REVIEW FACILITY.EU

Name: NGI Emergency Tech Review Facility In-depth assessment European Federation Gateway Service

Contract nr: LC-04199045

Date of signature: 08-02-2021

Name of the deliverable: Deliverable 3.5 – In-depth assessment European Federation Gateway Service

Authors: Tim Hummel, Stefan Marsiske, Marcus Bointon (Radically Open Security)

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

Confidential: Yes

EUROPEAN COMMISSION
DIRECTORATE-GENERAL FOR HEALTH AND FOOD SAFETY

Resource management and better regulation
Information systems

INFORMATION ON THE EUROPEAN FEDERATION GATEWAY SERVICE DESIGN AND SOURCE CODE EVALAUATION REPORT

The Directorate-General for Health and Food Safety and the Directorate-General for Communications Networks, Content and Technology as co-System Owners of the European Federation Gateway Service solution inform that:

- The Design and Source Code Evaluation performed by Radically Open Security B.V. addressed the EFGS code version retrieved from the public repository on GitHub up to and including commit 5d0b2b8c8528c5268fbd5ada2f34e8577e1bef35. This snapshot is between release version 1.0.0-rc5 and 1.0.1-rc1.
- The finding CLN-002 in the report, has been addressed at the deployment level, replacing this code with a different secure solution for password storage.
- There is no other comment in any other findings of the report.

Contact: SANTE-EFGS-OPERATIONS@ec.europa.eu

European Federation Gateway Service Design and Source Code Evaluation

European Commission -
Directorate General CONNECT

V 1.0
Diemen, December 3rd, 2020
Confidential

Document Properties

Client	European Commission - Directorate General CONNECT
Title	European Federation Gateway Service Design and Source Code Evaluation
Target	The public source code and documentation of the European Federation Gateway Service
Version	1.0
Pentesters	Tim Hummel, Stefan Marsiske
Authors	Tim Hummel, Stefan Marsiske, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	October 3rd, 2020	Tim Hummel	Initial draft
0.2	October 12th, 2020	Stefan Marsiske	Added note on the architecture
0.3	October 15th, 2020	Tim Hummel	Report review
1.0	December 3rd, 2020	Marcus Bointon	Review

Contact

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

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

Table of Contents

1 Executive Summary

1.1 Introduction

Between October 1, 2020 and October 19, 2020, Radically Open Security B.V. carried out a design and source code evaluation for European Commission - Directorate General CONNECT.

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

1.2 Scope of Work

The scope of this evaluation was limited to a documentation and code review of the following target:

Source code and documentation of the European Federation Gateway Service retrieved from the public repository on github [1] up to and including commit `5d0b2b8c8528c5268fbd5ada2f34e8577e1bef35`.

- The scope ONLY includes the public available repository. We did not examine any potentially existing non-public developer repositories or take non-public documentation into account.
- The scope does NOT include a review of the test code in the repository's "\src\test" path.
- The scope does NOT include a penetration test of the actual infrastructure such as load balancers or reverse proxies and their configuration.

1.3 Project objectives

There were two objectives for this evaluation:

Verify Software Design European Federation Gateway Service: We reviewed if the design mentioned in the documentation in the github repository is in line with the EU document eHealth Network Guideline on "Interoperability specifications for cross-border transmission chains between approved apps" [5] and checked if the design and security requirements make sense.

Code review of the java code: We reviewed the Java code in the "\src\main" path. We checked the code for general security issues, potential vulnerabilities, that e.g. allow taking control of the service. We checked if the software is in line with the design.

1.4 Timeline

The Security Audit took place between October 1, 2020 and October 19, 2020.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 Low-severity issue and 9 observations.

The only discovered issue is minor. We did not discover any issues that would make us advise against using the provided application. Overall we had a good impression of the code quality of this project.

We however feel that the solution is implemented in an overly complex architecture. Minimization can reduce the Trusted Computing Base (TCB), and decentralization can prevent a single point of failure.

We strongly recommend performing an additional pentest and configuration review of the infrastructure to provide further assurance.

1.6 Summary of Findings

ID	Type	Description	Threat level
CLN-002	Credential management	A config file in the repository contains keystore passwords	Low

1.7 Summary of Recommendations

ID	Type	Recommendation
CLN-002	Credential management	• Consider storing the private key in a HSM instead of a keystore. • Do not store hard-coded passwords in a public repository. At least store them in the server environment.

2 Abbreviations and Terms Used in This Document

Abbreviation/Term	Definition
API	Application programming interface
GAEN	Google/Apple Exposure Notification (system)
EFGS	European Federation Gateway Service
TCB	Trusted Computing Base

3 Analysis

3.1 Design

By design this solution contains minimal assets. The goal is the handling of diagnosis keys for the individual countries. Diagnosis key upload and download are protected by secondary assets such as the allow-lists, database encryption, trust anchor certificates, and key-pairs for callbacks. None of these assets hold great value for an attacker if made public. Most relevant is the integrity for these assets and the integrity and authenticity of diagnosis keys.

The design fulfills its purpose, but this could have been also achieved in cheaper ways with a smaller TCB, see our note on this in section 3.2.1 (page 8).

Not all parts of the solutions are part of the source code. In particular the user authentication for API access is out-sourced to the load balancers. It is the load balancer's responsibility to check if a client certificate is valid and if a user really possesses the matching private key. Valid requests are forwarded to the EFGS service, which checks it against an allow-list, but does not check if the user has the matching the private key. For key upload an additional check is required for which the EFGS service itself checks the signature.

It is important that this infrastructure, including the load-balancer, is configured correctly.

On-boarding countries is, by design, a manual process. The on-boarding operator has follow a procedure and manually add the country to the allow-list. It is vital that this on-boarding process is performed with the utmost care, so as to prevent malicious parties uploading their keys or gaining access to the system.

3.2 Source code

Overall we have a positive impression of the code. In the parts of code we looked at we find:

- Incoming parameters are validated appropriately.
- Many test cases for all kinds of fault scenarios.
- Use of a language and framework that makes typical vulnerabilities such as buffer overflows and SQL injection unlikely.
- Use of stable crypto libraries rather than implementing their own custom crypto.
- Well-commented code e.g functions are explained clearly in comments or OpenAPI.
- No extraneous or unexpected additional components, which might have unnecessarily expanded the attack surface.

All EFGS APIs, except one with no security impact, are only accessible when authenticated. Only signed-up countries are supposed to be allowed to access these APIs, which severely limits the attack risk and potential. An attacker would either have to find a way to bypass authentication or get hold of the private keys of a country. A malicious user would have to be authenticated to abuse these APIs in order to e.g. to submit crafted malicious data.

We succeeded in running the test container as described in the documentation. A minor improvement for making testing easier and running the application smoothly for interested parties could be example certificates, example certificate insert statements, and prepared requests for testing the application.

3.2.1 A Note on the Architecture of EFGS

Summary: The architecture is very complex and unnecessarily expensive.

The EFGS is essentially a centralized service which collects the keys of infected people from each cooperating national healthcare service and shares all collected information with all cooperating healthcare services. In the words of the architecture document[5]:

> Each national backend uploads the keys of newly infected citizens (“diagnosis keys”) every couple of hours and downloads the diagnosis keys from the other countries participating in this scheme. That’s it. Data conversion and filtering is done in the national backends.

The problem as stated above is a very simple one, and it has been solved countless times in the past. The architecture document proposes the following:

> The least complex and most robust way to connect the backends behind all the different national proximity tracing apps is a Federation Gateway Service, which accepts diagnosis keys from all countries, buffers them temporarily, and provides them for all countries to be downloaded. Additionally, all backends can be informed immediately if new data is available, so that transmission lags are kept minimal. In this document, we propose a definite ready-to-implement architecture of the Federation Gateway Service.

This sounds reasonable. It is a store-and-forward broadcast pull messaging architecture with optional new-data-available notifications, quite similar to email. The architecture document estimates the worst-case upload data traffic from each participating country to be around 20-30MB per day:

> The amount of data uploaded by each backend server is comparatively miniscule; we’re talking about 20-30 MB per day at most (compare section 5.5). Additionally, the number of participants is restricted, since each country operates only one backend.

This paragraph continues:

> It follows that a small web service, equipped with a simple load balancer and replicated storage to ensure high availability, is enough to meet the demand in even the most unwelcome pandemic scenarios.

It is unclear why it follows that a web service is necessary at all to solve this problem, and a load balancer also does not seem at all necessary handling worst-case 30MB per country per day. How much would the downloaded data be in the

most unrealistic worst case? The following paragraph estimates about 390MB in case all european and russian citizens (in total 750million) use a GAEN app and the daily infection rate is 0.01%:

> We estimate the amount of data uploaded to the Federation Gateway Service during a 24-hour period, assuming a very bad pandemic situation and complete pan-European participation in this scheme. The basis of our estimate is the upload size of a single key including metadata, which is less than 200 bytes.

> Each currently infected user uploads one key, while a newly infected user uploads up to 14 daily keys of the past two weeks. Hence, we need the current number of infections (say, 1M - the total cumulative number of reported infections in Europe and Russia, as of June 2020, is less than 2.5M) and the rate of daily new infections (say,$0.01\% = 10^{-4}$, which is large). Let's assume the European population at 750M and virtually complete app adoption. This gives$14 * 10^{-4} * 750 * 10^{6} = 1.05 * 10^{6}$ new diagnosis keys and 1M other diagnosis keys per day, summing up to roughly 2.05M keys in total. Consequently, the Federation Gateway Service receives$2.05 * 10^{6} * 200 \text{ bytes} \approx 390 \text{ MB}$ per day, most of which has to be downloaded by each participating country.

This is a high upper bound with unrealistic adoption rate, possibly off by an order of magnitude from a realistic estimate. The architecture document also gives a lower estimate:

> In theory, higher values are possible. This is a pragmatic upper bound; we expect much lower values in practice. Factoring app adoption rates below 75% and significantly lower infection rates than assumed above, daily volume won't exceed 100 MB. Moreover, the precise numbers vary somewhat, depending on formatting, frameworks, header compression, batch size, and other technical details.

The chosen solution that is implemented contains the following components that are also part of the TCB:

- Load balanacer (F5)
- HTTP server/proxy (Nginx)
- REST API implementation (Tomcat)
- Database backend (MySQL)

The architecture document mentions two alternatives to this design:

> If a single European Federation Gateway Service, run in a suitable cloud environment, cannot be agreed upon for political reasons, the Federation Gateway Service can also be implemented in a distributed fashion using either of two different technologies: mirroring or a blockchain.

> Both technologies offer neither better performance nor more security,

It is unclear what kind of security is meant in this context.

It is also questionable what kind of performance is meant here? Considering the numerous components in the proposed solution, it can be expected to be slower than e.g. mirroring. If performance is also measured in cpu cycles spent then there are much more efficient solutions available.

> and they're both adding an additional layer of complexity.

The usage of blockchains possibly does, but other alternative solutions could be much less complex than the solution implemented.

> Nevertheless, a storage solution that is distributed across several or all participating countries may be the preferred solution for some policymakers.

Indeed such a solution would be much more resistant than a centralized solution. And remember that the participating member states have to implement an converter from their own format to the interoperable format to be exchanged anyway.

The architecture document dismisses the blockchain alternative:

> Nevertheless, blockchain technology arouses as much doubt and criticism in some as it produces enthusiasm in others.

It takes a lot more text to give an unconvincing argument against mirroring.

Alternative and possibly more efficient solutions include (considering each country provides a converter to a canonical interoperability format anyways):

- Signed emails broadcast to a mailing list
- XMPP messaging protocol
- RSS feed
- RabbitMQ
- bittorrrent distribution
- MQTT push updates
- Downloading the daily updates directly from the endpoints that the apps running on phones use, e.g. the German data is available on this url: `curl https://svc90.main.px.t-online.de/version/v1/diagnosis-keys/country/DE/date/2020-10-05`

Conclusion

In conclusion the architecture proposed is a reasonable one, however the proposed implementation based on multiple components, custom implemented REST service etc. is much more complex than necessary. The trusted components TCB could be reduced, reducing the attack surface on this service. The centralized architecture offers one single point

of failure. However we have to concede that the implementation itself is sound if we ignore the fact that the framework in which it exists is over-engineered.

4 Findings

We have identified the following issues:

4.1 CLN-002 — Repository Contains Passwords

Vulnerability ID: CLN-002

Vulnerability type: Credential management

Threat level: Low

Description:

A config file in the repository contains keystore passwords

Technical description:

According to the documentation the EFGS contains the following secrets in keystores:

```
private key of for outgoing TLS connections (for call back), to allow mTLS authentication
public key of Trust Anchor
```

Storing passwords in a public repository is not good practice.

The `application.yml` file contains the passwords for these keystores. It might be that these are only development values, but they are not clearly marked as such. There is a separate `application-dev.yml` in which is intended for development values.

If an attacker gets hold of the keystore files, they can use this knowledge to open the keystores. For the EFGS Trust Anchor's public key this is not an issue, because it does not need protection, but for the private key for outgoing TLS connections this is a concern. However, its impact is limited, because this key would only allow an attacker to inform countries' back-ends of new keys.

Impact:

The key in the config file can be used to open the keystore files, though this would first require a compromise of the keystore file. The attacker would then gain the ability to inform countries' backends of new keys and abuse it for DOS.

Recommendation:

- Consider storing the private key in a HSM instead of a keystore.

- Do not store hard-coded passwords in a public repository. At least store them in the server environment.

5 Non-Findings

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

5.1 NF-001 — Disclosure Policy Is Vague

The repository contains a `security.md` file. It is positive to see that instructions for secure disclosure exist.

It states:

```
Please do not report security vulnerabilities directly on GitHub. GitHub Issues can be publicly seen
 and therefore would result in a direct disclosure.

Please address questions about data privacy, security concepts, and other media requests to the
 cert@telekom.de mailbox.
```

This leaves it up to interpretation for the reader where to report security issues to. Is it `cert@telekom.de` or perhaps `opensource@telekom.de`, the owner of the repository? We recommend giving a clear contact including a way to report issues confidentiality protected e.g. via PGP, and suggest consulting disclosure policies from other projects for further guidance.

5.2 NF-003 — Authentication Relies on Load Balancer

Calls to the EFGS server have to be authenticated. They pass trough a load balancer which is supposed to check and terminate the client certificate based TLS authentication. The load balancer has to forward the client certificate information `X-SSL-Client-SHA256` and `X-SSL-Client-DN` as headers to the EFGS server.

The EFGS server then checks if these values are in its allow-list.

If anyone can send a request to the EFGS server and can control these two header values, authentication can be bypassed by just providing any valid country's values. This does not requires knowledge of the private key, because only the authentication in the load balancer requires knowledge of the private key.

The deployers have to make sure that the EFGS is not reachable via public routes other than the load balancer. They have to make sure that the load balancer does not allow additional or freely configurable `X-SSL-Client-SHA256` and `X-SSL-Client-DN` header values.

We recommended testing this during a penetration test or infrastructure review. We also recommend adding code that verifies that there is only one `X-SSL-Client-SHA256` and `X-SSL-Client-DN` header in each request and that the origin of the request is the load balancer.

Note that upload of keys is additionally protected by a check inside the EFGS service, which requires knowledge of a country's private signing key.

5.3 NF-004 — All but One API Endpoint Are Authenticated

All EFGS APIs are only accessible when authenticated, except one that has no security impact.

The exception is `/diagnosiskeys`. This API endpoint is harmless because it accepts no user data and only responds with hard-coded data. This API endpoint could potentially be removed.

5.4 NF-005 — Documentation Is Marked as Work in Progress

The main design document "Software Design European Federation Gateway Service" in the repository states *This document is not finished and major aspects are missing. This document is still in proposal state, meaning feedback is welcome and will change its content.*

This is surprising given that the software is already beyond version 1.0. We recommend finishing this documentation, which includes the security requirements.

5.5 NF-006 — Unused Variable CertWhitelist Implies an Additional White List Feature

The file EfgsProperties.java contains:

```
public static class CertAuth {
```

```
private final HeaderFields headerFields = new HeaderFields();
private List<String> certWhitelist;
```

The variable `certWhitelist` is never used, so this is not an active issue, but does hint at a potential allow-listing feature. This might refer to the ordinary allow-listing via database entries, or a short-cut left over from testing. We mark this as an observation, because any additional allow-list entries beyond the documented approach might represent a security risk.

5.6 NF-007 — SecReq-006 Might Allow Self-signed CA Certificates

```
SecReq-006 The Load Balancer MUST maintain a bundle containing the root CA certificates or
 intermediate
CA certificates needed to verify (trust) the clients' authentication certificates. If a national
 backend
uses a self-signed client authentication certificate, this certificate MUST be added to the CA
 bundle.
```

This requirement does not define which self-signed certificates under which criteria are allowed in this CA bundle.

A country could submit a self-signed certificate to be included in this bundle; But would this self-signed certificate be allowed to be a CA certificate or only a non-CA certificate? This should be clearly defined.

If it is allowed to be a CA certificate, this country could issue certificates imitating other countries. However this would only allow bypassing the load balancer authentication, not the allow-list in the EFGS itself.

This is an observation because the process to add certificates is outside the scope of the source code.

5.7 NF-008 — Google Apple Notification Framework Is the Basis

The EFGS is built for countries using the Google Apple Notification framework (GAEN) according to [4]

GAEN has known attack paths and risks. Some of these risks and issues are inherent in the contact tracing technology and there are many trade-offs in its approach, but some are implementation-dependent and could be improved upon.

For further information on this topic consider reading our report on the Dutch Coronamelder app [4] or analyses from other researchers [3] and [2].

5.8 NF-009 — Timing Oracle in Hash Comparison

The function `verifyThumbprintMatchesCertificate` in the file `CertificateService.java` contains a timing-oracle in a hash comparison

```
return certHash != null && certHash.equals(certificateEntity.getThumbprint());
```

This is not best practice, and can create security vulnerabilities, but does not in the given use case.

5.9 NF-010 — Database Encryption

There is database encryption for diagnosis key values. There are multiple observation regarding this feature:

1. The design in [1] does not mention database encryption. We recommend updating the design to include this.
2. Strings are used for the storage of the password

   ```
   String dbEncryptionPassword = System.getenv().containsKey(PASSWORD_PROPERTY_NAME)
   ? System.getenv(PASSWORD_PROPERTY_NAME)
   : System.getProperty(PASSWORD_PROPERTY_NAME);
   ```

   This is not recommended, because strings are immutable and cannot explicitly be cleared from memory after use.

3. The IV is hard-coded in the source code:

```
private IvParameterSpec getInitializationVector() {
  return new IvParameterSpec("WnU2IQhlAAN@bK~L".getBytes(charset));
}
```

   In special chosen plain-text scenarios this can be used to gain insight into the encrypted values or craft values. We recommend generating a new random IV for encryption.

4. The diagnosis key encryption is practically pointless – diagnosis keys are typically public anyway. Diagnosis keys are distributed to the countries' backends which then distribute them to their apps. These apps and their data can be accessed by almost anyone. This is probably implemented to comply with the statement in [5] chapter "8.2 Data Privacy": "Encryption at rest in the database".

We do not mark these points as issue, because the diagnosis keys are public knowledge anyway.

6 Future Work

- **Penetration test / Infrastructure review**
  Some security aspects cannot be verified by static analysis of files on GitHub alone. In particular, the solution depends on the proper configuration of the infrastructure, which should not make the EFGS accessible without passing through the load balancer. The proper configuration of the load balancer is essential, as it verifies the authentication of requests to the EFGS. We recommend performing a penetration test or configuration check on the deployment.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

7 Conclusion

We discovered 1 Low-severity issue during this evaluation and 9 general observations. The only discovered issues is minor.

The solution to the general problem of sharing Diagnosis Keys across EU countries could have been implemented in very many different ways, and many of those are possibly cheaper and built with less TCB components than the one under investigation in this report. The centralized architecture represents a single point of failure when compared to a distributed solution.

We recommend that the developers have a look through this report and consider our recommendations. We did not discover any issues that would make us advise against using the provided application. Overall we had a good impression of the code quality of this project. Our observations show that some improvements can be made, but these observations did not lead to security vulnerabilities.

We strongly recommend performing a pentest and configuration review of the infrastructure to provide further assurance.

We want to emphasize that security is a process – this evaluation is just a one-time snapshot. Security must be continuously evaluated and improved.

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

8 Bibliography

[1] EU Federation Gateway Service Github repository. https://github.com/eu-federation-gateway-service/efgs-federation-gateway. Accessed: 2020-10-05.

[2] DP3T White Paper. https://github.com/DP-3T/documents/raw/master/DP3T%20White%20Paper.pdf. Accessed: 2020-10-05.

[3] Security Analysis of COVID-19 Contact Tracing Specifications. https://eprint.iacr.org/2020/428.pdf. Accessed: 2020-10-05.

[4] Cryptographic Framework andBack-end Security Evaluation Dutch COVID-19 Notification App. https://raw.githubusercontent.com/minvws/nl-covid19-notification-app-coordination/master/privacy/Duidingsrapportage/Bijlage%20I%20-%20Codereview%20Radically%20Open%20Security.pdf. Accessed: 2020-10-05.

[5] Interoperability specifications for cross-border transmission chains between approved apps. https://ec.europa.eu/health/sites/health/files/ehealth/docs/mobileapps_interoperabilitydetailedelements_en.pdf. Accessed: 2020-10-05.

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.
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.