Penetration Test Report:

DENIC ID Relying Party - Member Login

Version: 1.2.1
22.04.2024

Dr. Juraj Somorovsky
Phone: (+49)(0)234 / 45930961 | E-Mail: Juraj.Somorovsky@hackmanit.de

Project Information

Customer: DENIC eG
                      Kaiserstraße 75 - 77
                      60329 Frankfurt am Main, Deutschland

Contact: Marcos Sanz

Commissioned to: Hackmanit GmbH
                      Universitätsstraße 60 (Exzenterhaus)
                      44789 Bochum, Germany

Project executive: Dr. Juraj Somorovsky
                      Phone: (+49)(0)234 / 45930961
                      Fax: (+49)(0)234 / 45930960
                      E-Mail: Juraj.Somorovsky@hackmanit.de

Project members: Mario Korth (Hackmanit GmbH)
                      Dr. Christian Mainka (Hackmanit GmbH)
                      Karsten Meyer zu Selhausen (Hackmanit GmbH)
                      Dr. Vladislav Mladenov (Hackmanit GmbH)

Project period: June 4, 2019 – June 11, 2019

Version of the report: 1.2.1

This report was technically verified by Dr. Christian Mainka.
This report was linguistically verified by David Herring.

Hackmanit GmbH
Represented by: Prof. Dr. Jörg Schwenk, Dr. Juraj Somorovsky,
Dr. Christian Mainka, Prof. Dr. Marcus Niemietz
Register court: Amtsgericht Bochum, HRB 14896, Germany

Contents

1 Summary

DENIC ID is the first widely-deployed implementation of the ID4me protocol [1]. ID4me is a novel protocol for federated identity management whose two main goals are to provide *(1) Authorization of a user for access to any third party accepting ID4me identifiers* and *(2) Controlled communication of the user's personal information to the third parties accessed by the user* [1]. ID4me is based on well-established standards such as OpenID Connect (OIDC) [8] and the domain name system (DNS) [4].

Hackmanit GmbH was commissioned to perform a penetration test on a relying party in the context of DENIC ID - *the new DENIC Member Login page*. The penetration test was performed remotely with a total expense of 11 PT.

Weaknesses. During the penetration test, three weaknesses classified as *Medium* were identified. Two of these weaknesses relate to the insufficient protection against cross-site request forgery (CSRF) attacks. First, the login page does not contain CSRF protection mechanisms like CSRF tokens, which allows an attacker to force a victim to start an authentication flow without its consent (M01). Second, the presence of the `state` parameter, which is used to protect against CSRF attacks in the OIDC protocol, is not enforced by the relying party (M02). This enables an attacker to log a victim into an account controlled by the attacker which might result in the victim revealing personal information or files to the attacker. The third weakness could allow an attacker to compromise the account of a victim due to faulty session and cookie management when the victim logs in again after a successful logout using the same browser (M03). Some of the weaknesses identified during the penetration test are weaknesses in the library `OpenID-Connect-PHP`[1] which the tested relying party is based on. We responsibly disclosed these weaknesses to the library developers in June 2019 and supported them by implementing security fixes.

Structure. The report is structured as follows: In Section 2, the timeline of the penetration test is listed. Section 3 introduces our methodology and Section 4 explains the general conditions and scope of the penetration test. Section 6 provides an overview of the identified weaknesses, as well as, further recommendations and information. In Section 7, all identified weaknesses are discussed in detail and specific countermeasures are described. Section 8 summarizes our recommendations resulting from observations of the application. Finally, Section 9 lists additional tests that did not reveal any weaknesses.

---

[1]https://github.com/jumbojett/OpenID-Connect-PHP

2 Project Timeline

The penetration test was performed remotely between June 4, 2019 and June 11, 2019. Four penetration testers with different technical backgrounds were involved with a total expense of 11 PT.

3 Methodology

Among others, the following tools were used for the penetration test:

Tool	Link
Mozilla Firefox	https://www.mozilla.org/de/firefox/
Google Chrome	https://www.google.com/intl/de_ALL/chrome/
Burp Suite Professional	https://portswigger.net/burp
EsPReSSO	https://github.com/RUB-NDS/BurpSSOExtension
testssl.sh	https://testssl.sh/
TLS-Scanner	https://github.com/tls-attacker/TLS-Scanner
Self-developed tools	-

Risk Rating. Each weakness has its own CVSS 3.1 base score rating (*Common Vulnerability Scoring System Version 3.1 Calculator*).[2,3] Based on the CVSS 3.1 base score, the following weaknesses assessment is performed:

0.0 – 3.9:	Low
4.0 – 6.9:	Medium
7.0 – 8.9:	High
9.0 – 10.0:	Critical

4 General Conditions and Scope

In the scope of the grey-box penetration test was the new DENIC Member Login page which was accessible at: https://member.secure.denic.de/member-login/new/

In contrast to the old login page, it supports the ID4me implementation, DENIC ID, to allow users to log in using their DENIC ID identifier.

In terms of the ID4me standard, the login page represents a relying party which uses ID tokens issued by an identity authority to identify and authenticate the user during the login process.

---

[2]https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator
[3]https://www.first.org/cvss/v3.1/user-guide

The relying party only supports the use of one predefined identity authority operated by DENIC. Therefore, a scenario with multiple identity authorities is explicitly out of the scope of this penetration test. If an relying party supports more than one identity authority, further security considerations and possible attacks must be taken into account.

5 Scenario Description

DENIC ID is an implementation of ID4me [1] – an "Open, Global, Federated Standard For The Digital Identity Management".[4] It is based on established standards such as OIDC and the DNS. In contrast to other single sign-on (SSO) schemes, ID4me divides the duties of the identity provider (IdP) into two separated entities: an identity agent and an identity authority. The identity agent provides registration services and manages user data. The identity authority is responsible for user authentication and authorization. This role separation results in the following four entities being involved in a login process based on ID4me:

User A user utilizing ID4me to log in at an online service. His user account is associated with an ID4me identifier.

Relying party An online service which supports logins using an ID4me identifier.

Identity agent The entity providing ID4me services to the user. This includes the registration and management of ID4me identifiers as well as storage and distribution of the user's personal data to relying partys in so-called "claims".

Identity authority The entity responsible for user authentication and for ensuring that the user authorized the specific relying party to access his personal information.

ID4me identifiers are used to identify the user when he/she wants to log in at a relying party. An ID4me identifier can be any hostname identified by a valid DNS entry which contains a TXT record. This record specifies the responsible identity authority and identity agent.

The process of registering a new ID4me identifier was not in the scope of this penetration test. Therefore, it is not described here. Information on the process can be found in the ID4me documentation [1].

The process of logging in at a relying party using an ID4me identifier is depicted in Figure 1 and described in the following:

1. The user starts the login process on the relying party by providing his/her ID4me identifier.

2. The relying party queries the DNS for the user's identifier to acquire the responsible identity authority and identity agent.

3. If the relying party is not already registered at the identity authority, it performs Dynamic Client Registration [7] according to the OIDC standard.

---

[4]https://id4me.org/about/
[5]https://gitlab.com/ID4me/documentation/blob/1a8e464b42ef6f57e75ec3c7f1a23e878dbbe42d/id4me%20Technical%20Overview%20v1.3.pdf
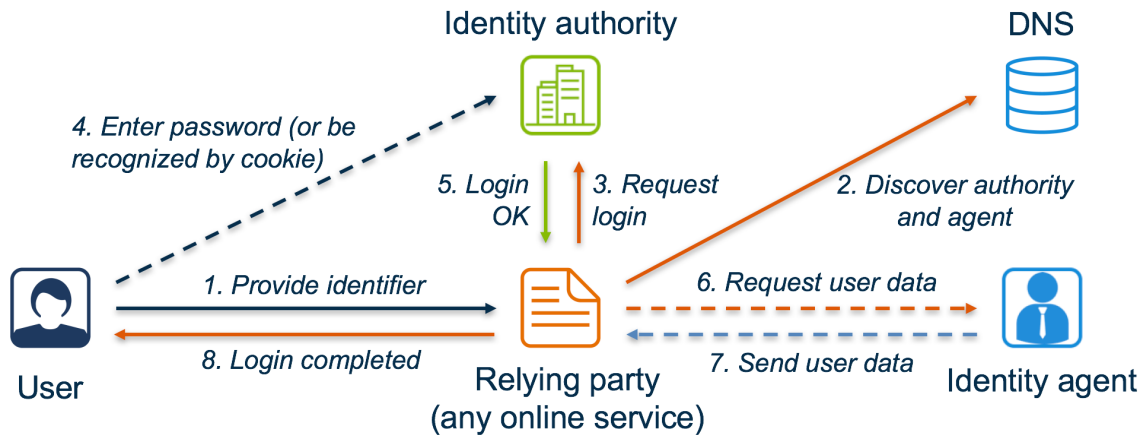
Figure 1: Process of logging in at a relying party using an ID4me identifier. The figure is taken from the official ID4me documentation.[5]

4. The relying party redirects the user to the identity authority. The user authenticates at the identity authority and authorizes, or rejects, access to the claims requested by the relying party on the consent page displayed by the identity authority.

5. The identity authority redirects the user back to the relying party and delivers the authorization code to the relying party in this redirection. The relying party redeems the authorization code at the token endpoint of the identity authority and receives an access token and an ID token.

6. If the relying party wants to access claims in addition to the information present in the ID token, it queries the userinfo endpoint of the identity authority using the access token. The identity authority makes use of the OIDC distributed claims mechanism[6] and refers the relying party to the identity agent. The relying party queries the userinfo endpoint of the identity agent using the access token.

7. If the access token is valid, the identity agent provides all claims which the relying party is authorized to access. If there is no information stored for a requested claim, the claim is omitted from the identity agent's response.

Despite generally implementing ID4me, DENIC ID differs from the standard in some crucial aspects. ID4me does not cover the trust relationship between the identity agent and the identity authority; in ID4me every user is allowed to set up and to operate his/her own identity agent. DENIC ID is more specific in this regard and only supports pre-registered identity agents which have a valid contract with the DENIC. Additionally, DENIC ID suggests that a relying party does not trust every identity authority but only a list of predefined authorities. This limits the degrees of freedom provided by ID4me, but increases the security by limiting the parties which can participate in the protocol and establishes more trust between these parties.

The relying party in the scope of this penetration test – the new DENIC Member Login page – only supports the use of one predefined identity authority. For the penetration test, DENIC

---

[6]https://openid.net/specs/openid-connect-core-1_0.html#AggregatedDistributedClaims

configured it to use an identity authority operated by us. This allowed us to craft validly signed ID tokens containing arbitrary information and use them to test the behavior of the new login page.

Due to the hardcoded configuration, the relying party does not use the discovery or dynamic client registration process of the OIDC protocol but always uses predefined client credentials and URLs for the invocation of different OIDC endpoints.

We were provided with the following two test accounts which were already registered and could be linked to DENIC ID identifiers choosen by us using the DENIC member area: *pentest1* and *pentest2*.

6 Overview of Weaknesses, Recommendations, and Information

Risk Level	Finding	Reference
M01	**Valid OpenID Connect Flow with a Missing `state` Parameter:** The relying party does not enforce the presence of the `state` parameter.	Section 7.1, page 10
M02	**Insufficient Cross-Site Request Forgery Protection:** The new Member Login page does not provide sufficient protection against CSRF attacks.	Section 7.2, page 11
M03	**Faulty Session Management and Missing Fresh Cookie Generation:** Users retrieve the same cookies after repeating the login procedure at the relying party.	Section 7.3, page 12
L01	**Valid OpenID Connect Flow with a Replayed `state` Parameter:** The relying party does not verify whether the value of the `state` parameter has been reused.	Section 7.4, page 13
L02	**Enforce Strict Comparisons for the Values of ID Token Claims:** The `OpenID-Connect-PHP` library should be modified to use strict comparisons.	Section 7.5, page 14
L03	**Enforce Validation of `iat` and `exp` Claims in the ID Token:** The presence of the claims `iat` and `exp` is not enforced.	Section 7.6, page 15
R01	**Use the OpenID Connect Parameter `nonce`:** The relying party should use the `nonce` parameter.	Section 8.1, page 17
R02	**Repeating Values in Ephemeral TLS-ECDH Keys:** The TLS server should be configured to always use fresh ECDH ephemeral keys.	Section 8.2, page 17
R03	**Remove References to `CRYPT_RSA`:** Any reference to the obsolete `CRYPT_RSA` library should be removed from the relying party.	Section 8.3, page 19
R04	**Prevent the Use of Uninitialized Values:** All values used in the authentication process should be initialized properly.	Section 8.4, page 19

R05	**Potentially Insecure XML Parsing of RSA Keys:** The content of RSA keys should be verified before parsing them as XML.	Section 8.5, page 19

Risk Definitions:

Critical Risk	Weaknesses classified as *Critical* can be exploited with very little effort by an attacker. They have very large negative effects on the tested system, its users and data, or the system environment.
High Risk	Weaknesses classified as *High* can be exploited with little effort by an attacker. They have a major negative impact on the tested system, its users and data, or the system environment.
Medium Risk	Weaknesses classified as *Medium* can be exploited with medium effort by an attacker. They have a medium negative impact on the tested system, its users and data, or the system environment.
Low Risk	Weaknesses classified as *Low* can only be exploited with great effort by an attacker. They have little negative impact on the tested system, its users and data, or the system environment.
Recommendation	*Recommendation* identifies measures that may increase the security of the tested system. Implementation is recommended, but not necessarily required.
Information	Observations classified as *Information* are usually no weaknesses. Examples of these observations are unusual configurations and possibly unwanted behavior of the tested system.

7 Weaknesses

In the following sections, we list the identified weaknesses. Every weakness has an identification name which can be used as a reference in the event of questions, or during the patching phase.

7.1 M01 Valid OpenID Connect Flow with a Missing `state` Parameter

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**Low**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**Low**
Privileges Required (PR)	**Low**	Availability Impact (A)	**None**
User Interaction (UI)	**Required**	Scope (S)	**Unchanged**
Subscore: **2.1**		Subscore: **2.5**	

Overall CVSS Score for M01 : **4.6**

General Description. Cross-site request forgery (CSRF) is an attack in which an attacker tricks his victim into performing authenticated commands changing the application state [5] without the victim's consent. In OAuth and OIDC the `state` parameter is used to mitigate CSRF attacks. It is randomly generated by the relying party at the beginning of each authentication flow. The redirect, which is used to send the code generated by the identity authority to the relying party, also contains the `state` parameter. This enables the relying party to verify that the authentication flow was triggered by the user.

Weakness. The relying party does not enforce the presence of a `state` parameter. If the `state` parameter is missing and only a valid `code` is provided, the relying party redeems the `code` at the identity authority and uses the issued ID token to successfully log in the user; see also Figure 2.[7]

This behavior allows an attacker to force the victim to sign in at the relying party using the attacker's account.

Countermeasures. The relying party must enforce the presence of the `state` parameter and validate that its value matches the value chosen at the beginning of the authentication flow.

---

[7]Note that a valid `fe_typo_user` cookie is needed to perform this operation. This cookie can be present in the user's browser after visiting the relying party or can simply be obtained by triggering the login procedure at the relying party.
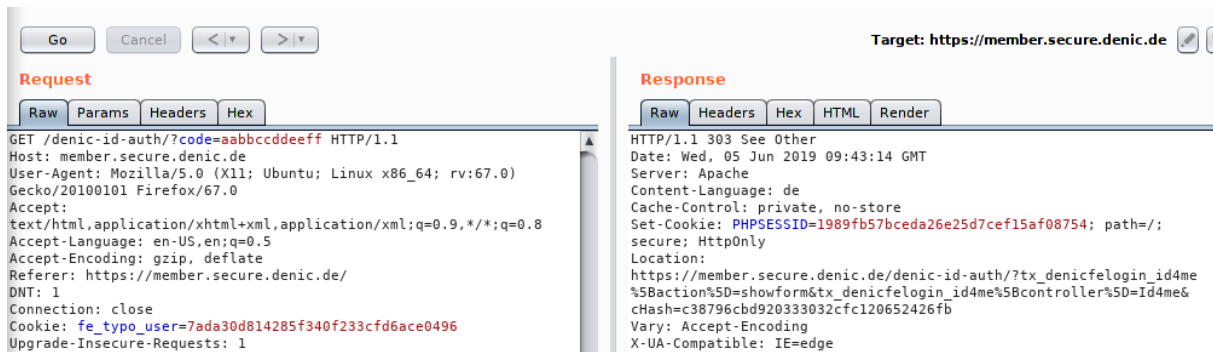
Figure 2: Successful session initialization with a missing `state` parameter.

7.2 M02 Insufficient Cross-Site Request Forgery Protection

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**None**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**Low**
Privileges Required (PR)	**None**	Availability Impact (A)	**None**
User Interaction (UI)	**Required**	Scope (S)	**Unchanged**
Subscore: **2.8**		Subscore: **1.4**	

Overall CVSS Score for M02 : 4.3

General Description. CSRF attacks are usually possible since browsers automatically attach cookies to every HTTP request, regardless of the request origin. Therefore, it's impossible for the server application to distinguish between a valid user-initiated request and an invalid request executed without the user's consent.

Weakness. The Member Login page does not apply any CSRF protection. An attacker could abuse this to force his victim to perform an authentication flow. If the user is logged in at the identity authority and the identity authority does not provide any consent page, the user would perform the complete OIDC authentication flow and seamlessly log in at the relying party.

A proof-of-concept attack vector is provided in Listing 1.

Countermeasures. We recommend to add CSRF protection to all parts of the web application which allow the execution of crucial actions. This can be achieved by using CSRF tokens.

```
1  <html>
2    <!-- CSRF PoC - generated by Burp Suite Professional -->
3    <body>
4      <script>history.pushState('', '', '/')</script>
5      <form action="https://member.secure.denic.de/denic-id-auth/" method="POST">
6        <input type="hidden" name="tx&#95;denicfelogin&#95;id4me&#91;denicid&#93;" value="pentest1&#46;pen190603de
              &#46;hckmnt&#46;de" />
7        <input type="hidden" name="pass" value="" />
8        <input type="hidden" name="pid" value="99&#44;2337&#44;2497&#44;2494" />
9        <input type="hidden" name="redirect&#95;url" value="&#47;startseite&#47;" />
10       <input type="hidden" name="tx&#95;felogin&#95;pi1&#91;noredirect&#93;" value="0" />
11       <input type="submit" value="Submit request" />
12     </form>
13   </body>
14 </html>
```

Listing 1: A proof-of-concept for a CSRF attack forcing the victim to log in.

7.3 `M03` Faulty Session Management and Missing Fresh Cookie Generation

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Physical**	Confidentiality Impact (C)	**High**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**None**
Privileges Required (PR)	**None**	Availability Impact (A)	**None**
User Interaction (UI)	**Required**	Scope (S)	**Unchanged**
Subscore: **0.7**		Subscore: **3.6**	

Overall CVSS Score for `M03` **: 4.3**

General Description. Proper session management requires that sessions are invalidated upon logout. OWASP states that "if a session can still be used after logging out, then the lifetime of the session is increased and that gives third parties that may have intercepted the session token more (or perhaps infinite, if no absolute session expiry happens) time to impersonate a user."[8] Users might want to log out at the relying party for different reasons and should be able to terminate their sessions. One of the more obvious reasons is the use of public computers on which users might not use the private mode. Being unable to log out correctly increases the risk that the user's session is compromised and an attacker takes over the user's account [6]. After the user logs in again at the web application, a new fresh session ID must be generated.

Weakness. The relying party provides a logout functionality and correctly invalidates the session ID cookies (`fe_typo_user` and `PHPSESSID`) upon logout. However, after performing the authentication flow with a logged out user again, the relying party does not generate fresh cookies. Instead, the old cookies (which are still present in the user's browser) become valid again.

This problem could, for example, allow an attacker to steal cookies after the user logs out. Once the user performs another login, the cookies become valid again and can be misused by the attacker.

---

[8]https://owasp-aasvs.readthedocs.io/en/latest/requirement-3.2.html

Countermeasures. The relying party must generate new fresh session ID cookies after every successful login.

We also recommend to unset the cookies in the user's browser upon logout and restrict their validity period [6]. Currently, the session ID cookies have no expiration period.

7.4　L01　Valid OpenID Connect Flow with a Replayed `state` Parameter

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Adjacent Network**	Confidentiality Impact (C)	**Low**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**Low**
Privileges Required (PR)	**High**	Availability Impact (A)	**None**
User Interaction (UI)	**Required**	Scope (S)	**Unchanged**
Subscore: **0.7**		Subscore: **2.5**	

Overall CVSS Score for L01 : 3.2

General Description. As described in M01 , the `state` parameter is used to mitigate CSRF attacks. In order to fulfill its purpose, the relying party needs to randomly generate a new value for the `state` parameter at the beginning of each authentication flow and ensure that each choosen `state` value is only valid once.

Weakness. The relying party does not correctly validate whether the `state` parameter has been reused. When the `code` generated by the identity authority is submitted to the relying party, it is possible to reuse an old value for the `state` parameter. The relying party accepts the requests (see Figure 3 and Figure 4) and uses the provided `code` to obtain tokens at the token endpoint of the identity authority.

This behavior allows an attacker to bypass the CSRF protection usually provided by the `state` parameter, if he is able to obtain a valid value from a previous login flow of the user. For example, the `state` could appear in server logs which could leak to an attacker.

Countermeasures. The `state` parameter is a one-time-use parameter and the relying party must ensure that an already used value for the `state` parameter is not accepted again.
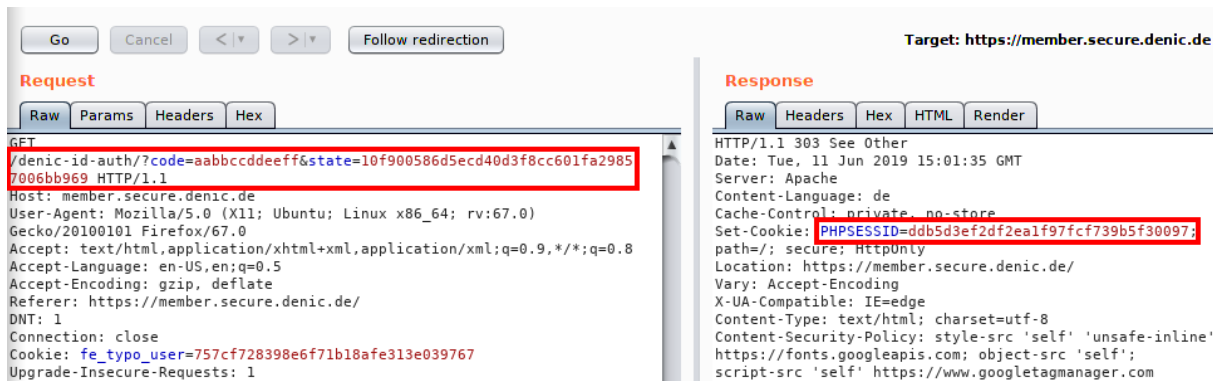
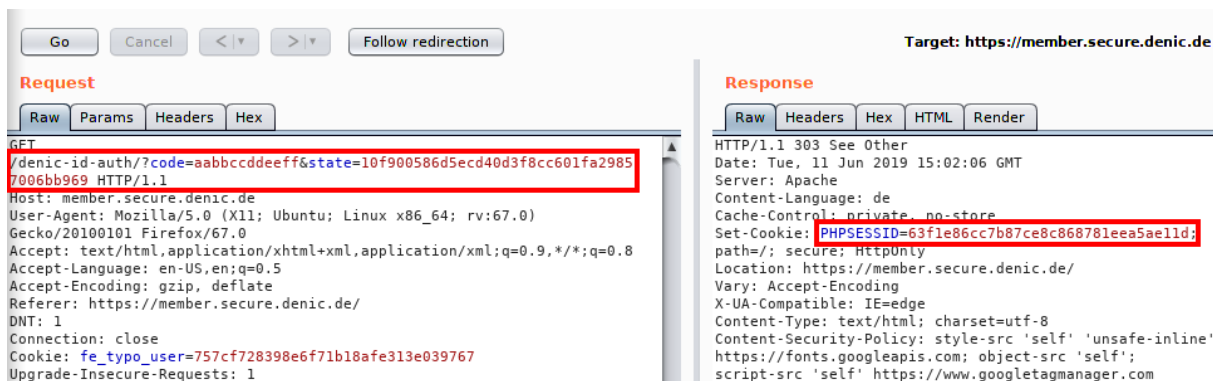Figure 3: First authentication response with a valid `code` and `state`.

Figure 4: Authentication response with a valid `code` and a replayed `state`.

7.5 L02 Enforce Strict Comparisons for the Values of ID Token Claims

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Adjacent Network**	Confidentiality Impact (C)	**None**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**Low**
Privileges Required (PR)	**High**	Availability Impact (A)	**None**
User Interaction (UI)	**None**	Scope (S)	**Unchanged**
Subscore: **0.9**		Subscore: **1.4**	

Overall CVSS Score for L02 : **2.4**

General Description. In PHP, there are two types of comparison operators: loose and strict. If a loose operator is used (e.g., ==, <=), the PHP interpreter first attempts to convert the two variables to the same type before performing the actual comparison. If a strict operator is used (e.g., ===), the comparison returns true if and only if both the types and values of the variables are equal.

Weakness. We discovered that the underlying `OpenID-Connect-PHP` library uses loose comparisons. This leads to comparisons like `$claims->iss == $this->getIssuer()` evaluating to `True` even if `$claims->iss` is set to the integer 0 and `$this->getIssuer()` returns a string. Another example would be the comparison `$claims->exp >= time()- $this->leeway`. Setting `$claims->exp` to `True` leads to the comparison *always* evaluating to `True`. This is possible since `$claims` stems from the JSON decoded ID token. Therefore, an attacker which is able to craft an ID token has full control over the types of the properties.

This weakness affects the following claims:

- `at_hash`

- `aud`

- `exp`

- `iss`

- `nbf`

Countermeasures. In general, the relying party should use strict comparisons. In PHP, this requires three comparison operators instead of two. In particular, a strict comparison for equality uses three equal signs (===). A strict comparison for greater or greater equal does not exist in PHP. Therefore, it is recommended to verify that the variables are the correct type before comparing their values.. A truth table for loose and strict comparisons can be found at https://www.php.net/manual/en/types.comparisons.php.

7.6 `L03` Enforce Validation of `iat` and `exp` Claims in the ID Token

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Adjacent Network**	Confidentiality Impact (C)	**None**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**Low**
Privileges Required (PR)	**High**	Availability Impact (A)	**None**
User Interaction (UI)	**None**	Scope (S)	**Unchanged**
Subscore: **0.9**		Subscore: **1.4**	

Overall CVSS Score for `L03` **: 2.4**

General Description. According to the OIDC standard [8] an ID token must contain two timestamps: One which states the time at which the ID token was issued (`iat` claim) and another that states the time at which the ID token expires (`exp`). The claims must be validated in the following ways:

1. The current time must be after the time represented by the `iat` claim.

2. The current time must be before the time represented by the `exp` claim [8].

Weakness. The underlying `OpenID-Connect-PHP` library does not enforce the presence of the claims `iat` and `exp`; both claims can be absent. Therefore, the client accepts tokens which might not expire at all.

Countermeasures. The relying party must enforce that the claims `iat` and `exp` are present in every ID token and that their values are validated within a reasonable time skew.

8 Recommendations

In the following sections, we provide our recommendations to improve the security of the tested system.

8.1 `R01` Use the OpenID Connect Parameter `nonce`

General Description. The OIDC standard suggests to use the `nonce` parameter "to associate a Client session with an ID token, and to mitigate replay attacks" [8]. The relying party randomly chooses a value for the `nonce` parameter and sends it to the identity authority in the authentication request. The identity authority later adds this value to the issued ID token. The relying party must verify that the `nonce` parameter is present when it receives the ID token, and contains the same value which was chosen earlier for this specific protocol flow.

Recommendation. We recommend to further increase the security of the relying party and the protection against well-known attacks, such as CSRF and replay attacks, by adding a binding between the authentication request and the ID token. This is achieved using the OIDC parameter `nonce` in the way described above.

8.2 `R02` Repeating Values in Ephemeral TLS-ECDH Keys

General Description. When performing a TLS-ECDHE handshake, the server sends a fresh elliptic curve (EC) key in the `ServerKeyExchange` message. The key should always be generated at random in order to achieve perfect forward secrecy.

Our tests with TLS-Scanner revealed that the server caches the EC key values and uses the same EC key for multiple connections. See Figure 5 and Figure 6.

Recommendation. We recommend configuring the TLS server to use fresh ephemeral keys for every handshake.

Figure 5: First TLS handshake performed with `member.secure.denic.de`.

Figure 6: Second TLS handshake performed with `member.secure.denic.de`. The EC key used is identical to the first handshake (see Figure 5).

8.3 `R03` Remove References to `CRYPT_RSA`

General Description.

We discovered that the underlying `OpenID-Connect-PHP` library still contains references to the superseded and unsupported library `CRYPT_RSA`.[9] While the superseding library `phpseclib/Crypt/RSA`[10] is still maintained, the `CRYPT_RSA` library received it's last update five years ago.[11]

Recommendation. We recommend removing any reference to the library `CRYPT_RSA` in the underlying `OpenID-Connect-PHP` library and use the `phpseclib/Crypt/RSA` library instead.

8.4 `R04` Prevent the Use of Uninitialized Values

General Description. The underlying library uses values from the session. However, the session which the library uses can be different from the session the application itself uses. Therefore, it could be possible to complete an authentication flow without providing the session ID required by the library. This would lead to the usage of uninitialized values, which can lead to unintended behavior.

Recommendation. We recommend implementing proper checks for uninitialized values. If a value isn't initialized, i.e., if something goes wrong during the authentication flow, it's recommended to abort the authentication flow.

8.5 `R05` Potentially Insecure XML Parsing of RSA Keys

General Description. In order to verify the RSA signature of an ID token, the client has to obtain the RSA public key. This key can be obtained by accessing the JSON-formated JWKS file provided by the identity authority.

When verifying the RSA signature, the `OpenID-Connect-PHP` library reads the provided JWKS file, extracts the particular RSA key, and puts it directly into the XML key format for "simple" processing.[12] The key is then loaded using the `load` function of the `phpseclib` library.[13]

This type of processing allows a malicious identity authority to inject arbitrary XML contents into the parsed XML structure. Note that we were not able to find any specific attack vector to exploit this feature.

---

[9]https://pear.php.net/package/Crypt_RSA

[10]https://github.com/phpseclib/phpseclib/tree/master/phpseclib/Crypt

[11]https://github.com/pear/Crypt_RSA

[12]See the `verifyRSAJWTsignature` function: https://github.com/jumbojett/OpenID-Connect-PHP/blob/62d557c86d9b2b8607e254064bda400c8fccf656/src/OpenIDConnectClient.php#L809

[13]See: https://github.com/phpseclib/phpseclib/blob/master/phpseclib/Crypt/RSA/Keys/XML.php#L45

Recommendation. Although we were unable to find any practical exploit, we recommend to harden the RSA key parsing process. The `OpenID-Connect-PHP` library should only accept valid RSA keys before injecting their contents into the XML structure.

9 Further Evaluations

In this section, we list further evaluations we conducted in our penetration test. It provides useful information for future security evaluations.

9.1 Binding Between Cookies and the `state` Parameter

As described in M01 , the relying party does not enforce the presence of the `state` parameter. The relying party must also ensure that its value is bound to the user's session in order to prevent CSRF attacks. We verified that if the `state` parameter is present in the request, then its value is correctly bound to the session cookie `fe_typo_user`.

9.2 Changing the OpenID Connect Flow

The relying party utilizes the OIDC code flow (`response_type=code`), to obtain an access token and an ID token. If the hybrid flow (`response_type=code id_token` or `response_type=code token id_token`) is used instead, the login process is still successful. However, the relying party uses the ID token delivered in the back-channel to log in the user, similarly to the code flow. It seems to ignore the tokens delivered in the front-channel completely; delivering an ID token which has expired, contains an invalid signature, or a DENIC ID identifier other than the one specified by the ID token delivered in the back-channel does not result in an error, and seems not to influence the login process in any way.

If the implicit flow (`response_type=id_token` or `response_type=id_token token`) is used instead of the code flow, the login process is not successful. Independently of how the tokens are delivered in the authentication response, (in the query string or the fragment) the login process is not successful and the following error message is displayed: `Login fehlgeschlagen [...] Die eingegebenen Zugangsdaten sind ungültig`. The relying party seems to ignore the tokens delivered in the front-channel and tries to obtain tokens in the back-channel by issuing a token request to the token endpoint of the identity authority, similarly to the code flow. However, the relying party did not receive a code to redeem from the identity authority and the value of the `code` parameter in the token request is an empty string.

9.3 ID Token Validations

9.3.1 Claim Validations

Duplicate Claims. If a claim is present in an ID token more than once, the relying party always uses its second appearance for all validation steps and further processing. The first appearance is ignored. It was not possible to "confuse" the relying party to use one of the appearances for one purpose and the other appearance for another purpose.

`id4me.identifier`. The relying party uses the value of the `id4me.identifier` claim to determine the user identity. It validates whether the value matches the DENIC ID identifier

which the user entered at the beginning of the login process and rejects the ID token otherwise. Injecting different malicious payloads in the value of the `id4me.identifier` claim does not result in successful attacks or verbose error messages. Different cross-site scripting (XSS) and SQL injection (SQLi) payloads all result in the same error message being displayed on the login page: `Login fehlgeschlagen [...] Die eingegebenen Zugangsdaten sind ungültig.`

aud and iss. The relying party enforces that both the `aud` and `iss` claims are present in the ID token which was issued by the identity authority. The value of both claims must not be an arbitrary or empty string but instead needs to be the client ID of the relying party for the `aud`, and the correct issuer property of the identity authority for the `iss` claim. However, as described in `L02`, it is possible to bypass the validation of these claims by setting their value to an integer instead of a string. All tested values besides the expected string or an integer result in an error message similar to this: `Oops, an error occurred! Code: 201906051317374364a8d4`.

sub. According to the OIDC standard [8] an ID token must contain a `sub` claim. However, the relying party does not enforce the presence of a `sub` claim in an ID token. It seems that the relying party ignores the `sub` claim if it is present and accepts ID tokens independently of the presence or value of the `sub` claim. Although this behavior is not compliant to the OIDC standard, it is not classified as a weakness because the `id4me.identifier` is used to identify the user instead.

nbf. ID tokens are JSON web tokens (JWTs). According to the standard [3] a JWT can contain a third timestamp in addition to the `iat` and `exp` claims. The `nbf` claim states a time, before which, the token given must not be accepted for processing. If this claim is present, the relying party validates its value and rejects an ID token if the timestamp states a time in the future. However, the same mistakes during the validation, as described in `L02`, also apply to the `nbf` claim.

at_hash. The OIDC standard defines an optional claim which binds the ID token to an access token by containing a hash value of this access token. If the optional `at_hash` claim is present in an ID token, the relying party validates its value. Otherwise, it displays an error message if the value is an arbitrary or empty string, an integer, or the boolean `false`. However, setting the value to the boolean `true` bypasses the validation and the relying party accepts the ID token. This behavior results from the same mistakes during the validation, as described in `L02`.

9.3.2 Replacing the ID Token in the Token Response

The relying party does not accept values for the ID token which have a type other than string. We evaluated the following values for the ID token in the Token Response:

- `true`

- `1`

- `0`

All tested values result in an error message similar to this: `Oops, an error occurred! Code: 201906061323214e5cc09a.`

9.3.3 Signature Exclusion

- The relying party enforces that ID tokens issued by the identity authority are secured by a valid signature or HMAC. Removing or invalidating the signature or HMAC results in an error being displayed to the user: `Oops, an error occurred! Code: 201906061323214e5cc09a`.

- While the JSON Web Signature standard [2] specifies the usage of `None` as a valid algorithm used for the calculation of the signature, the relying party rejects ID tokens which contain the value `None` in the `alg` header field. The relying party displays the following error message to the user if it receives an ID token using the `None` algorithm: `Oops, an error occurred! Code: 201906061324556ffa5099`.
  Other values for the `alg` header field including `none`, `NONE`, `NoNe`, `plain`, or `test` result in a similar error message.

9.3.4 Key Information

The relying party always requests the JWKS file of the identity authority (located at `https://*iauth-url*/jwks`) and uses the public keys provided in this file to verify the signatures of ID tokens. The following manipulations were conducted to make the relying party use a different key instead:

- Placing a key in the header of the ID token using the `jwk` claim.

- Placing a certificate in the header of the ID token using the `x5c` claim.

- Referencing an external url to access the key in the header of the ID token using the `jku` claim.

- Referencing an external URL to access the certificate in the header of the ID token using the `x5u` claim.

- Referencing an external url to access the key in the JWKS file using the `jku` claim.

- Referencing an external url to access the certificate in the JWKS file using the `x5u` claim.

The relying party does not invoke URLs specified in the `jku` or `x5u` claim and ignores keys/certificates provided in the `jkw` or `x5c` claim. Instead, it always uses the keys directly provided in the JWKS file.

9.4 Covert Redirect

The POST request sent to the relying party (i.e., when the DENIC ID identifier is entered on the login page) contains a parameter called `redirect_url`. The default value of this parameter is `/startseite/`.

We evaluated different manipulated values for the parameter including:

- `/startseite1111/`

- `/startseite/1111/`

- `https://member.secure.denic.de/mydenic/denic-id-verwalten/`

- `@attacker.de/startseite/`

- `attacker.com/startseite/`

- `http://attacker.com/startseite/`

- `/../`

- `/startseite/../`

- `javascript:alert(1)`

In addition, we removed the value of the parameter and the parameter itself.

None of the manipulations was successful; the user was always redirected to `https://member.secure.denic.de/startseite/` at the end of a successful login process and to `https://member.secure.denic.de/denic-id-auth/` if there was an error during the login process.

9.5 Malicious Values for the `state` Parameter

The value of the `state` parameter is initially chosen by the relying party when a new login flow is initiated. It is reflected to the relying party in the authentication response by the identity authority later. Injecting different malicious payloads in the value of the `state` parameter does not result in successful attacks or verbose error messages. Different XSS and SQLi payloads all result in the same error message displayed on the login page: `Login fehlgeschlagen [...] Die eingegebenen Zugangsdaten sind ungültig.`

9.6 TLS Configuration

We tested the TLS configuration of the relying party with testssl.sh and TLS-Scanner. We did not find any configuration issues beyond the repeating ephemeral values summarized in `R01` . The server is not vulnerable to any relevant attack. It only supports TLS 1.2 and secure cryptographic algorithms.

The results of TLS-Scanner are provided in Listing 2.

```
1  Supported Protocol Versions
2
3  TLS12
4
5  _____
6  Supported Ciphersuites
7
8  TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
9  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
10 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
11 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
12
13 _____
14 Symmetric Supported
```

```
15
16  Null    : false
17  Export   : false
18  Anon    : false
19  DES     : false
20  SEED    : false
21  IDEA    : false
22  RC2     : false
23  RC4     : false
24  3DES    : false
25  AES     : true
26  CAMELLIA   : false
27  ARIA    : false
28  CHACHA20 POLY1305 : false
29
30  ————————————————————————————————————————————————
31  KeyExchange Supported
32
33  RSA    : false
34  DH    : false
35  ECDH   : true
36  GOST    : false
37  SRP    : Unknown
38  Kerberos  : false
39  Plain PSK  : false
40  PSK RSA    : false
41  PSK DHE   : false
42  PSK ECDHE  : false
43  Fortezza  : false
44  New Hope   : false
45  ECMQV    : false
46
47  ————————————————————————————————————————————————
48  Perfect Forward Secrecy
49
50  Supports PFS  : true
51  Prefers PFS   : true
52  Supports Only PFS  : true
53
54  ————————————————————————————————————————————————
55  Cipher Types Supports
56
57  Stream   : false
58  Block   : true
59  AEAD    : true
60
61  ————————————————————————————————————————————————
62  Ciphersuite General
63
64  Enforces Ciphersuite ordering : true
65
66  ————————————————————————————————————————————————
67  Supported Extensions
68
69  EC_POINT_FORMATS
70  EXTENDED_MASTER_SECRET
71  RENEGOTIATION_INFO
72
73  ————————————————————————————————————————————————
74  Supported Named Groups
75
76  SECP256R1
77  ECDH_X25519
78  SECP384R1
79
80  ————————————————————————————————————————————————
81  Supported Compressions
```

```
 82
 83  NULL
 84
 85  ——————————————————————————————————————————————
 86  Common Bugs [EXPERIMENTAL]
 87
 88  Version Intolerant  : false
 89  Ciphersuite Intolerant  : false
 90  Extension Intolerant  : false
 91  CS Length Intolerant (>512 Byte) : false
 92  Compression Intolerant  : false
 93  ALPN Intolerant   : false
 94  CH Length Intolerant  : false
 95  NamedGroup Intolerant  : false
 96  Empty last Extension Intolerant : false
 97  SigHashAlgo Intolerant  : false
 98  Big ClientHello Intolerant : false
 99  2nd Ciphersuite Byte Bug : false
100  Ignores offered Ciphersuites : false
101  Reflects offered Ciphersuites : false
102  Ignores offered NamedGroups : false
103  Ignores offered SigHashAlgos : false
104
105  ——————————————————————————————————————————————
106  Attack Vulnerabilities
107
108  Padding Oracle   : false
109  Bleichenbacher   : false
110  CRIME    : false
111  Breach    : false
112  Invalid Curve   : false
113  Invalid Curve Ephemerals : false
114  SSL Poodle   : false
115  TLS Poodle   : false
116  CVE—20162107   : false
117  Logjam    : false
118  Sweet 32   : false
119  DROWN    : false
120  Heartbleed   : false
121  EarlyCcs   : false
122
123  ——————————————————————————————————————————————
124  RFC
125
126  Checks MAC (AppData)  : correct
127  Checks MAC (Finished)  : correct
128  Checks VerifyData   : correct
129
130  ——————————————————————————————————————————————
131  Certificates
132
133  Fingerprint   : f9caa681bc02bbfe4b4183137db0190d4355823de0ca285fb3e34f978aac01a6
134  Subject    : SERIALNUMBER=GnR 770,1.3.6.1.4.1.311.60.2.1.3=DE,BusinessCategory=Private Organization,C=DE,PostalCode
             =60329,ST=Hessen,L=Frankfurt am Main,STREET=Kaiserstr. 75—77,O=DENIC eG,OU=IT Services,OU=Authorized by
             United SSL,OU=COMODO EV SSL,CN=member.secure.denic.de
135  CommonNames   : #311f301d060355040313166d656d6265722e7365637572652e64656e69632e6465
136  Valid From   : Wed Jan 17 01:00:00 CET 2018
137  Valid Till   : Thu Feb 06 00:59:59 CET 2020
138  PublicKey   : RSA Public Key [68:ed:4e:f1:7c:1d:ce:41:7f:bf:7d:7d:40:9f:73:82:e2:ad:3b:c1]
139  modulus: d37aab82cf4fac3cf5b28f8c34a651bf36f572bbe687dde56d46919e9d29b05b1c855904aec2aee6948574b
             7731657099f32fa2d6576a8991459171e85987b9bbe2f030aba3ddebf59835783be7dbe35a50c7285279e5aa4bea6f
             952506ea2fe5e70e1e6f027748c93c5976361f63360066d6f18de9c488abaeccfa43535af3471e4cfb69497a66826a67
             aedcffb32c3d2fe33315b882e1ca1a0f4f5a479c1c768bac79138826e828cb32a86bc89e88e641363217841ada7067f
             82bcd307984230a5ba181f9068855302bd6e1f791db5de311b5b74d40e8648755d2bad14b18efb8b78bf4eee59b
             370593c58afe4bcd5e90e9de209914baa9d381c6ba002d0b7
140  public exponent: 10001
141
```

```
142  Issuer   : C=GB,ST=Greater Manchester,L=Salford,O=COMODO CA Limited,CN=COMODO RSA Extended Validation Secure
          Server CA
143  Signature Algorithm : RSA
144  Hash Algorithm   : SHA256
145  ROCA (simple)   : false
146  Fingerprint   : 7e0e16c0056f41a9f4c61f571503c3bcf079e2bddb228bf2219ac31200496b5c
147  Subject   : C=GB,ST=Greater Manchester,L=Salford,O=COMODO CA Limited,CN=COMODO RSA Extended Validation Secure
          Server CA
148  CommonNames   : #3138303606603550403132f434f4d4f444f2052534120457874656e6465642056616c69646174696f6e
          2053656375726520536572766572204341
149  Valid From   : Sun Feb 12 01:00:00 CET 2012
150  Valid Till   : Fri Feb 12 00:59:59 CET 2027
151  PublicKey   : RSA Public Key [a2:26:20:54:4e:0a:e3:47:b6:74:41:da:e5:2f:ae:9d:01:12:54:d9]
152  modulus: 9556de54b4dfd502497bd15b5ca2b21e8f9c2b624c2b8d1228f31a95a3c610fd29dee19f0b384093d1ef6e9510fce
          19017772cee753e7b63ec61926e4f3bab80496bdf00ea03007f2f75d5282fec56678f8083a3bddc0399938b9491565
          ba1b86a3a3f06bd0e92cc609cfdb5e09f66305fdbe694f0956aafc88aaf80d9e68839017c1cc0c52af77b95a0f276ab6d9
          b723930ebd15755019d58119d7c6d848f49e89d09fc3cfd0a4a7614215c167340231974c3ba580aa6962ede36e59fd0
          c2f0e1e0c162e3c218451951aa171ee82375d4c8d09613ffc724d18c0b27ae9e7adc3a61636088972d5d050be53
          bebaece3a477376a8fa2cddc08717e9ac3099f81f
153  public exponent: 10001
154
155  Issuer   : C=GB,ST=Greater Manchester,L=Salford,O=COMODO CA Limited,CN=COMODO RSA Certification Authority
156  Signature Algorithm : RSA
157  Hash Algorithm   : SHA384
158  ROCA (simple)   : false
159  Fingerprint   : 4f32d5dc00f715250abcc486511e37f501a899deb3bf7ea8adbbd3aef1c412da
160  Subject   : C=GB,ST=Greater Manchester,L=Salford,O=COMODO CA Limited,CN=COMODO RSA Certification Authority
161  CommonNames   : #312b302906603550403132243f4d4f444f2052534120436572746966696361746f6e20417574686f
          72697479
162  Valid From   : Tue May 30 12:48:38 CEST 2000
163  Valid Till   : Sat May 30 12:48:38 CEST 2020
164  PublicKey   : RSA Public Key [2e:30:a8:20:a9:7e:d4:33:04:78:84:53:7d:4d:c1:5d:0d:0d:6f:04]
165  modulus: 91e85492d20a56b1ac0d24ddc5cf446774992b37a37d23700071bc53dfc4fa2a128f4b7f1056bd9f7072b7617fc94
          b0f17a73de3b00461eeff1197c7f4863e0afa3e5cf993e6347ad9146be79cb385a0827a76af7190d7ecfd0dfa9c6cfadfb
          082f4147ef9bec4a62f4f7f997fb5fc674372bd0c00d689eb6b2cd3ed8f981c14ab7ee5e36efcd8a8e49224da436b62b
          855fdeac1bc6cb68bf30e8d9ae49b6c6999f878483045d5ade10d3c4560fc32965127bc67c3ca2eb66bea46c7c720a0b
          11f65de4808baa44ea9f283463784ebe8cc814843674e722a9b5cbd4c1b288a5c227bb4ab98d9eee05183c309464e6
          d3e99fa9517da7c3357413c8d51ed0bb65caf2c631adf57c83fbce95dc49baf4599e2a35a24b4baa9563dcf6faaff4958
          bef0a8fff4b8ade937fbbab8f40b3af9e843421e89d884cb13f1d9bbe18960b88c2856ac141d9c0ae771ebcf0edd3da996
          a148bd3cf7afb50d224cc01181ec563bf6d3a2e25bb7b204225295809369e88e4c65f191032d707402ea8b
          671529695202bbd7df506a5546bfa0a328617f70d0c3a2aa2c21aa47ce289c064576bf821827b4d5aeb4cb50e66bf44
          c867130e9a6df1686e0d8ff40ddfbd042887fa3333a2e5c1e41118163ce18716b2beca68ab7315c3a6a47e0c37959d
          6201aaff26a98aa72bc574ad24b9dbb10fcb04c41e5ed1d3d5e289d9cccbfb351daa747e58453
166  public exponent: 10001
167
168  Issuer   : C=SE,O=AddTrust AB,OU=AddTrust External TTP Network,CN=AddTrust External CA Root
169  Signature Algorithm : RSA
170  Hash Algorithm   : SHA384
171  ROCA (simple)   : false
172
173  ————————————————————————————————————————————————————————————
174  Certificate Checks
175
176  Expired Certificates  : false
177  Not yet Valid Certificates : false
178  Weak Hash Algorithms  : false
179  Weak Signature Algorithms  : false
180  Matches Domain   : Unknown
181  Only Trusted   : Unknown
182  Contains Blacklisted  : Unknown
183
184  ————————————————————————————————————————————————————————————
185  HSTS
186
187  HSTS   : true
188  HSTS Preloading   : false
189  max—age (seconds) : 15552000
```

```
190
191 ——————————————————————————————————————————————————————
192 HTTPS Response Header
193
194 Date:Mon, 10 Jun 2019 20:38:18 GMT
195 Server:Apache
196 Location:https://member.secure.denic.de/startseite/
197 Content−Length:249
198 Content−Type:text/html; charset=iso−8859−1
199 Content−Security−Policy:style−src 'self' 'unsafe−inline' https://fonts.googleapis.com; object−src 'self'; script−src 'self' https://
        www.googletagmanager.com https://www.google−analytics.com 'unsafe−inline'; img−src 'self' https://www.denic.de
        https://www.google−analytics.com; frame−src 'self'
200 X−Content−Security−Policy:style−src 'self' 'unsafe−inline' https://fonts.googleapis.com; object−src 'self'; script−src 'self'
        https://www.googletagmanager.com https://www.google−analytics.com 'unsafe−inline'; img−src 'self' https://www.
        denic.de https://www.google−analytics.com; frame−src 'self'
201 X−XSS−Protection:1; mode=block
202 X−Content−Type−Options:nosniff
203 X−Frame−Options:SAMEORIGIN
204 X−Varnish:18417071
205 Age:0
206 Via:1.1 varnish−v4
207 Strict−Transport−Security:max−age=15552000
208 Connection:keep−alive
209
210 ——————————————————————————————————————————————————————
211 PublicKey Parameter
212
213 EC PublicKey reuse  : true
214 DH PublicKey reuse  : false
215 Uses Common DH Primes  : false
216 Uses Non−Prime Moduli  : false
217 Uses Nonsafe−Prime Moduli : false
```

Listing 2: An excerpt of the TLS-Scanner scan report for `member.secure.denic.de`

10 References

[1] Vittorio Bertola. *ID4me Technical Overview*. https://id4me.org/documents/. June 2018.

[2] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Signature (JWS)*. RFC 7515 (Proposed Standard). Internet Engineering Task Force, May 2015. url: https://tools.ietf.org/html/rfc7515.

[3] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT)*. RFC 7519 (Proposed Standard). Internet Engineering Task Force, May 2015. url: https://tools.ietf.org/html/rfc7519.

[4] P.V. Mockapetris. *Domain names - concepts and facilities*. RFC 1034 (Proposed Standard). Internet Engineering Task Force, Nov. 1987. url: https://tools.ietf.org/html/rfc1034.

[5] OWASP. *Cross-Site Request Forgery (CSRF)*. May 2016. url: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[6] OWASP. *Session Management Cheat Sheet*. https://www.owasp.org/index.php/Session_Management_Cheat_Sheet.

[7] N. Sakimura, J. Bradley, and M. Jones. *OpenID Connect Dynamic Client Registration 1.0*. OpenID Foundation, Nov. 2014. url: http://openid.net/specs/openid-connect-registration-1_0.html.

[8] N. Sakimura et al. *Openid connect core 1.0*. OpenID Foundation, Nov. 2014. url: http://openid.net/specs/openid-connect-core-1_0.html.