WHO DDCCG Security Review Report

Security Review Report

T-Systems International GmbH
PU Digital Solutions

Version:	1.0	Valid from	11.03.2022
Status:	Final	Review Date:	11.03.2022

T · · Systems ·

Let's power
higher performance

Imprint

Issued by	T-Systems International GmbH PU Digital Solutions Hahnstraße 43d 60528 Frankfurt am Main GERMANY

Area of application	**Valid from**	**Title**
WHO DDCCG	11.03.2022	WHO DDCCG Security Review Report

Version	**Last Review**	**Status**
1.0	11.03.2022	Initial

Contact	**Telephone**	**E-Mail**
Franziska Maeder	Phone no. +41 76 382 86 49	franziska.maeder@t-systems.com

Brief Details
This is the Security Review Report according to the OFFER RFP - DDCC Gateway for World Health Organization (WHO), chapter “4.4.1 Security Reviews” dated 08.11.2021, offer no. 1001886906.

Table 1: Imprint Contact

Version History and Distribution List

Version	Last Revised	Edited by	Changes/Comments
0.8	11.03.2022	D. K.	Initial
0.9	11.03.2022	J. K.	Internal Review
1.0	11.03.2022	J. S.	Release

Table 2: Version History

Table of Contents

List of Tables

Management Summary

Deutsche Telekom Security GmbH provides a security review, which includes a source code review and penetration test, for the “Digital Documentation of COVID-19 Certificates Gateway” (DDCCG). This security review has been conducted in accordance to the offer RFP - DDCC Gateway for World Health Organization (WHO), chapter “4.4.1 Security Reviews” dated 08.11.2021, offer no. 1001886906.

The goal of this test is to identify vulnerabilities that an attacker can exploit to compromise systems or the data stored on them, to gain access to sensitive information, or to compromise their availability.

Examination results

The during the penetration, no significant vulnerabilities in the application were discovered. The behavior of the test object meets expectations and did not suggest any deviant behavior. From an overall perspective the application appears highly resilient and leverages proven technologies.

However, while isolated, small deficits could be identified, which do not pose significant security threats. Findings – executed not in line with the “best practice” principle – could be identified both in the code and the behavior of the application. However, throughout the test these could not be successfully exploited.

The following should be improved upon in the context of one of the application’s next updates:

- Critical functions in the code, especially those, that process user input, should be equipped with exceptions in order to increase the application’s stability.

Automatically sending

A detailed description of the findings related to the application can be found in chapter 2 and 3. These chapters break down the two parts of the test, which essentially include a static code review as well as an active check of the API.

For each finding we provide a proposal for a solution or an improvement. If and how these are implemented is beyond the scope of our work and sphere of influence. It cannot be guaranteed that executing these recommendations strengthens the application or prevents unknown or not publicly known vulnerabilities, which for example are contained in third party parts of the applications, from posing a potential security threat.

Disclaimer

The tested application was a reference object, which is used for illustration and demonstration purposes only. The application was not tested in a productive environment or in an environment similar to a productive one. It is strongly recommended to perform a security test by an independent third party in the productive environment before release of the application.

Deutsche Telekom Security GmbH is a subsidiary of Deutsche Telekom AG, which also owns T-Systems. Therefore, Deutsche Telekom Security GmbH cannot be considered an “independent third party”. A security test of the DDCCG by an independent third party is strongly recommended, as the present report cannot fulfil the purpose of an independent security review.

1 Introduction

The test was carried out as a “White Box” penetration test with access to source code and APIs. Detailed knowledge of the system landscape where the application was deployed was not included.

The following artifacts were provided for the test:

- Access to source code repositories
- Specification for external APIs which are used by the reviewed system
- Certificates for client authentication
- URL where the API is reachable

1.1 Object of investigation

The following Objects was given from the development:

Source Code

Name	**URL**	**Version**
ddcc-gateway-lib	https://github.com/WorldHealthOrganization/ddcc-gateway-lib	1.2.2
ddcc-gateway	https://github.com/WorldHealthOrganization/ddcc-gateway	1.4.2
ddcc-trusted-party-reference-implementation	https://github.com/WorldHealthOrganization/ddcc-trusted-party-reference-implementation	-

API Endpoint

http://ddcc-gateway.861b530c4a22413cb791.westeurope.aksapp.io/

1.2 Investigation period and effort

The investigation was carried out between the 07.03.2022 and 10.03.2022. The total effort for the security investigation was four person days.

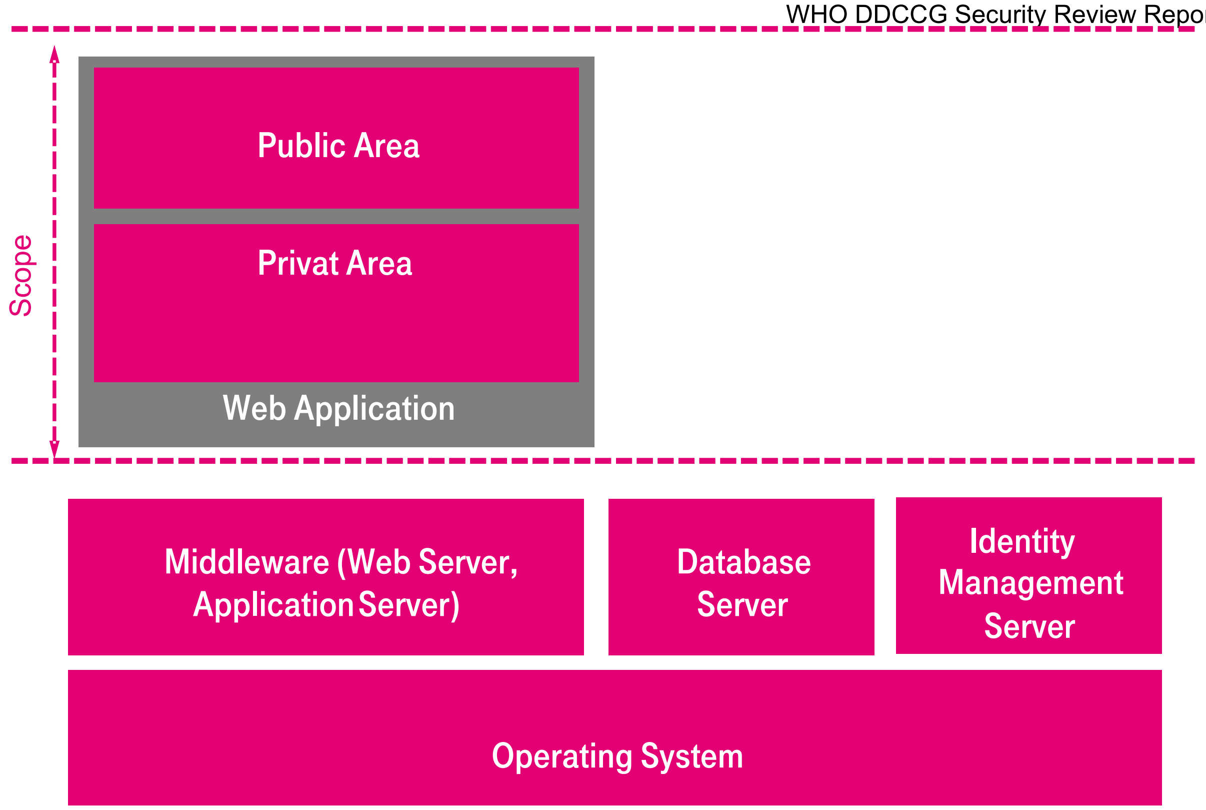

Figure 1: Representation of the system components

1.3 Delineation of the targets

The penetration test for the API only includes the known endpoint. Things like operating systems, as well as the middleware products, databases, web servers and participating network components are not included.

1.4 Source code analysis

In a source code analysis, the source code of an application is subjected to a code analysis. During the analysis, security-relevant aspects are considered in a focused manner.

The code analysis is carried out in four steps:

Preparation (Information & Scoping)

After receiving the code, e.g. via a code repository, it is roughly sifted through and the environment for compiling the code is set up.

Automatic analysis

In the second step, a code analysis tool is applied to the code received, or alternatively, existing results from a code analysis tool are used. After the tool has been used, the reported findings are checked by an experienced examiner, as automated code analysis produces a high number of false positives. As an interim result, this produces an initial list of vulnerabilities. Furthermore, the assessment of the severity of the potential vulnerability is checked manually for each identified vulnerability, as the automatic assessment of criticality is mostly based on the isolated finding and does not consider the full context. The intermediate result is a list of vulnerabilities with an assessment of the severity.

Manual review

The most important step is a manual code analysis. This can focus on specific topics such as permissions and roles, cryptography functions or memory management. The aim is to find errors that are not detected automatically with static code analysis tools. These types of errors are mostly logical errors. In addition, it is partly checked whether there are deviations from the best possible implementation proposed for secure designs.

Documentation

The review is followed by the documentation phase.

1.5 Dynamic API testing

Dynamic API testing is used to see how an API behaves, when live requests are issued to it.

The API testing is carried out in four steps:

Test setup

The technical requirements are created to carry out the test. This involves setting up systems that are necessary for communication with the API. These systems are needed to send, receive, and analyze data traffic. Required software comments are selected and added to the systems. Furthermore, it is ensured whether special authorizations are required for communication with the API, such as certificates. Existing descriptions of the API to be tested are loaded into the test system.

Automatic scans

In this step scanners automatically check for weak points. A large number of predefined requests are sent to the API endpoint and the responses are evaluated. In addition to regular scans, techniques such as fuzzing are also used. Fuzzing helps to generating the highest possible volume of different requests to make the system and its behavior transparent. Also, the scanners check how data was transported. This indicates whether weak transmissions and encryption methods are being used. At the end of the scanning process, the output is reviewed to identify any weak points.

Manual testing

This step is the most time-intensive in the process. The API specification is reviewed for any weak points or possible configuration errors that may lead to weaknesses. In addition, targeted requests are sent to the API, which contain data that either comply with the API specifications or deviate from them. Logical attack vectors are also identified, and exploitation is tested. This involves checking certain API functions and trying to force them to behave differently. Here too, the behavior of the API is documented and compared to the expected behavior.

Among other things, the transport of data – i.e., if and how data was transported – is analyzed. Particularly relevant is the forcing the transport of data via identity spoofing or an unauthorized transport between client and application.

Documentation

The API testing is followed by the documentation phase.

1.6 Limit of the tests performed

A security investigation has the goal to uncover security vulnerabilities and recommend measures to remedy the situation.

Maintaining system and network security is a dynamic process as new weaknesses in IT systems are exposed each day. The tests performed should therefore not be taken as a guarantee that the systems are permanently immune to any kind of attack.

Each penetration test is performed by Deutsche Telekom Security with the utmost care and based on the attack methods known at the time of the test. The source code review is conducted for a specific version of the application. The test results should therefore be considered as a snapshot of system security. Changes to the application after performing the tests may result in a positive or negative change in system security. It is therefore recommended to conduct security investigations after any major change or update.

Furthermore, safety investigations should be repeated at regular intervals. Only in this way can it be ensured that the systems are adequately protected against current attacks.

1.7 Rating system for the weak points

CVSS 3.1 score is used to evaluate the vulnerabilities. Common Vulnerability Scoring System (CVSS) (https://www.first.org/cvss/)

2 Vulnerabilities found in the code review

This chapter addresses vulnerabilities which were identified as part of the static analysis.

2.1 ddcc-gateway-lib

2.1.1 Improper exception handling

Improper Exception Handling

Description

The method update at line 53 and 68 of **ddcc-gateway-lib-main\src\main\java\eu\europa\ec\dgc\generation\CopyDigest.java** performs an operation that could be expected to throw an exception, and is not properly wrapped in a try-catch block.

```
44     /**
45      * Updates the Message Digest with one byte.
46      *
47      * @param in byte to update.
48      */
49     public void update(byte in) {
50         if (wasReset) {
51             sha256Digest.update(in);
52         } else {
53             binaryOut.write(in);
54         }
55     }
```

```
57     /**
58      * Updates the Message Digest with a byte array.
59      *
60      * @param in     byte array to insert
61      * @param offset Offset
62      * @param length length
63      */
64     public void update(byte[] in, int offset, int length) {
65         if (wasReset) {
66             sha256Digest.update(in, offset, length);
67         } else {
68             binaryOut.write(in, offset, length);
69         }
70     }
```

CVSS 3.1 Score

0.0 None (informational)
CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:N

Recommendation/Corrective measures

Properly handle exceptions to avoid unintended behavior and crashes.

2.1.2 Input stream is not properly closed

Input stream is not properly closed

Description

The application's coseToQrCode method in **ddcc-gateway-lib-main\src\main\java\eu\europa\ec\dgc\generation\DgcGenerator.java** defines and initializes the DeflaterInputStream / Input stream object at line 136. This object encapsulates a limited resource and is not properly closed. This behavior ties up resources unnecessarily, which can cause a drop in performance.

```
128     /**
129      * convert cose bytes to qr code data.
130      *
131      * @param cose signed edgc data
132      * @return qr code data
133      */
134     public String coseToQrCode(byte[] cose) {
135         ByteArrayInputStream bis = new ByteArrayInputStream(cose);
136         DeflaterInputStream compessedInput = new DeflaterInputStream(bis, new Deflater(9));
137         byte[] coseCompressed;
138         try {
139             coseCompressed = compessedInput.readAllBytes();
140         } catch (IOException e) {
141             throw new IllegalArgumentException(e);
142         }
143         String coded = Base45Encoder.encodeToString(coseCompressed);
144         return "HC1:" + coded;
145     }
```

The application's loadKeyStore method in **ddcc-gateway-lib-main\src\main\java\eu\europa\ec\dgc\gateway\connector\config\DgcGatewayConnectorKeystore.java** defines and initializes the FileInputStream object at 153. This object encapsulates a limited resource and is not properly closed. This behavior ties up resources unnecessarily, which can cause a drop in performance.

```
142     private void loadKeyStore(KeyStore keyStore, String path, char[] password)
143         throws CertificateException, NoSuchAlgorithmException {
144         try {
145 
146             InputStream stream;
147 
148             if (path.startsWith("$ENV:")) {
149                 String env = path.substring(5);
150                 String b64 = System.getenv(env);
151                 stream = new ByteArrayInputStream(Base64.getDecoder().decode(b64));
152             } else {
153                 stream = new FileInputStream(ResourceUtils.getFile(path));
154             }
155 
156             if (stream.available() > 0) {
157                 keyStore.load(stream, password);
158                 stream.close();
159             } else {
160                 keyStore.load(null);
161                 log.info("Could not load Keystore {}", path);
162             }
163         } catch (IOException e) {
164             log.info("Could not load Keystore {}", path);
165         }
166     }
167 }
```

CVSS 3.1 Score

0.0 None (informational)
CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:N

Recommendation/Corrective measures

Close the opened InputStreams correctly.

2.1.3 Use of a static initialization vector in cryptographic operations

No random initial vector in use

Description

The encryption method encryptData in **\ddcc-gateway-lib-main\src\main\java\eu\europa\ec\dgc\generation\DgcCryptedPublisher.java** does not use a random initial vector. This decreases the computational effort needed for an attacker to decrypt the data.

```java
    private void encryptData(DgcData dgcData, byte[] edgcCbor, PublicKey publicKey) throws
        NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
        InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException {
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(256); // for example
        SecretKey secretKey = keyGen.generateKey();

        // TODO set iv to something special
        byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        IvParameterSpec ivspec = new IvParameterSpec(iv);
        Cipher cipher = Cipher.getInstance(DATA_CIPHER);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivspec);
        byte[] edgcDataEncrpyted = cipher.doFinal(edgcCbor);

        dgcData.setDataEncrypted(edgcDataEncrpyted);

        // encrypt RSA key
        Cipher keyCipher = Cipher.getInstance(KEY_CIPHER);
        keyCipher.init(Cipher.ENCRYPT_MODE, publicKey, OAEP_PARAMETER_SPEC);
        byte[] secretKeyBytes = secretKey.getEncoded();
        dgcData.setDek(keyCipher.doFinal(secretKeyBytes));
    }
}
```

CVSS 3.1 Score

3.3 Low

CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N

Recommendation/Corrective measures

A random initial vector should be used. Additionally, the repeated use of the same combination of IV and encryption key should be avoided.

2.2 ddcc-trusted-party-reference-implementation

2.2.1 Content Security Policy and HSTS Header is not explicitly defined

Content Security Policy and HSTS is not explicitly defined

Description

In **\ddcc-trusted-party-reference-implementation-master\components\trustregistry\server.js** it could not clearly be identified if an explicit Content Security Policy and HSTS Header is defined.

```
const express = require('express')
const trusthandler = require('./modules/TrustHandler')
const app = express()
const port = 8080

app.get('/dcc', (req, res) => {
  return res.send(trusthandler.TRUST_REGISTRY["EUDCC"]);
})

app.get('/shc', (req, res) => {
  return res.send(trusthandler.TRUST_REGISTRY["SHC"]);
})

app.get('/icao', (req, res) => {
  return res.send(trusthandler.TRUST_REGISTRY["ICAO"]);
})

app.get('/cred', (req, res) => {
  return res.send(trusthandler.TRUST_REGISTRY["CRED"]);
})

app.listen(port, () => {
  trusthandler.initialize();
  console.log(`Listening on port ${port}`)
})
```

The Content-Security-Policy header enforces that the source of content, such as the origin of a script, embedded (child) frame, embedding (parent) frame or image, are trusted and allowed by the current webpage; if, within the webpage, a content's source does not adhere to a strict Content Security Policy, it may be rejected by the browser. Failure to define a policy may leave the application's users exposed to Cross-Site Scripting (XSS) attacks, Clickjacking attacks, content forgery and more.

Without a HTTP Strict Transport Security (HSTS) Header an attacker can perform a Man-in-the-Middle attack and manipulate it to redirect users to a malicious webpage of the attacker's choosing. To protect the user from such an occurence, the (HSTS) header instructs the user's browser to disallow use of an unsecure HTTP connection to the the domain associated with the HSTS header. Once a browser that supports the HSTS feature has visited a webpage and the header was set, it will no longer allow communicating with the domain over a HTTP connection. Once an HSTS header was issued for a specific website, the browser is also instructed to prevent users from manually overriding and accepting an untrusted SSL certificate for as long as the "max-age" value still applies. The recommended "max-age" value is for at least one year in seconds, or 31536000.

CVSS 3.1 Score

3.3 Low

CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

Recommendation/Corrective measures

It must be ensured that explicit settings are made for the setting of the Content Security Policy and HSTS.

2.3 ddcc-gateway

2.3.1 InputStream is not properly closed

InputStream is not properly closed

Description

The application's loadKeyStore method in **ddcc-gateway-main\src\main\java\eu\europa\ec\dgc\gateway\config\DgcKeyStore.java** defines and initializes the getResourceAsStream object at line 95. This object encapsulates a limited resource and is not properly closed. This behavior ties up resources unnecessarily, which can cause a drop in performance.

```
88      private void loadKeyStore(KeyStore keyStore, String path, char[] password)
89          throws CertificateException, NoSuchAlgorithmException, IOException {
90
91          InputStream fileStream;
92
93          if (path.startsWith("classpath:")) {
94              String resourcePath = path.substring(10);
95              fileStream = getClass().getClassLoader().getResourceAsStream(resourcePath);
96          } else {
97              File file = new File(path);
98              fileStream = file.exists() ? getStream(path) : null;
99          }
00
01          if (fileStream != null && fileStream.available() > 0) {
02              keyStore.load(fileStream, password);
03              fileStream.close();
04          } else {
05              keyStore.load(null);
06              log.info("Could not find Keystore {}", path);
07          }
08
09      }
10
```

The application's loadKeyStore method in **ddcc-gateway-main\src\main\java\eu\europa\ec\dgc\gateway\config\DgcKeyStore.java** defines and initializes the getStream object at line 98. This object encapsulates a limited resource and is not properly closed. This behavior ties up resources unnecessarily, which can cause a drop in performance.

```
88     private void loadKeyStore(KeyStore keyStore, String path, char[] password)
89         throws CertificateException, NoSuchAlgorithmException, IOException {
90 
91         InputStream fileStream;
92 
93         if (path.startsWith("classpath:")) {
94             String resourcePath = path.substring(10);
95             fileStream = getClass().getClassLoader().getResourceAsStream(resourcePath);
96         } else {
97             File file = new File(path);
98             fileStream = file.exists() ? getStream(path) : null;
99         }
100 
101        if (fileStream != null && fileStream.available() > 0) {
102            keyStore.load(fileStream, password);
103            fileStream.close();
104        } else {
105            keyStore.load(null);
106            log.info("Could not find Keystore {}", path);
107        }
108 
109    }
110 
```

The application's loadKeyStore method in **ddcc-gateway-main\src\main\java\eu\europa\ec\dgc\gateway\config\DgcKeyStore.java** defines and initializes the getStream object at line 98. This object encapsulates a limited resource and is not properly closed. This behavior ties up resources unnecessarily, which can cause a drop in performance.

```
    }

    private InputStream getStream(String path) {
        try {
            return new FileInputStream(path);
        } catch (IOException e) {
            log.info("Could not find Keystore {}", path);
        }
        return null;
    }
```

CVSS 3.1 Score

0.0 None (informational)

CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:N

Recommendation/Corrective measures

Close the object InputStreams correctly.

3 Vulnerabilities found based on penetration testing

In this part active tests on the endpoint of the application were conducted to monitor and analyze the behavior towards various requests.

The following vulnerabilities could be identified:

3.1 Information disclosure in responses

Information disclosure

Description

Information Disclosure refers to the publication of not publicly available information/data. This could include eMail addresses, internal IP addresses, error messages, source code, version numbers of utilized software, client data or other information.

The API Endpoint responds to certain request, which if not correctly interpreted or are invalid, with information about the details of the error. A potential attacker can gather information about the system or application behavior from this. This kind of information can be used for the development of further attack vectors.

Example:

```
GET /trustList/DSC/zipfiles HTTP/1.1
Accept: application/json
User-Agent: PostmanRuntime/7.28.3
Host: ddcc-gateway.861b530c4a22413cb791.westeurope.aksapp.io
Accept-Encoding: gzip, deflate
Connection: close
Cookie: JSESSIONID=7D21A62E136464978DA0C10EC559DECD


HTTP/1.1 400
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
Content-Type: application/json
Date: Tue, 08 Mar 2022 09:48:17 GMT
Connection: close
Content-Length: 156

{"code":"0x001","problem":"Validation Error","sendValue":"","details":"downloadTrustListFilteredByCountryAndType.countryCode: size must be between 2 and 2"}
```

Information disclosure
CVSS 3.1 Score
0.0 None (informational) https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N
Recommendation/Corrective measures
Details about errors should not be sent to the client. For logging or debugging reasons, internal status codes can be used, instead of detailed information.

4 Tools and Applications used for the penetration test

Application	Version	Description
NMap	7.92	Port scanner
Kali Linux	Rolling	Distribution for Penetration Tests
Burp	2021.10.2	Pentesting-Framework for webapplications
Postman	8.10.0	API-Development platform
OWAS-Dependency Check	7.0.0	Software Composition Analysis (SCA) tool
Checkmarx	9.4.3	Static analysis tool
Visual Studio Code	1.65.1	Code Editor