UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Jan Erik Kriisk

Security analysis of RIA's authentication service TARA

Bachelor's Thesis (9 ECTS)

Supervisor: Arnis Paršovs, PhD

Tartu 2021

Security analysis of RIA's authentication service TARA

Abstract:

Estonian Information System Authority (Riigi Infosüsteemi Amet - RIA) governs an authentication service called TARA. Many public sector e-services use the authentication service to authenticate users via ID-card, Mobile-ID, Smart-ID, or EU eID. The thesis aims to analyse the security of TARA, document the protocol, and analyse what could go wrong.

Keywords:

TARA, authentication, security analysis

CERCS:

P170 Computer science, numerical analysis, systems, control

RIA autentimisteenuse TARA turbeanalüüs

Lühikokkuvõte:

Eesti Infosüsteemi Amet haldab autentimisteenust TARA. Paljud avaliku sektori e-teenused kasutavad autentimisteenust kasutajate autentimiseks ID-kaardi, Mobiil-ID, Smart-ID või EU eID kaudu. Lõputöö eesmärk on analüüsida TARA turvalisust, dokumenteerida protokoll ja analüüsida, mis võib valesti minna.

Võtmesõnad:

TARA, autentimine, turbeanalüüs

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

Terms and Notations

RIA – Estonia Information System Authority

TARA – RIA's authentication service

URL – Uniform Resource Locator

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

TLS – Transport Layer Security

GET – HTTP method that is used to request data from a specified resource

POST – HTTP method is used to send data to a server to create/update a resource

PHP – General-purpose scripting language that is suited for web development

cURL – Command line tool and library for transferring data using various network protocols

JSON – Open standard file and data interchange format

JWT – JSON Web Token

1 Introduction

Estonian Information System Authority (Riigi Infosüsteemi Amet - RIA) provides an authentication service called TARA. Many public sector e-services use the authentication service to authenticate users via ID-card, Mobile-ID, Smart-ID, or EU eID [1]. It was introduced by the Estonian Information System Authority in 2017 to create a standard solution for logging in to the public sector e-services and provide a uniform user experience to avoid parallel developments [1]. The TARA authentication service is based on the OpenID Connect protocol [2], based on the OAuth 2.0 protocol [3]. It is a centralised system, meaning that if the authentication protocol is abused, it could give an attacker access to much sensitive information of an arbitrary user.

The thesis aims to analyse the security of TARA, document the protocol and analyse what could go wrong. To experiment with the protocol, a TARA test account was obtained, and a mock service implemented. The mock service was used to understand the system better and then draw conclusions from the service and the official documentation. The mock service was built using HTML and PHP to send and receive HTTP GET and POST requests and utilised command-line cURL to transfer the HTTP identity token request between the TARA server and the mock service (see Appendix I. The mock applications HTML and PHP code). The service was built on the domain of http://kodu.ut.ee/~janerikr/TestTARA/.

1.1 Authentication process from a user's perspective

The process described is an authentication process with TARA from the perspective of the user. Some things may differ depending on the application or service being used, method of authentication, and/or the user being authenticated.

When the user opens up a website and navigates to the login page, the browser redirects the user to the TARA service by authentication request. A selection of authentication methods is shown to the user:

1) ID card – the state-issued ID
2) mobile ID – allows people to use a mobile phone as a form of secure digital ID.
3) Cross-border (eIDAS-Node) authentication – allows for other EU citizens to authenticate themselves.

4) Smart-ID – a new and convenient mobile application that works as an identification solution.

After an authentication method is selected by the user, and depending on the selection, a few more steps might occur, where the user must either enter valid credentials or submit other data. In the case of successful authentication at the TARA service, the user is redirected back to the website they were before, and the user is notified of a successful login. In case of an error, the user is sent back and given the option to try a different authentication method or terminate the authentication process.

1.2 Registration procedure on becoming a user of TARA

The government institutions that need TARA are service providers who wish to use cross-border authentication and use authentication methods without implementing these themselves. For an institution to join, they must submit an application to RIA detailing the information about their service [1]:

- Information about the institution: name, address, phone number, email, registry number;
- Information about the contact person of the institution: name, phone number, email, personal identification code;
- TARA purpose of use;
- Name of the client application;
- Description of the client application;
- Authentications methods the service provider would like to use;
- Estimated number of users per month;
- The client application identifier (`client_id`);
- The client application redirect-URL (`redirect_uri`).

The applications to join the authentication service can be found at https://www.ria.ee/et/riigi-infosusteem/eid/partnerile.html. When RIA approves the application, it will send the service provider an email containing a `client_secret` value. This value is used to authenticate the service provider to TARA when an identity token request is sent (see section 2.1.3).

2 Authentication Protocol

The Estonia Information System Authority states that the main parties of the protocol flow are the browser, the server component of the client application, and the server component of TARA. The browser is what the user interacts with, where the user enters necessary information and where the results of each request and step is shown. The client application runs in the service the user would like to authenticate. TARA, the server component hosted by RIA, handles the HTTP requests, and the client application uses it to authenticate the user trying to authenticate into the service. (see Figure 1)

All of the HTTP requests mentioned in the authentication protocol will be described in more detail in a separate section later on. The protocol description presented in this section is based on the TARA technical specification available at [4].

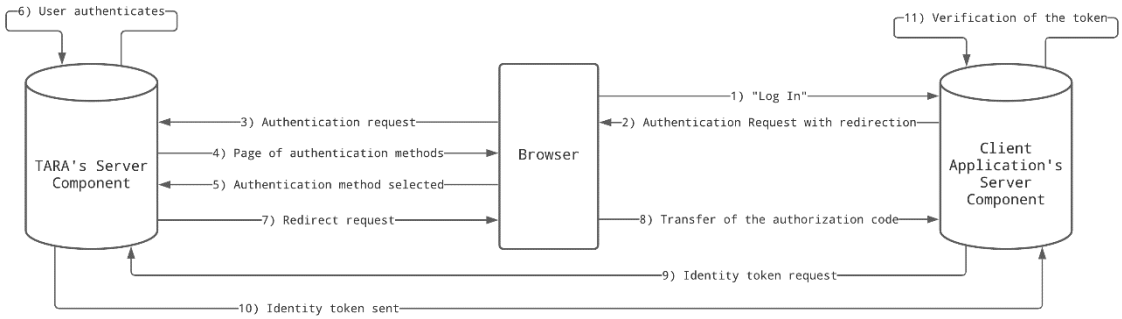

Figure 1. Diagram of the authentication process

The protocol flow begins with the user's browser loading a login page of some service provider (TARA client), and the user clicks the login button. The browser sends an HTTP request to the service provider's server, as a response to which the service provider creates an authentication request and redirects the user's browser with the request to the TARA server. When TARA receives the request, it generates a page with authentication options and shows it in the user's browser (see Figure 2). The user then selects an authentication method (e.g., ID-card, Mobile-ID, Smart-ID, or EU eID) and performs authentication at TARA.

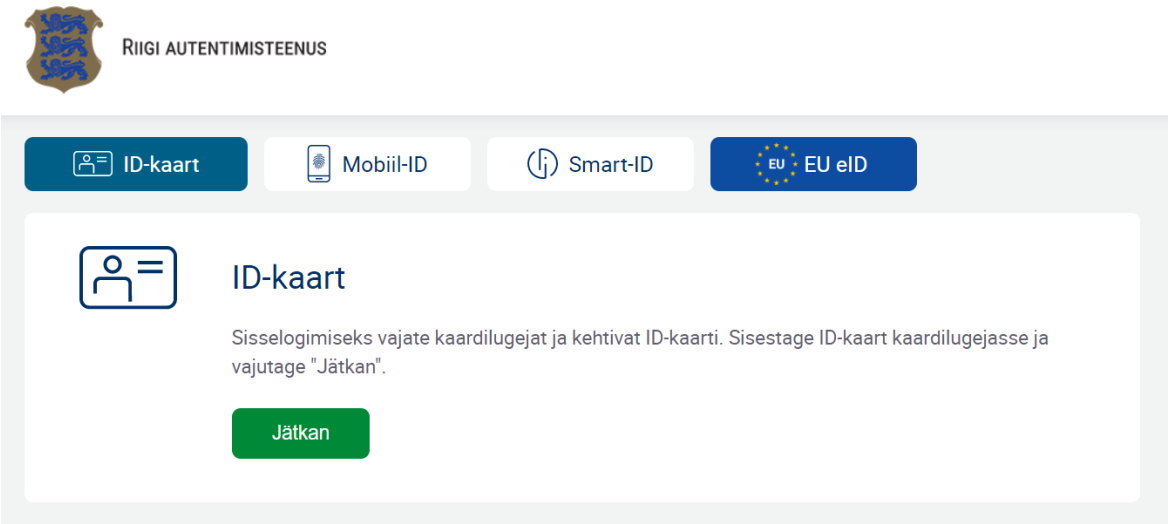

Figure 2. The authentication methods provided by TARA when trying to log in at eesti.ee

After the user has successfully authenticated at TARA, the user's browser is redirected back to the service provider. In case of successful authentication, the redirect request contains the authorisation code. After the service provider receives the redirect request, the service provider sends an identity token request to TARA, containing the authorisation code and a client secret code to authenticate the client. This request is a backend request, so it is not sent through the browser but directly between the service provider's server and TARA. TARA verifies the client's secret and issues a signed identity token in response. The application server verifies whether the token was signed by TARA and is not expired. Finally, the service provider creates an authenticated browser session with the user.

2.1 Protocol Requests

There were mentions of different HTTP requests sent between the browser, application server, and TARA server in the authentication protocol flow. This section will give a summary of each of the requests for a better understanding of the process.

2.1.1 Authentication request

The authentication request is an HTTP GET request that is sent to TARA to initiate an authentication process. The authentication request is constructed by the client application and sent to the user's browser, where it initiates an authentication process by receiving an HTTP redirect from the service provider. This request is sent when a user initiates the "log in" process. After a successful user authentication at TARA, a redirection request (see Section 2.1.2) containing an authorisation code will be sent to the client application.

```
HTTP GET https://tara.ria.ee/oidc/authorize?

redirect_uri= http%3A%2F%2Fkodu.ut.ee%2F~janerikr%2FTestTARA%2F&

scope=openid&

state=
YTdhZTUyNzExNjM2OWUzZWZjYjMyZjliN2QwMzI4YjU5NDA3OWNmNjljOTkwZWMxZDM0MzY1NWM0O
GZjNWEwOA==&

response_type=code&

client_id= ut_bsc_testing_service
```

Figure 3. Example of an authentication request.

The HTTP request comprises of multiple elements (see Figure 3) [2]:

- `redirect_uri` – the redirect URL, specifying the service provider's URI where the request containing the authorisation code should be sent;
- `scope` – specifies the allowed TARA authentication methods (e.g., '`idcard`', '`mid`', '`smartid`', '`eidas`'). The authentication scope comprises of a compulsory '`openid`' value and of the authentication method or methods used, which are all separated with a SPACE character URL encoded;
- `state` – a security code to protect against cross-site request forgery. The value in this variable will be reflected in the redirect request;
- `response_type` – for TARA, the supported value is '`code`', which, by the OpenID Connect protocol, refers to the authorisation code flow;
- `client_id` – used to identify from which service provider (TARA client) the request comes from. Each service provider is assigned a unique client_id when the service provider's application is registered as a TARA user.

While there are more parameters in the authentication request, the ones above are compulsory. The parameters '`ui_locales`', '`nonce`' and '`acr_values`' are optional. A successful authentication request is followed by a user authenticating themselves at TARA, and then a redirect request is initiated by the TARA server.

2.1.2 Redirect request

The redirect request is an HTTP GET request, which is sent to the service provider's `redirect_uri` specified in the authentication request, which is used to redirect the user

back to the application from TARA. It is sent to the client application when the TARA server successfully processes the previous authentication request and the user has finished authenticating with TARA.

```
HTTP GET http://kodu.ut.ee/~janerikr/TestTARA/?

code= OC-710--CVfXs2eiWFJx5Eqf8Shq7dM8Mx-Z-LI&

state=
YTdhZTUyNzExNjM2OWUzZWZjYjMyZjliN2QwMzI4YjU5NDA3OWNmNjljOTkwZWMxZDM0MzY1NWM0O
GZjNWEwOA==
```

Figure 4. Example of a redirect request.

The HTTP request comprises of two elements (see Figure 4):

- `code` – the authorisation code, which can be used to receive the user's identity token;
- `state` – the same value that was included in the authentication request.

The redirect request redirects the user back to the application. After the redirect request is received, the service provider can use the authorisation code to obtain an identity token from TARA.

2.1.3 Identity token request

The identity token request is an HTTP POST request made by the service provider to obtain an identity token from TARA. The authorisation code is sent directly to TARA, which results in an identity token being returned by TARA. The request has to be sent in 30 seconds after being issued by TARA the authorisation code, which can only be sent once.

```
POST /token HTTP/1.1

Host: tara.ria.ee/oidc/token

Content-Type: application/x-www-form-urlencoded

Authorization: Basic
dXRfYnNjX3Rlc3Rpbmdfc2VydmljZTpOdTdrV3lwV3pGT2E4N3FYdFhtczJEeV8=


grant_type=authorization_code&

code= OC-710--CVfXs2eiWFJx5Eqf8Shq7dM8Mx-Z-LI&

redirect_uri= http%3A%2F%2Fkodu.ut.ee%2F~janerikr%2FTestTARA%2F
```

Figure 5. Example of an identity token request.

The service provider must authenticate the request by including the client secret code in the Authorisation request header. The Authorisation request header of the identity token request comprises of the word '`Basic`', a space, and a string in the format of `<client_id>:<client_secret>` encoded to Base64 (see Figure 6).

```
client_id = ut_bsc_testing_service
client_secret = Nu7kWypWzFOa87qXtXms2Dy_

client_secret_code = Base64(client_id + ":" + client_secret)
```

Figure 6. Example of a client secret code calculation

The body of the HTTP request comprises of multiple elements (see Figure 5) [3]:

- `grant_type` – the value required based on the protocol, which is hardcoded to '`authorisation code`';
- `code` – the authorisation code received from TARA in the redirect request
- `redirect_uri` – the `redirect_uri` value of the service provider.

The values in the HTTP POST body must be form URL encoded. After TARA has verified that the identity token request is sent by an authorised service provider, TARA checks whether the identity token corresponding to the authorisation code has been issued in an authentication process to the service provider that is sending the identity token request. TARA also checks whether the code has not expired (30 seconds) and has not been requested before. TARA issues the identity token to the client application in an HTTP response body.

2.1.3.1 Identity token response

The identity token is a certificate issued by the TARA authentication service as a result of the identity token request. The token is issued in JSON Web Token (JWT) format and returned in an HTTP response body.

The token is a JSON serialised data structure that contains four fields (see Figure 7):

- `access_token` – OAuth 2.0 access code can be used to obtain the user's data via the user info request.
- `token_type` – OAuth 2.0 access token type set to value '`bearer`'. The value is returned by TARA but does not need to be processed by the service provider, and it is a mandatory value by the specifications of the OAuth 2.0 protocol.
- `expires_in` – the lifetime of the access token provided in seconds. The value is returned by TARA but does not need to be processed by the service provider, and it is a mandatory value by the specifications of the OAuth 2.0 protocol.
- `id_token` – the JWT identity token, encoded in Base64 format.


```
{
"access_token":"AT-328-jnKC3iVMmegbpdu05HZzzh899XABzu-c",
"token_type":"bearer",
"expires_in":600,
"id_token":"eyJhbGciOiJSUzI1NiIsImtpZCI6InB1YmxpYzp4V2JiVm9ZcTlFd01xcGhwIn0.e
yJqdGkiOiJiZTA1NzhkYy03NjdhLTRmMzAtOTAxOC0wODhiMjgwMmI0ZjMiLCJpc3MiOiJodHRwcz
ovL3RhcmEtdGVzdC5yaWEuZWUiLCJhdWQiOiJ1dF9ic2NfdGVzdGluZ19zZXJ2aWNlIiwiZXhwIjo
xNjE3NzA2NDIzLCJpYXQiOjE2MTc3MDU4MjMsIm5iZiI6MTYxNzcwNTUyMywic3ViIjoiRUUxMDEw
MTAxMDAwNSIsInByb2ZpbGVfYXR0cmlidXRlcyI6eyJkYXRlX29mX2JpcnRoIjoiMTgwMS0wMS0wM
SIsImZhbWlseV9uYW1lIjoiU01BUlQtSUQiLCJnaXZlbl9uYW1lIjoiREVNTyJ9LCJhbXIiOlsic2
1hcnRpZCJdLCJhY3IiOiJoaWdoIiwic3RhdGUiOiJZVGRoWlRVeU56RXhOak0yT1dVelpXWmpZak1
5WmpsaU4yUXdNekk0WWpVNU5EQTNPV05tTmpsak9Ua3daV014WkRNME16WTFOV00wT0daak5XRXdP
QT09Iiwibm9uY2UiOiIiLCJhdF9oYXNoIjoiOVVaYnNNNXNNaWdZRlAzb2J4b291UT09In0.tvPgn
gqBaiYWTgSsvP5trvr7sC5SpSFkauMOv-v8D-
Ay0ouXHAtj_DLkZPjpFlK3hxxQfFDKUw3o4hZyT5T5_AbOEn30mapmgWivE4EdKFgY-
BP8q_C9CkxvongMgBWSR9ODyIXndzaNFwx_PJNLu_g1atccR3dRk7pJzqVMiW2AiJZlqgHqchYfaE
W8JYvqmb4gUV2EOeM79ZMbb4WqT2GpM-
QeJ0UhBqbQrrZm0Z8ZgkpZlrfufGtsSDjQ20cu4m4IjqO9yxE6kQVcXg1fu08k0afiYoD_zuF115G
eg3Cq2ESpGSIywQB_12z1iQnKQztDFX0pzRmpkyz7vdBcKcUJBLtYI6cQJuxDF8L1WCbw2jyul-
q4Y1935-LbG0_NpX3ZuDHtAtLD_m0mW_1bbeqJnkQGevvb-
qWjUVOKT1Puvx2imzLMU_5GH2MBcVQRVjmWMhzxDgAhfILRHsMAAbi8E_b4DtKJHDfq6GReKf9Vod
62PEeMR0XZKawAttrIHg1hn32FA1_yZpG2x2Q-
f4r44kEb6EkjfWn0f98oy11e4HiCP55DMpkLt4wvqEIK0F2TjLDVK8pfjXg_HsQNE3bqI_TRHJH5r
nthFBe5wO-dmBukhHdzcW8eRQAaPQx49gHIf_6mhPbFbygvERUvQjZOuWVfV0AWaEiPr8qwQQw"
}
```


Figure 7. Example of an HTTP response body of identity token request.

The identity token is encoded as JWT. The JWT contains three fields that are split by a period (‘.’) character. The three parts are header, payload, and signature, all of which are separated by period (‘.’) characters.

The header consists of two fields: ‘`alg`’ and ‘`kid`’. The field ‘`alg`’ contains the signature algorithm, which in the case of TARA is set to ‘`RSA256`’, the RS256 signature algorithm, RSA Signature with SHA-256 (see Figure 8). The field ‘`kid`’ contains the public signature key identifier.


```
{
  "alg": "RS256",
  "kid": "public:xWbbVoYq9EwMqphp"
}
```


Figure 8. The example identity token’s header from Figure 5 (decoded).

The payload is the second part of the token (see Figure 9). A payload contains so-called claims that are statements that give information about the authenticated user and the authentication process. [5]


```
{
  "jti": "be0578dc-767a-4f30-9018-088b2802b4f3",
  "iss": "https://tara-test.ria.ee",
  "aud": "ut_bsc_testing_service",
  "exp": 1617706423,
  "iat": 1617705823,
  "nbf": 1617705523,
  "sub": "EE10101010005",
  "profile_attributes": {
    "date_of_birth": "1801-01-01",
    "family_name": "SMART-ID",
    "given_name": "DEMO"
  },
  "amr": [
    "smartid"
  ],
  "acr": "high",
  "state":
"YTdhZTUyNzExNjM2OWUzZWZjYjMyZjliN2QwMzI4YjU5NDA3OWNmNjljOTkwZWMxZDM0MzY1NWM0
OGZjNWEwOA==",
  "nonce": "",
  "at_hash": "9UZbsM5sMigYFP3obxoouQ=="
}
```


Figure 9. The example identity token’s payload from Figure 5 (decoded).

The signature is the third part of the identity token and verifies the integrity of the token. The signature is created by signing the Base64-encoded header and payload concatenated together using a period (‘.’) character and merged (see Figure 10). The signature is made by TARA using the algorithm specified in the header and private key that corresponds to the

key identifier specified in the header. In the case of TARA, the RS256 signing algorithm is used.

```
RSASHA256(
        base64UrlEncode(header) + "." +
        base64UrlEncode(payload),
        private-key
)
```

Figure 10. The signing of identity token

3 Security analysis

This section contains the security analysis of the TARA authentication protocol. The protocol was analysed by looking at the technical specification [4], implementing a demo service, analysing the implemented security features and the security-related checks that have to be implemented by service providers according to the TARA technical specification. The security of the authentication protocol implementation on the TARA server was not in the scope of the analysis.

Security critical considerations for the security analysis:

1. What are the critical assets that have to be protected?
2. What if the `client_id` and/or `redirect_uri` variables are not checked?
3. What if an attacker could add/modify valid `redirect_uri` for a client?
4. What if the service provider does not check the state variable?
5. What if an attacker would have access to a specific authorisation code of a service?
6. What if an attacker would have access to a `client_secret` value?
7. What if the client application does not verify an identity token's signature?
8. What if the client application does not verify an identity token's claims?
9. What if an attacker would have the private keys of RIA used to decrypt the communication between the client application and TARA endpoint?
10. What if a malicious user would have the private key of RIA used to sign identity tokens?
11. What if a malicious user could perform a man-in-the-middle attack, impersonating tara.ria.ee for user connections or client's connections to obtain identity token/access token?
12. What if a malicious user could perform a cross-site request forgery?
13. What if a malicious user could perform a phishing attack?

The entire integrity of the authentication protocol relies on the fact that some essential values and assets are protected, either by one specific party or two parties, between whom the protocol is taking place:

- TARA private keys – used for signing identity tokens and encrypting communications between it and the client applications. Only TARA knows these.
- Client secret – a unique password for each client application, who are users of the authentication service. Both TARA and the respective holder of the secret know this.

- Original nonce of the state value – used to protect against cross-site request forgery. Only the client application knows this.
- Authorisation code – used to receive the identity token via the identity token request. The code is issued by TARA and sent in the redirect request to the client application

3.1 Authorisation code and the client secret

The client_secret is a client application password issued by RIA to the client application once the registration application has been submitted and approved. The client secret code is included in the identity token request and used by the server to authenticate the service provider.

The authorisation code can be exchanged for an identity token. The authorisation code has a one-time use and, by TARA specifications, must be sent within 30 seconds. These are used to mitigate the replay of tokens, token leakages, and online guessing (see Section 5.1.5.3 [6]). There are mainly two reasons why authorisation codes are sent in redirect requests instead of the tokens themselves (see Section 3.4 [6]):

1. Browser-based processes are vulnerable to attackers since they expose protocol parameters to attacks like replay attacks and web parameter tampering attacks. The one-time codes are used to minimise this threat so that the tokens can be received over a more secure connection between the service provider and the TARA server;
2. It is easier and safer to authenticate clients during a direct request, meaning an identity token request, between a client and an authorisation server than in an indirect authorisation request, which can have the potential of being a malicious HTTP request.

Using authorisation code flow mitigates the most considerable risk of the implicit code flow. If the identity token private key is stolen or obtained, then an attacker could create valid tokens. However, in the authorisation code flow, having the private key is not enough as the attacker needs man-in-the-middle capability between the service provider and TARA.

Authorisation code and the `client_secret` are two essential values that ensure that the identity token request is submitted to TARA by the correct client application.

In the event the `client_secret` of a client application is obtained, then an attacker could issue valid identity token requests to TARA. However, without knowing a specific authorisation code issued to that specific client application, an attacker could not obtain an

identity token. Guessing an authorisation code is not feasible either. The code contains 36 random characters, which by current cryptography standards, is relatively safe.

As previously mentioned, a `client_secret` can obtain nothing if an attacker has not obtained an authorisation code. A similar thing could be said in reverse if an attacker had obtained an authorisation code. Without the `client_secret` of the application the code was issued for; an attacker could not submit a valid identity token request to TARA. However, an authorisation code can also be sent via redirect request. Let us consider an attack, where an attacker has a legitimate website, `attacker.com`, and is a client of TARA. The attacker in his browser opens `eesti.ee`, clicks the "login" button, and gets redirected to the TARA authentication page. The attacker does not allow his browser to connect to TARA and extracts `eesti.ee` 'state' variable from the authentication request. Then, when a victim clicks on the "login" button on `attacker.com`, the attacker redirects the victim's browser to the TARA authentication page via authentication request with the `attacker.com` `client_id` and `redirect_uri` but using the `eesti.ee` 'state' variable. Victim authenticates to TARA and with a valid 'code' is redirected to the attacker's website. The attacker will not contact TARA to obtain the identity token corresponding to that 'code' but instead sends the 'code' as via redirect request to `eesti.ee` with `eesti.ee` 'state' variable. Since the 'state' is valid for the attacker's browser and 'code' has not been used, the service provider `eesti.ee` obtains the identity token from TARA that contains the victim's data, but actually, the identity token was issued for attacker.com and not for eesti.ee. To prevent this attack, TARA must bind every authorisation code to the client application `client_id` and `redirect_uri` that issued the authentication request (see Sections 5.2.4.4 and 5.2.4.5 [6]). This binding prevents the use of authorisation codes for services that the code was not issued for.

3.2 Client Identifier

The client identifier (`client_id`) is the identifier issued to the institution by RIA when the service provider is registered as a user of TARA [4]. TARA does not accept authentication requests and identity token requests that were made with an invalid client identifier. The same happens when the client identifier is missing. OAuth 2.0 demands the use of the client identifier in authorisation scenarios where the client can act without the need of a mechanical action by the user, like when an authorisation code is being exchanged for an identity token (see section 3.7 [6]).

TARA may also use the client identifier to limit the number of authentication requests sent out to TARA for a particular `client_id` (see section 3.7 [6]). This method can protect against possible denial-of-service attacks, aiming to overload the authentication service with excessive authentication requests with a specific `client_id`.

3.3 Redirect URI and its necessity in HTTP requests

The redirect URI (`redirect_uri`) specifies where the user's browser should be redirected together with the authorisation code after the authentication request has been sent to TARA, and the user has finished authenticating with TARA. An authorisation server should require their clients to register their client applications where the applications `redirect_uri` is validated against the `redirect_uri` in the authorisation request. (see Section 3.5 [6])

The binding of the authorisation code to the `client_id` brings up the question of the necessity of submitting authentication requests and identity token requests with `redirect_uri` value. TARA already has the hardcoded `redirect_uri` value in its database from when the service provider registered the client application to be a user of their service. If a request is made with a certain `client_id`, TARA can reference that `client_id` and retrieve its `redirect_uri` from its database. There is also no need for the user to be redirected anywhere else besides the client applications `redirect_uri`. Thus, to possibly optimise the authentication process, the `redirect_uri` variable could be removed from the authentication request and identity token request.

3.4 Identity token verification

The client application does the identity token verification once it has received the token from TARA as the result of the identity token request. The technical specification of TARA (Section 5.1 in [4]) mandates several checks that have to be done:

- Verifying the signature of the token;
- Verifying the issuer of the token;
- Verifying the addressee of the token;
- Verifying the validity of the token;
- Verifying the authentication method used in the authentication;
- Verifying the eIDAS level of assurance.

These checks (and possible extra checks not mentioned in this list) will be analysed in more detail in the subsections below.

3.4.1 Verifying the signature of the token.

The signature is the third part of the identity token and needs to be verified to check the token's integrity. To verify the token's signature, the client application needs to generate a new signature, which is done using the public key of TARA and check if it matches the original signature included in the token. The new signature is created by taking the Base64 encoded header and payload of the token and signing with the RS256 signing algorithm using the TARA public key, which is then Base64 encoded. If this check were missing or implemented incorrectly, an attacker would be able to submit an identity token and impersonate any user they want.

3.4.2 Verifying the issuer of the token

The issuer of the certificate is held in the '`iss`' value of the token. TARA uses only two values for this parameter: '`https://tara.ria.ee`' and '`https://tara-test.ria.ee`'. The client application must make sure the '`iss`' value is one of the values mentioned above. This check is done more to add a layer of security than anything else and does not seem to have any specific attack if this check is missing. There could be a problem if there were several issuers. An identity token issued by a less trusted issuer could be submitted to a service provider as if it was a more trusted issuer.

3.4.3 Verifying the addressee of the token

The addressee of the certificate is held in the '`aud`' value of the token. This check verifies if the identity token is used in the service, it was issued for by TARA. The client application must check that the '`aud`' value of the token matches the `client_id` of the client application. If the check were missing, a valid token could be used in other services besides the service the token was issued for.

3.4.4 Verifying the validity of the token

The validity of the certificate is verified using the values held in the '`iat`', '`nbf`', and '`exp`' values of the token. The '`iat`' value is the time of issue of the certificate, '`nbf`' is the validity start time of the certificate, and '`exp`' is the certificate's expiration time; all these values are in Unix epoch format. The client application needs to use its clock to verify

that the issued identity token is valid and not expired by checking that the current time is larger than the 'nbf' value and smaller than the 'exp' value. TARA deems that the identity token must be used within 5 minutes.

If this check were not made, then an expired, but most importantly, valid identity token could be used to log in to a service.

3.4.5 Verifying the authentication method used in the authentication

The authentication method used in the authentication is held in the 'amr' value of the token. This check verifies whether the service provider authorises the authentication method used in the authentication process. The client application has to check if the 'amr' value of the token is amongst the allowed values.

If this check were not made, a user could authenticate themselves with an authentication method not allowed by the service provider, which could be done through the value manipulation of the authentication requests' scope' parameter.

3.4.6 Verifying the eIDAS level of assurance

In the event a user authenticates themselves via EU eID, then the identity token would contain the 'acr' value, which is the level of authentication based on the eIDAS level of assurance. There are three levels: 'low', 'substantial' and 'high'. The client application has to check whether the 'acr' value is of the same level or higher than the client application has set in the 'acr_values' parameter of the authentication request. If the 'acr_values' parameter is not specified in the authentication request, the level of assurance of the identity token has to be 'substantial' or 'high'.

If this check were not made, then a user authenticating with cross-border authentication services of a lower level of security can gain access to a higher level of assurance services.

3.4.7 Verifying the state value

The state value is held in the 'state' value of the token. TARA deems this check not necessary. The necessity of this check stems from the fact that when random 'state' value is generated by eesti.ee, saved in a cookie, and the user is redirected, with the authentication request and the 'state' value, to TARA. Later eesti.ee receives from the user a redirect request that contains a valid 'code' and the same 'state' value as in the authentication request. Since the 'state' value in the redirect request matches the 'state'

value in the user's session cookie, `eesti.ee` obtains an identity token using the 'code' and logs the user into the service. The problem is that this identity token was issued in some other authentication process to `eesti.ee` performed simultaneously. To prevent this, a check should be made that verifies that the 'state' value saved into the cookie is the same as the one in the identity token.

3.5 Man-in-the-middle attacks

A man-in-the-middle attack is where the user and the server think they are communicating over a secure and private connection, when really the interaction is being observed by a third party, usually the attacker. The attacker needs to create mutual authentication between both parties in order for the attack to succeed. This type of attack allows the attacker to listen into the conversation between the user and service and potentially steal sensitive user credentials or information that can be used to impersonate said user. [7]

Suppose an attacker were to listen in on the communication between the user's browser and the TARA server. In that case, an attacker could modify the HTTP requests sent between the user and server. Defence against these types of attacks is done using strong encryption and authentication between the user and the server. All of TARA's endpoints for communication use HTTPS, which use TLS to encrypt the communication protocol. The attacker has to obtain the private keys used to encrypt the communication to decrypt their communication.

3.6 Cross-site request forgery and the 'state' variable

Cross-site request forgery is an attack that forces a user's browser to initiate a specific GET or POST request to a third party in which they are authenticated. These requests are planted inside websites created by the attacker and sent through social engineering methods [8]. Cross-site request forgery targets websites that do not supplement sensitive HTTP requests with cross-site request forgery countermeasures.

Suppose a scenario where an attacker would create a webpage that contains an HTML image tag `<img src="https://...">` and, during the authentication process, gets the user's browser redirected to the attacker's webpage. After this page finishes rendering in the user's browser, the browser will try to load the image specified in the '`src`' attribute, which effectively will send a GET request to the referenced site with any specified parameters.

To prevent this attack, TARA deems it necessary to add the state value to the authentication requests sent by the client application. Once received, TARA sends back the state value in the redirect request, and it is up to the client application to verify whether the sent state value is the same as the received. Per OAuth 2.0 specifications, the application must keep it in a location that is only accessible to the user and client application (see Section 3.6 [6]). In TARA, the state value is saved into a cookie with the '`HttpOnly`' attribute applied to it [4]. The state value is based on a nonce generated by the service provider nonce after sending an authentication request and receiving a redirect request in return. As a nonce, TARA suggests generating a string of at least 16 characters, which then is hashed with the SHA256 hashing algorithm and converted to Base64 for the state value [4].

While the TARA technical specification [4] does not state this, but when the nonce is saved into a cookie, in addition to the '`HttpOnly`' attribute, the '`Secure`' attribute should be applied as well. A cookie with the '`secure`' attribute will only be included in HTTP requests transmitted over HTTPS [9]. Sending a cookie over unencrypted channels could be stolen by attackers. Modern browsers like Google Chrome no longer allow cookies with the '`Secure`' attribute to be accessed and modified by insecure HTTP requests [10]. Therefore, the need to generate the 'state' value in the manner of hashing a nonce value with the SHA256 hashing algorithm and then Base64 encoding the results seems unnecessary. The nonce value could be saved in a cookie with the '`HttpOnly`' and the '`Secure`' attributes.

3.7 Resistance to phishing attacks

A phishing or fishing attack is a hacking technique used to create a copy of a website or a service to steal or use the victim's credentials to access sensitive information [11]. Phishing attacks are among the most common types of cyber-attacks used on the internet, so protection against them is vital.

To prevent these types of attacks, TARA utilises three key elements: the `client_id` variable, `redirect_uri` variable, and the 'state' variable. Upon sending an authentication request to the TARA server, TARA checks the `client_id` and `redirect_uri` values whether the client application is a registered user of the authentication service [4]. If the `client_id` value does not match with the `redirect_uri`, then the server sends back an error message with an incident number (see Figure 11).

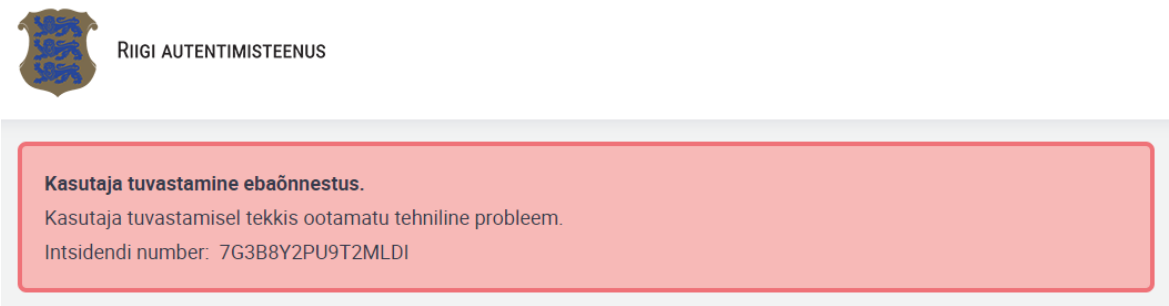

Figure 11. Example of an error on a failed authentication request with an incorrect `client_id` and/or `redirect_uri` value.

However, if both the `client_id` and `redirect_uri` are deemed correct by TARA, a redirect request is sent, and the client application will have to check the 'state' value. In analysing cross-site request forgery, it was noted that the state value is a compulsory part of any authentication request. However, a phishing attack will not be prevented, and the user is logged into a service against their will if the client application does not correctly implement a 'state' variable check.

Another way to prevent these types of attacks is to increase security on the human side of the process. Currently, when a user is authenticating themselves via TARA to a service provider, TARA does not display the client application the user is authenticating themselves, but just the fact that they are authenticating via TARA (see Figure 2). Neither are they displayed when using Smart-ID or Mobile-ID (see Figure 12, Figure 13, Figure 14 and Figure 15).

Figure 12. Smart-ID is used for logging into SEB bank

Figure 13. Smart-ID is used for logging into `eesti.ee` (uses TARA)

Figure 14. Smart-ID is used for logging into `rahvaregister.ee` (uses TARA)

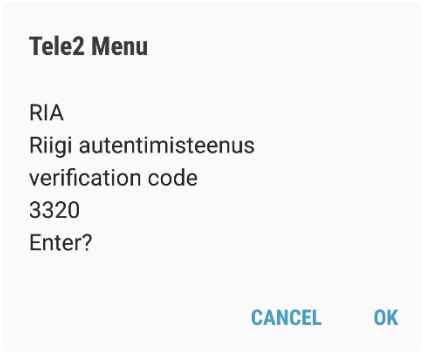

Figure 15. Using Mobile-ID to authenticate with a service that uses TARA.

If the service provider identifier were shown during authentication with TARA, users would notice that they are not authenticating to the service they want and cancel the authentication.

4 Summary of Findings

Below is a list of noteworthy security suggestions that can be drawn from the analysis:

1. RIA should display the service provider's identifier, e.g., the domain name, on the TARA authentication page. In addition, RIA should also include the service provider's identifier in Mobile-ID and Smart-ID authentication prompt. This method would help the user detect potential phishing attacks because they would then know what they would be authenticating.
2. TARA specification does not require verifying the 'state' parameter in the identity token, which could allow an attack where the service provider gets an authentication code that corresponds to an identity token issued for a different 'state' parameter in the same service. A check that verifies the 'state' value in the identity token using the value in the saved cookie should be made.
3. The need to generate the 'state' value in the manner of hashing a nonce value with the SHA256 hashing algorithm and then Base64 encoding the results seems unnecessary if the nonce value was instead saved into a cookie with the 'HttpOnly' and the 'Secure' attributes. The 'Secure' attribute does not allow the cookie to be accessed or modified by insecure HTTP requests.
4. The inclusion of the `redirect_uri` variable in the authentication request and identity token request may not be necessary. TARA could very well use the hardcoded `redirect_uri` value in its database to redirect the user's browser, and the authorisation code can be binded to just the `client_id` instead of them both.

5 Conclusions

The objectives set in the introduction of the thesis were achieved. The TARA authentication protocol was documented, and its aspects were reviewed, a mock service was used to obtain a better understanding of the system, and a security analysis of the TARA authentication protocol was performed to understand its security aspects and see what could go wrong.

Since the authentication service TARA is based on the OpenID Connect protocol, which is based on the OAuth 2.0 protocol, the designers of TARA did not need to reinvent the wheel. The OAuth 2.0 authorisation framework is documented in exquisite detail and is very battle-tested. TARA mostly just needed adjustments to suit the needs of Estonia's public sector's e-services.

While the author of the thesis did not find any glaring security flaws, the author pointed out some points of interest that could either optimise the authentication process or add a few extra security layers to the protocol. The authentication service TARA is based on an already robust system, and one can assume those strong traits and features would carry over well.

References

[1] Estonia Information System Authority, “Estonia Information System Authority’s Authentication Services,” 9 November 2020. [Online]. Available: https://www.ria.ee/en/state-information-system/eid/partners.html. [Accessed 7 December 2020].

[2] N. Nakimura, J. Bradley, M. B. Jones, B. d. Medeiros and C. Mortimore, “OpenID Connect Core 1.0 Technical Specification,” 8 November 2014. [Online]. Available: https://openid.net/specs/openid-connect-core-1_0.html#CodeFlowAuth. [Accessed 8 December 2020].

[3] D. Hardt, “OAuth 2.0 Authorization Framework,” October 2012. [Online]. Available: https://tools.ietf.org/html/rfc6749. [Accessed 8 December 2020].

[4] Estonia Information System Authority, “TARA Service Technical Specification,” 2 April 2019. [Online]. Available: https://e-gov.github.io/TARA-Doku/TechnicalSpecification. [Accessed 7 December 2020].

[5] Auth0, “Introduction to JSON Web Tokens,” [Online]. Available: https://jwt.io/introduction. [Accessed 6 April 2021].

[6] T. Lodderstedt, M. McGloin and P. Hunt, “OAuth 2.0 Threat Model and Security Considerations,” January 2013. [Online]. Available: https://tools.ietf.org/html/rfc6819. [Accessed 5 April 2021].

[7] Auth0, “Bucket brigade attacks,” [Online]. Available: https://auth0.com/docs/security/prevent-common-cybersecurity-threats#bucket-brigade-attacks. [Accessed 12 April 2021].

[8] KirstenS, “Cross Site Request Forgery,” 25 January 2021. [Online]. Available: https://owasp.org/www-community/attacks/csrf. [Accessed 27 March 2021].

[9] A. Barth, “HTTP State Management Mechanism,” April 2011. [Online]. Available: https://tools.ietf.org/html/rfc6265#section-4.1.2.5. [Accessed 7 May 2021].

[10] M. West, “Feature: Strict Secure Cookies,” 9 November 2020. [Online]. Available: https://www.chromestatus.com/feature/4506322921848832#details. [Accessed 7 May 2021].

[11] L. Muthiyah, “Phishing,” 10 August 2018. [Online]. Available: https://thezerohack.com/phishing. [Accessed 1 April 2021].

Appendix

I. The mock applications HTML and PHP code

```
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>TARA Testing Service</title>
    <style>
        div#auth {
            line-height: 40px;
            padding: 20px;
            width: 98%;
            word-wrap: break-word;
        }
    </style>
</head>
<body>
<?php
//Endpoint for authentication request
$auth_endpoint = "https://tara-test.ria.ee/oidc/authorize";
//The redirect URL
$redirect_uri = "http://kodu.ut.ee/~janerikr/TestTARA/";
//The authentication scope, "openid" is compulsory
$scope = "openid";
//Security code against false request attacks.
$state = base64_encode(hash("sha256", "utthesisresearch"));
//Determines the manner of communication of the authentication result to the server. The
default value is "code"
$response_type = "code";
//Chosen client ID
$client_id = "ut_bsc_testing_service";
//Secret code issued by RIA
$client_secret = "Nu7kWypWzFOa87qXtXms2Dy_";
//Endpoint for identity token request
$identity_endpoint = "https://tara-test.ria.ee/oidc/token";

//HTTP GET authentication request
$authReq =
    $auth_endpoint . "?" .
    "redirect_uri=" . $redirect_uri . "&" .
    "scope=" . $scope . "&" .
    "state=" . $state . "&" .
    "response_type=" . $response_type . "&" .
    "client_id=" . $client_id;

//Received response code
$response_code = $_GET['code'];

$error = $_GET['error'];
$error_desc = $_GET['error_description'];

$curl cmd = 'curl -H "Authorization: Basic ' . base64 encode($client id . ":" . $client se-
cret) .
    '" -d "grant_type=authorization_code&code=' . $response_code . '&redirect_uri=' . $re-
direct_uri .
    '" -X POST ' . $identity_endpoint;
?>
<div id="auth">
    <a href="index.php">Refresh</a>
    <hr>
    <p>
        <?php if ($response_code == null) {
            echo "The authentication request being sent out:";
            echo "<br>";
            echo $authReq;
        } ?>
    </p>
    <?php if ($response_code == null) echo "<hr>"; ?>
    <a href=<?php echo $authReq ?>>
        <?php if ($response_code == null) echo "Send an authentication request" ?>
    </a>
    <p>
        <?php if ($response_code != null) echo "Response code: " . $response_code; ?>
```

```
    </p>
    <p>
        <?php if ($response_code != null) echo "cURL Command: " . $curl_cmd; ?>
    </p>
    <?php if ($response_code != null) echo "<hr>"; ?>
    <a href="https://jwt.io/" target="_blank">
        <?php if ($response_code != null) echo "Link to decode access token"; ?>
    </a>
    <?php if ($error != null) {
        echo "<hr>";
        echo "Error: " . $error;
        echo "<br>";
        echo "Error description: " . $error_desc;
    } ?>
</div>
</body>
</html>
```

II.License

Non-exclusive license to reproduce thesis and make thesis public

I,

Jan Erik Kriisk,

(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive license) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Security analysis of RIA's authentication service TARA,

(title of thesis)

supervised by Arnis Paršovs, PhD.

(supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons license CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Jan Erik Kriisk

07/05/2021