *Other smartphones observe these EphIDs and store them together with the duration and a coarse indication of time (e.g., "April 2").*

It can be argued whether a 2-hour granularity is coarse enough. The underlying problem is that a finer granularity makes it easier for a warned person to identify the positively tested person which might have infected her. However, this is a local problem, and the timestamp, while stored in the database and as such available to the user via a memory dump or backup, is not actually displayed by the app to the warned user. It can also be argued whether these timestamps should really be kept secret from the user to protect the privacy of the positively tested patient, or at the opposite made explicitly available to make her risk assessment easier. The granularity of the timestamp is basically a tradeoff decision between preserving as much privacy as possible vs giving the user precise information about a potential infection. For the GAEN version, this question became irrelevant as the time resolution is given by the API that is used.

This *contact table is not stored in a protected way*, but this is not considered as a serious problem, as only encrypted EphIDs are stored; however, as soon as the Apple API is available, this part of the data will most probably be stored in a more secure way.

Known cases downloaded from the backend are also stored in a local SQLite database called `known_cases` using columns id, `batchTimestamp`, onset, and key. This data is not stored in a protected way; but as this data is downloaded from the backend and public anyway, this is not considered as an issue.

Functional Issues (not security related)

Several small issues were found in the SDK and app code and reported in the git or via messenger app. These problems were solved immediately:

- The URL for reporting and the one for downloading keys were swapped (https://github.com/DP-3T/dp3t-sdk-ios/issues/53). Fixed.
- Timing for DB syncs (infected keys downloads) did not start properly, so keys were initially not downloaded at all (https://github.com/DP-3T/dp3t-sdk-ios/issues/51). Fixed.
- Cases where the onset date of a downloaded, infected key is older that the oldest stored contact were checked with a wrong day offset (https://github.com/DP-3T/dp3t-sdk-ios/issues/50). Fixed. The solution however leaves still a small window for missed contacts in the case subsequent days (after the oldest stored contact) will be within reach. However, the scenario is probably very rare as onset dates in downloaded keys older that 21 days should rarely happen (e.g. if a person did not have network connectivity for a long time).
- The timing algorithm for fake POSTs was originally randomized only once, which would have resulted in periodic fake POSTs instead of randomized ones. Fixed.
- The timing algorithm for fake POSTs was re-initialized every time the App was put into background, which might have caused timing of fake POSTs to not work as expected. Fixed.
- Certificate pinning could not be turned off in the code without making networking dysfunctional. Fixed in https://github.com/DP-3T/dp3t-app-ios-ch/commit/db3a455f842072536dceba2180b275d5b841255c

Background Tasks

Fake POSTs are used to make it harder for an observer to detect and de-anonymize key uploads of people with a positive test result. They are implemented by using background tasks in iOS (`BGProcessingTask`) which should trigger once all ~5 days. Unfortunately, the scheduling algorithm of Apple is not predictable, so it's not sure these tasks are activated as expected.

Moreover, as these tasks only occur when the device is idle, but real POSTs occur when the device is not idle, a network observer might be able to differentiate between fake and real

POSTs by just observing for the presence or absence of other network traffic caused by the device.

One still not completely transparent issue lies in the background tasks used to download new keys from the backend (DB sync) on one hand, and to initiate fake POSTs on the other hand:

- For *a DB sync*, a `BGAppRefresherTask` with ID `org.dpppt.synctask` is scheduled all 15 minutes. If at least 2 hours are passed since the last sync, one batch for each rounded 2-hour period is requested from the backend for DB sync. This syncing process seems to work well on one tested device, while on another, they only occurred after the app was taken into foreground. This behavior is bad as it prevents a user from being warned until she actually puts the app into foreground. Unfortunately, the device where these background tasks worked fine, dropped into the non-working behavior after rebooting it. We assume this problem might disappear as soon as the new Apple API is available, because they announced a prioritized version of it (https://developer.apple.com/documentation/exposurenotification/building_an_app_to_notify_users_of_covid-19_exposure) :

  > "The app uses a background task to periodically check whether the user may have been exposed to an individual with COVID-19. The app's Info.plist file declares a background task named `com.example.apple-samplecode.ExposureNotificationSampleApp.exposure-notification`. *The BackgroundTask framework detects apps that contain the Exposure Notification entitlement and a background task that ends in `exposure-notification`.* The operating system automatically launches these apps when they aren't running and guarantees them more background time to ensure that the app can test and report results promptly."

- For fake POSTs, a `BGProcessingTask` with ID `ch.admin.bag.dp3t.fakerequesttask` is scheduled using a Poisson distribution initiated on average once all 5 days. This time period was reduced to 1 hour for testing – however, it was not possible to actually observe such a fake POST in our test devices. This does not mean it does not work – but according to Ubique, the time scheduling for these background tasks can't be predicted. We consider this as a still open issue and there might be some feedback from Apple regarding the issue.

Old iOS devices

The current implementation requires at least iOS version 13. Also, the announced Apple API will only be available for iOS 13. Currently, this excludes devices older than 4 years – the oldest device able to run iOS 13 is the iPhone 6s and SE, while the standard iPhone 6 can only run iOS 12[1]. According to Comparis[2], this might affect around 15% of all iPhone users, but we don't have reliable data about this. It is unknown if maybe Apple will make the API also available for iOS 12, or make iOS 12 available for older devices.

JWT signature checks

Server responses carry a `Signature:` header, used as a JWT signature. We tried to modify a *configuration update response* such that an app update would be enforced, without changing JWT. This configuration check was not accepted, which suggests the JWT check works for configuration updates.

To repeat the test for *DB syncs* (`GET /v1/exposed` requests), a proximity situation of 2 iOS devices was simulated in the lab. One of the devices was then reported as infected. The DB sync request the other devices initiated at the next bucket time interval was intercepted and the JWT signature modified to enforce a signature check failure. As expected, no warning

[1] https://www.apple.com/ios/ios-13/
[2] https://www.comparis.ch/telecom/handy-gadgets/analyse/smartphone-markt-schweiz

was shown to the user, so it is assumed the JWT check also works for DB syncs. When the next DB sync went through unmodified, the warning appeared, as it should; this serves as a positive control check.

Code analysis of https://github.com/DP-3T/dp3t-sdk-ios/blob/master-alpha/Sources/DP3TSDK/utils/JWTVerification.swift shows that

- JWT signatures are checked for configuration updates as well as for DB syncs using the standard library `SwiftJWT`.
- The `accessToken` value returned by the codegen server as a validation response to a TAN is not verified by the app, though it could. But this is not a problem, as this value is subsequently sent to the publish server in an `Authorization: Bearer` header and should be verified at this place.
- Public keys for signature checks are hardcoded.

Smartphone App Code: Notes about iOS gaen version

We made a short comparison to the new version used since May 20 using the Apple API, based on commit 27301e2 (May 22 2020) of https://github.com/DP-3T/dp3t-app-ios-ch/tree/feature/exposurenotification-entitlement, and could verify (as expected):

- *Contact data* is no longer stored in a SQL database inside the app, but handled by the OS. Apps can't access this data directly anymore. We could not verify if the data would be available in a jailbroken device, but assume it would be.
- *BT communication* and encryption of normal EphIDs are now also done by the OS.
- Still done inside the app are *fake POSTs*; this is the only place the random number generator is still used

One general remark about the iOS situation is that the exposure API is *only available on iPhones, not on iPads* (which is not considered a problem). As expected, the API is only valid on devices that can load iOS 13.5 (iPhone 6s and upward).

Background tasks on one standard device we were using are now much more reliable; they occur regularly, also while the device is locked, and we could also observe fake POSTs. One strange behavior is that *warnings to the user* currently seem to occur *later than necessary*:

- Device A uploads diagnosis keys for the previous 3 days.
- Device B (which had contact with device A on all days before) downloads these keys a few hours later (around noon).
- Nevertheless, device B does not emit an infection warning until 2AM of the next day (Sunday), but then for all 3 days before. We did not start the app manually until then, but used device B for other stuff, so we can't say if that might have triggered a warning.

One reason for this might be that the number of API calls the app is allowed to make is restricted. It comes with a price of a delay up to 1 day, 12 hours on average. The issue was reported to Ubique.

A problem we observed generally with background tasks is that on our second test device (an iPhone SE) they did not occur at all; this device only communicated whenever it was unlocked and the app taken into foreground. Note that this is only about internet traffic (database syncs, fake POSTs), not the Bluetooth activity itself, which we assumed worked as expected. However, such a behavior would mean a user *is not warned for days* (until she manually takes the app into foreground). Currently we have no explanation for this behavior; it might be either due to special lab environment, or due to the fact this device did not have a SIM card, but only worked via WLAN. The situation will be evaluated further. It has no direct security impact, except the fact fake POSTs do not happen and so a real key upload can be found out easily.

Fake key uploads: These now happen in a clean way. Generally, we see the following timeline:

- Config file is requested.
- At nearly the same time 10 GET requests are done to download exposure keys (DB sync).
- With a random delay of 20-30 seconds, a fake TAN is generated and uploaded.
- And shortly after, the fake keys are uploaded. It is correctly made sure that fake and real key uploads have about the same size.
- Immediately afterwards, another database sync (10 GET requests) occur. This does not seem necessary, but is the same behavior as in the case of a real key upload.

This looks good and can't be easily distinguished from real key uploads. Two points must be noted though:

- The time delay of 20-30 seconds is pretty short for simulating entering a 12 digit TAN. This was implemented this way because Apple restricts the maximum duration of such background tasks.
- When in the future database sync tasks are minimized (they seem to occur much more often than required), one has to be careful to adjust the fake POSTs accordingly.
- Mistyped TANs in real uploads still stand out, due to the missing local validation digit. But this problem is not unique to the iOS app.

One strange feature about database syncs is the fact that batches now seem to be 24 hour batches instead of 2 hour batches as in the pre-gaen version, and these batches refer to validity of batches instead of publish date. The reason for this implementation lies in the nature of the API: it is required to feed it all exposure keys together for a specific day and then do a risk calculation day per day. This differs from the original dp3t approach. Also, no keys are buffered. As an upload can affect several earlier days, previous batches can be changed at a later point in time. Nevertheless, the sync request could be one single GET request that would return 10 days in one request; also a download sorted after publish dates would still be possible, if the downloads were locally stored in the app and sent to the API in a sorted manner. This observation does not affect security though, but is a question of network load versus local memory storage. Issue was discussed with Ubique and it is expected Apple will decrease some restrictions to solve it.

Smartphone App Code: Android (pre-gaen Version)

Android manifest

We looked at the manifest and found one issue, which has been corrected in the meantime:

VULNERABILITY	EXPLANATION (IF NEEDED)	STATUS
MANIFEST OF ANDROID APP ALLOWS BACKUP OF DATA WHICH MIGHT LEAD TO A PRIVACY ISSUE.	N/A	Fixed

The permissions are reasonable and reduced to what is necessary. Two permissions are especially noteworthy and are due to the way Bluetooth and the app interact:

- android.permission.ACCESS_FINE_LOCATION: Is needed as Google considers Bluetooth a location device as well. The app does not access GPS or location data.

- android.permission.BLUETOOTH_ADMIN: Is needed due to the way the app works. This permission may however also be used to pair devices. We do not see any such code, but basically as Bluetooth is on and working with this permission an attacker might send a Bluetooth pairing request. This is however not a vulnerability but due to the way the Bluetooth works.

Storage

There are no findings here: The app uses EncryptedSharedPreferences as suggested by google. See https://developer.android.com/topic/security/data. All Keys and EphIDs are stored within Objects of type EncryptedSharedPreferences. There are two containers, one for codes entered by the user and jwt tokens, one for EphIDs and keys (class CryptoModule). The list of EphIDs is automatically cut at specified length at each update.

Background tasks / Fake requests

There is a background task doing fake requests. Requests could be observed. The worker is implemented in the class FakeWorker, extending class android.work.Worker. Delay is calculated using class ExponentialDistribution. The requests did work as expected, also when the device was locked.

Database injections

Even if the data is stored in a SQLite database, we did not find any possibility to inject or manipulate the data being stored.

Informing about an infection:

We successfully entered a covidcode. The key was accepted and distributed by the red system. An app installed on an iPhone successfully displayed a warning regarding a possible infection. After that the app stopped broadcasting EphIDs.

Communication

Communication with the backend is implemented using retrofit2 (uses okHttp as backend) and okHttp “standalone”. We did not find any special issues.

HTTP headers seen where

```
okHttp 3.12.0
ch.admin.bag.dp3t;1.0;Android;29
```

Crypto

Testing https://pt1.bfs.admin.ch/v1/onset no inline signature (black backend), **certificate pinning did not work.**

Testing https://codegen-service.bag.admin.ch no inline signature (black backend), **certificate pinning did not work** (name missing @class CertificatePinning?).

Certificate **is only been partially effective**. Pinning did not work for requests targeting https://www.pt.bfs.admin.ch/config/v1. All other requests were interceptable. This can be tested by using an intercepting proxy and adding the proxy's certificate to the device under test. The debug build must be used as only that build accepts root certificates in users trust store, as defined in network_security_config.xml (for the prod release the proxy's certificate must be stored in the system's trust store). Disabling pinning of single backends can be done in the file CertificatePinner.java. CertificatePinner.java lists the names of different endpoints, however the names of the issuing CA's seem to be missing there. Check the list in CertificatePinner.java against what has been defined by the project.
Also specify procedures in case a certificate needs to be replaced immediately. Consider adding additional certificates as a reserve.

The url https://www.pt.bfs.admin.ch/v1/exposed is interceptable**, certificate pinning did not work.**

- Removing signature headers leads to an error, i.e., works as it should.
- When changing the signature the app displays error in red letters, i.e., works as it should.
- When changing the content th app displays error in red letters, i.e., works as it should.

Testing https://www.pt.bfs.admin.ch/v1/config

- When applying automatic update at proxy (replacing "forceUpdate" by "forceUpdate-modified"), the application does not start. Works as it should and certificate pinning seems to be ok.

This issue has been fixed in later releases.

VULNERABILITY	EXPLANATION (IF NEEDED)	STATUS
CERTIFICATE PINNING WORKING ONLY PARTIALLY	Certificate pinning is used to ensure end-to-end encryption of the messages being transported without an attacker being able to MITM by intercepting the TLS connection.	Fixed

Signatures

The **red backend** signs its body content in its responses. The signature is within a HTTP header named "Signature". The corresponding public key is stored in the app's source code.

Error Handling

Error handling has been implemented correctly in the last tested version. The first version lacked a proper error handling during the tests, but this has been corrected.

VULNERABILITY	EXPLANATION (IF NEEDED)	STATUS
ERROR MESSAGES WERE DISPLAYED UNFILTERED TO THE USER (E.G. ERROR MESSAGES FROM THE BACKEND)	N/A	Fixed

Timing / Dates

Testing in different time zones: No tests have been done as all timestamps are in UNIX epoch time format and therefore free from time zone information.

Reset / Delete / Reinstall

The applications data is stored in the directory /data/data/ch.admin.bag.dp3t. The directory and all its content are removed when the application is deinstalled.
Deleting and reinstalling the app did work flawlessly.

Data storage

Handshakes and contacts are stored within a SQLite database (/data/data/ch.admin.bag.sp3t/databases/dp3t_sdk.db). Handshakes are stored in a table named “handshakes”. After connecting to the database interactively we could observe that the handshakes (EphIDs with timestamps and signal strength indicator) are deleted as soon as the current epoch moves on.
Contacts are stored in the table “contacts”. After changing the phones system time (by adding 4 weeks) all entries in these tables were deleted as expected. There is also a table named known_cases that is unencrypted but as already mentioned above this should be no problem as the information stored within that table is public anyway.

Smartphone App Code: Notes about Android gaen version

Using the gaen API of Google, some functionality is now implemented within the Exposure Notification Module, which is distributed to the phones by Google play services. We did a short analysis of some changes that had been introduced with the gaen Android version of the app. Noteworthy results are listed below:

Android manifest

The app now only uses the following two permissions:

- android.permission.BLUETOOTH: Is needed to use the gaen API
- android.permission.INTERNET: Is needed for backend communications

ACCESS_FINE_LOCATION and BLUETOOTH_ADMIN are not needed any more.

Storage

The keys and EphIDs are now stored within the following (implementation specific) directories in LevelDB databases:

- /data/data/com.google.android.gms/app_contact-tracing-contact-record-db/ (contacts)
- /data/data/com.google.android.gms/app_contact-tracing-self-tracing-key-db/ (keys)

Only the keys are readable to the health authority applications and only by using the API. According to Google the API will not be available to all other apps. By using a jailbroken phone, it was possible to extract keys from the database. Depending on the Android version and configuration those databases can be encrypted at a file or disk level.

As those locations are part of the Google play services app. We also tested whether those databases could intentionally or accidentally get backed up to Google Drive, which might lead to a privacy issue. Using the phones user interface and the command line interface (bmgr), it was not possible to backup those contents.

To delete the databases the user has two options:

- Deleting ALL health authority apps that are installed on the phone. The data will be removed during the deinstallation of the last app using the API.
- Using the GUI of play services by pressing a button

This could lead to some confusions: Should a user have installed multiple health authority apps, by removing one of them he could mistakenly believe he had deleted all data although the data would still be there.

Databases

The database holding keys and EphIDs has been removed from the app.

Bluetooth

The whole Bluetooth functionality is now handled by the Exposure Notification Module.

Key export, automatically deleting older data

When exporting the keys to the app, the user is asked for confirmation by a dialog. The dialog mentions a maximum key range of 14 days. When testing this functionality by setting new dates manually, we observed that it was possible to export more than 14 keys with a time range of more than 8 weeks. At the moment of this writing we do not know if during normal operations those keys would have been deleted. However, both the Exposure Notification Module and the app should make sure it is not possible to export and save more keys and EphIDs than absolutely necessary.

Keycloak server

VULNERABILITY	EXPLANATION (IF NEEDED)	STATUS
EXPOSED ADMIN INTERFACE OF KEYCLOAK SERVER	N/A	Fixed
WEAK PASSWORD ON KEYCLOAK SERVER	N/A	Fixed

Domain Security

DNSSEC

While DNSSEC is in place for admin.ch, there are endpoints that are not protected:

DOMAIN	DNSSEC	REMARK
***.ADMIN.CH**	Yes	-
***CLOUDFRONT.NET**	No	We consider this being a risk as it is an entry point to the whole system and if it is manipulated, an attacker has several attack vectors at hand.
SWISSPTAPP.CH	No	We consider this being a risk as it is an entry point to the whole system and if it is manipulated, an attacker has several attack vectors at hand.
COVIDCODE.CH	YES	-
***HIN.CH**	No	The implementation is currently planned by HIN and should be ready in Q3 2020.

TLS Checks of the endpoints

We tested all endpoints if the TLS settings are reasonable. We tested both by hand and using ssllabs.com. Status is being considered OK, if Grade A is reached and an officially used Certificate of the Federal Administration is being used.

ENDPOINT URL	STATUS	REMARK
CODEGEN-SERVICE-D.BAG.ADMIN.CH	OK	
CODEGEN-SERVICE-T.BAG.ADMIN.CH	OK	
CODEGEN-SERVICE.BAG.ADMIN.CH	OK	
IDENTITY-A.BIT.ADMIN.CH	OK	
IDENTITY-R.BIT.ADMIN.CH	OK	
IDENTITY.BIT.ADMIN.CH	OK	
WWW.COVIDCODE-A.ADMIN.CH	OK	
WWW.COVIDCODE-D.ADMIN.CH	OK	
WWW.COVIDCODE-T.ADMIN.CH	OK	
WWW.COVIDCODE.ADMIN.CH	OK	
WWW.PT-A.BFS.ADMIN.CH	OK	
WWW.PT-D.BFS.ADMIN.CH	OK	
WWW.PT-T.BFS.ADMIN.CH	OK	
WWW.PT.BFS.ADMIN.CH	OK	
WWW.PT1-A.BFS.ADMIN.CH	OK	
WWW.PT1-D.BFS.ADMIN.CH	OK	
WWW.PT1-T.BFS.ADMIN.CH	OK	
WWW.PT1.BFS.ADMIN.CH	OK	
WWW.COVIDCODE.CH	Weak Ciphers	TLS 1.0 still supported, CA not used by Federal Administration
WWW.SWISSPTAPP.CH	Unknown	DNS ServFail

Security Headers on endpoints

For API endpoints we tested if this minimal set of headers is provided by the server:

- Strict Transport Security: max-age=XXXXX; includeSubDomains
- X-Content-Type-Options: nosniff
- X-Frame-Options: DENY

Red Backend

ENDPOINT URL	STATUS
HTTPS://WWW.PT-D.BFS.ADMIN.CH/V1	OK
HTTPS://WWW.PT-A.BFS.ADMIN.CH/V1	OK
HTTPS://WWW.PT-T.BFS.ADMIN.CH/V1	OK
HTTPS://WWW.PT.BFS.ADMIN.CH/V1	OK
HTTPS://WWW.PT1-D.BFS.ADMIN.CH/	OK
HTTPS://WWW.PT1-A.BFS.ADMIN.CH/	OK
HTTPS://WWW.PT1-T.BFS.ADMIN.CH/	OK
HTTPS://WWW.PT1.BFS.ADMIN.CH/	OK

Black Backend

ENDPOINT URL	STATUS
HTTPS://WWW.COVIDCODE-D.ADMIN.CH/	OK
HTTPS://WWW.COVIDCODE-A.ADMIN.CH/	OK
HTTPS://WWW.COVIDCODE-T.ADMIN.CH/	OK
HTTPS://WWW.COVIDCODE.ADMIN.CH/	OK
HTTPS://CODEGEN-SERVICE-D.BAG.ADMIN.CH/	OK
HTTPS://CODEGEN-SERVICE-A.BAG.ADMIN.CH/	OK
HTTPS://CODEGEN-SERVICE-T.BAG.ADMIN.CH/	OK
HTTPS://CODEGEN-SERVICE.BAG.ADMIN.CH/	OK

Keycloak

ENDPOINT URL	STATUS
HTTPS://IDENTITY-R.BIT.ADMIN.CH/REALMS/BAG-PTS/.WELL-KNOWN/OPENID-CONFIGURATION	NOK (X-Content-Type-Options / X-Frame-Options missing)
HTTPS://IDENTITY-A.BIT.ADMIN.CH/REALMS/BAG-PTS/.WELL-KNOWN/OPENID-CONFIGURATION	NOK (X-Content-Type-Options / X-Frame-Options missing)
HTTPS://IDENTITY.BIT.ADMIN.CH/REALMS/BAG-PTS/.WELL-KNOWN/OPENID-CONFIGURATION	NOK (X-Content-Type-Options / X-Frame-Options missing)
HTTPS://IDENTITY-R.BIT.ADMIN.CH/REALMS/BAG-PTS	NOK (X-Content-Type-Options / X-Frame-Options missing)
HTTPS://IDENTITY-A.BIT.ADMIN.CH/REALMS/BAG-PTS	NOK (X-Content-Type-Options / X-Frame-Options missing)
HTTPS://IDENTITY.BIT.ADMIN.CH/REALMS/BAG-PTS	NOK (X-Content-Type-Options / X-Frame-Options missing)

Out of Scope Threats

In our opinion, one of the greatest threats to the overall system does not arise from the components used, for which we have not identified any serious vulnerabilities, but from vulnerable end-user devices.

Supporting systems

During the tests, we found several issues in supporting systems that were not within the scope of the test. We reported all issues to the responsible parties. Most of the findings have already been fixed or are in the process of being fixed.

Mobile devices

A mobile device can become vulnerable through weaknesses or vulnerabilities in the Bluetooth stack, this is especially true for Android devices due to the large number of manufacturer-specific modes. If Bluetooth is switched on permanently, this leads to an increased exposure of the mobile device.

A device infected with malware can also cause data to flow off. However, the structure of the protocol does not demask the users involved.

The disclosure of personal information such as device names in the style of "Max Muster's iPhone" also leads to a potential unmasking of the user if only a few other Bluetooth devices are nearby. However, this is not the fault of the Proximity App and would also lead to the user being unmasked if he or she has activated Bluetooth without the Proximity App.

Devices of medical stuff

An infection of a medical person's device that can issue Authcodes would allow an attacker to issue false codes and thus flood the system with false infection messages. Infections of medical devices are in principle very sensitive, on the one hand because of the medical data stored on them, on the other hand because of the enormous damage potential in relation to a targeted attack with a lateral movement and the subsequent distribution of ransomware. We consider the medical personnel's end devices to be "out of scope" for this test but would like to point out the considerable risk in this area.

incidents@govcert.ch
Digital unterschrieben von incidents@govcert.ch
Datum: 2020.06.02 22:34:37 +02'00'