# INTEGRATING KEYCLOAK SSO WITH ATYPICAL CLIENTS

Darius Mihai[1], Vlad Năstase[2], Elena Mihăilescu[3],
Mihai Carabaș[4], Sergiu Weisz[5], Nicolae Țăpuș[6]

*Authentication protocols like OpenID Connect and SAML are pervasive throughout the internet, allowing users to validate their identity to services without presenting their credentials on every visited website. Configuring authentication for a service is usually just a matter of configuring a 1-1 relationship between a client in the single sign-on service and the application that requires authentication, and exchanging a set of standard attributes is enough. However, there are some cases when more specific requirements must be addressed. This paper presents a few such cases, and some pointers for how they can be handled.*

**Keywords:** Keycloak, single sign-on, integration, Microsoft AAD, eduGAIN

## 1. Introduction

Single sign-on [1][2][3] (abbreviated as SSO) services and centralised authentication protocols are commonly used by organisations that desire to bolster their security and unify the authentication process across their services. By delegating user authentication to a dedicated service, developers are able to reduce the implementation complexity of applications (e.g., they no longer have to store and secure user passwords), improve the user experience (i.e., users can authenticate once on the SSO service, and subsequent logins reuse the same session), as well as provide access to users from outside their organisations (i.e., through federation protocols).

[1]PhD student, Computer Science Department, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: darius.mihai@upb.ro

[2]PhD student, Computer Science Department, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: vlad_iulius.nastase@upb.ro

[3]PhD, Computer Science Department, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: maria.mihailescu@upb.ro

[4]Professor, Computer Science Department, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: mihai.carabas@cs.pub.ro

[5]PhD, Computer Science Department, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: sergiu.weisz@upb.ro

[6]Professor, Computer Science Department, National University of Science and Technology POLITEHNICA Bucharest, Romania, e-mail: nicolae.tapus@upb.ro

The usual approach is to create a client service configuration on the single sign-on service for every web application that requires authentication and the client receives a standard set of user attributes (e.g., username, email address, first name, last name) from the SSO. The most commonly used protocols for single sign-on, OpenID Connect and SAML, use browser redirects in order to direct the user between the service domain and the single sign-on service, as well as for information exchange.

These observations are not always true, however [4][6]. There are specific cases where clients may require the released information to have a non-standard format, the list of clients may not be as strictly defined (i.e., a large number of client applications may need to be periodically updated, added or removed) or the authentication process needs to be extended to non-web based applications.

In this paper we will discuss about how we have managed to configure a single sign-on services for a number of irregular clients. Section 2 presents what single sign-on services and authentication protocols are and which implementation options we have evaluated, Section 3 presents how we have configured a number of clients with atypical requirements, Section 4 shows the results we have achieved, and Section 5 draws some short conclusions.

## 2. Single Sign-On

User information and access to company or institution resources have always been primary attack targets for attackers [7][8]. Blocking access to services (also known as denial of service attacks), stealing user information, keeping data hostage for ransomware via encryption, or more recently, using company resources occupied by mining cryptographic currencies, are usually common attack vectors for financially or politically motivated adversaries.

A common approach to enforcing stricter authentication policies and the use of stronger authentication mechanisms, while also reducing the inconvenience they impose on users, is to use single sing-on services. Single sign-on services handle the authentication process centrally, and provide user authentication and authorisation for other services through standardised protocols. For users this mitigates the stress of having to supply their credentials to each individual service, while also reducing the implementation complexity of said services.

### 2.1. Authentication protocols

User authentication protocols have been designed to exchange user information between services regardless of who designs the services. The most common protocols [9][10] used for web applications are **OpenID Connect** and **SAML**.

**Security Assertion Markup Language (SAML)** is a data exchange standard used for SSO that relies on assertions and documents in XML format to transfer data between the single sign-on service, called the *identity*

*provider* (IdP), and services provided to users, called *service providers* (SP). To establish the communication parameters, the identity provider publishes a set of endpoints the service provider can use to request information, as well as a set of keys used for assertion / document signature and encryption. To establish trust parameters with the identity provider, the service provider defines its own communication endpoints, and optionally, a certificate used for assertion / document signature and encryption. For improved security it is recommended that all messages between the parties are signed, if possible. SAML is provided as an authentication mechanism by many applications; in our case, eduGAIN, an international authentication inter-federation system for research and education institutions, and Microsoft services were of particular interest.

**OpenID Connect (OIDC)** is an extension of the OAuth 2.0 authorisation protocol[1]. Data transfers between the parties has a much simpler structure compared to SAML. In this case, the single sign-on service is called *OpenID provider*, and the application service is called the *relaying party*, but for simplicity we'll keep the same names used in SAML: *identity provider* and *service provider*. Instead of signed XML assertions and documents, OIDC usually relies on simpler authentication mechanisms for the service provider, using a client ID and client secret (i.e., which are equivalent to the username and password in regular authentications), and responses are formatted as JSON web tokens (JWTs), which are by default signed. OpenID Connect is very commonly used by public services to provide user authentication using public identity providers (e.g., Google, Microsoft, Facebook, Apple, GitHub). Using this mechanism, a service can get a trustworthy validation of the user's identity without having to manage the user's authentication credentials directly.

### 2.2. SSO Solutions

A first implementation option for a single sign-on service with strong authentication combined **privacyIDEA**[1] as a plugin for **simpleSAMLphp**[2]. Both solutions are open-source and in conjunction they could provide strong authentication to clients. In this setup privacyIDEA was a strong authentication provider that handled multi-factor user authentication for simpleSAMLphp, while simpleSAMLphp was a SAML identity provider for other services. While this setup worked, there were a few issues that made us look into other options. Firstly, the loose integration made the requirement of configuring the second authentication factor less obvious for users (i.e., they were not automatically redirected to the privacyIDEA page if no second factor was configured, but authentication could not go through using just the user's password). The second issue was the lack of OpenID Connect support, which would be required for some of the client services.

[1]https://www.privacyidea.org/, [Online. Accessed: 26 July 2024]
[2]https://simplesamlphp.org/, [Online. Accessed: 26 July 2024]

Because of the limitations mentioned above, we started looking into more complete solutions. While there were other options available, two options stood out, namely **Gluu**[3] and **Keycloak**[4], which were considered for evaluation. Both services were providing fully functional implementations for both SAML and OpenID Connect, strong authentication, as well as reasonable documentation with community support.

The first solution that we had considered of the two was Gluu, especially for its native authenticator application, called *SuperGluu*. Gluu offered an extreme level of configuration of all system interactions, including user authentication (i.e., authentication steps were provided as scripts that could be edited by administrators) and how the authentication process would behave (i.e., what kind of tokens would be sent to the client for every step of the process). While this level of fine-tuning would likely be acceptable, or even desirable, for administrators who are extremely familiar with the authentication protocols, it felt overwhelming and error-prone for anyone less experienced, and thus likely to create configurations that violate the respective protocols or introduce security vulnerabilities. From what we could tell, there was no obvious way to create a simple client that uses a specific authentication flow.

In the search for a simpler, but still fully-fledged single sign-on application we next evaluated Keycloak[4][5][6], and while it did not provide the same level of extreme configuration in the administration console itself, it had much more sensible defaults for client configurations. In our testing we managed to get a working deployment of Keycloak much easier because of its much simpler administration interface, that provides client configurations that comply with the relevant standards by default (i.e., it actually does not provide the options to set up invalid token exchange flows).

Table 1 shows a few factors we have considered when deciding which option would be best fit for our needs. Note that we have done this analysis around 2021, and some data may have changed since then (i.e., after we have moved to a different solution, we did not evaluate some features any further).

For the community size comparisons we used the GitHub contributors pages for simpleSAMLphp[5], Gluu[6], and respectively Keycloak[7].

Because of its ease of use and solid implementation, we ultimately decided to use Keycloak as the single sign-on service in our university.

[3]https://gluu.org/, [Online. Accessed: 20 September 2024]

[4]https://www.keycloak.org/, [Online. Accessed: 20 September 2024]

[5]https://github.com/simplesamlphp/simplesamlphp/graphs/contributors, [Online. Accessed: 23 September 2024]

[6]https://github.com/GluuFederation/oxAuth/graphs/contributors, [Online. Accessed: 23 September 2024]

[7]https://github.com/keycloak/keycloak/graphs/contributors, [Online. Accessed: 23 September 2024]

*Table 1*

**Comparison between simplaSAMLphp/privacyIDEA, Gluu and Keycloak**

| **Feature** | **simpleSAML/ privacyIDEA** | **Gluu** | **Keycloak** |
|---|---|---|---|
| Authentication protocols | SAML | SAML, OIDC | SAML, OIDC |
| Mobile application | no | yes | no |
| Ease of use (users) | hard | easy | easy |
| Administration complexity | moderate | very high | moderate |
| Configuration store | local files | LDAP | relational database |
| Supports high-availability | not tested | yes | yes |
| Community size | moderate | small | large |
| Available documentation and community support | reasonable | good | very good |

## 3. Configuring Clients

Adding single sign-on support for most web applications is usually simple, as the authentication protocols described in Section 2.**1** have been designed for ease of use in this scenario and there are various connector libraries available for the programming languages commonly used for this use-case (e.g., PHP, Python, JavaScript, Java). Configuring the connection parameters in Keycloak is also not complicated when setting up clients for most service providers.

Most web applications that we deploy in our university (e.g., Moodle, OpenStack, OpenSearch, GitLab) natively support the OpenID Connect or SAML standards, so integrating them with Keycloak was painless. The university also hosts various custom applications, some of which are written in PHP and use simpleSAMLphp as an authentication library, providing SAML-based centralised authentication.

However, there are some cases when the clients require more complex configurations. We will present a few such cases in the following sections.

### 3.1. Microsoft services

Most client applications simply require a set of attributes, as exported by Keycloak. There are, however, some clients that have special requirements that must be met. As mentioned before, the integration with Microsoft educational services was of particular interest for us. Enforcing strong authentication with Microsoft is possible by default, through their own single sign-on service, but in our case, we considered that requiring users to configure multiple authenticator applications on their devices could prove confusing. As such, we have opted to configure the Microsoft authentication service to delegate authentication to Keycloak for the university's tenant using the SAML protocol.

This integration, however, was not trivial because of how our services interact. The Microsoft Entra ID single sign-on heavily relies on the Microsoft Active Directory (AD) service architecture. One of the mechanisms that the AD service uses to ensure that objects (e.g., users and groups) can be moved between various organisational units inside the AD forests and domains is called the `sourceAnchor` or the `immutableId`[8] of the object. This attribute is used to uniquely identify an object inside Active Directory and must be sent in the SAML response as the `NameID`[9] attribute of the response. The issue, in our case, was that since we connected Keycloak to LDAP, instead of the Active Directory of the tenant, we could not access the `immutableId` attribute directly.

Despite not having direct access to the attribute itself, we could take advantage of the following in order to manually derive it:

- the `sourceAnchor` may be computed based on a different attribute; in our case the attribute is the `objectGUID`;
- the `sourceAnchor` is computed as `base64(objectGUID)`;
- the `objectGUID` attribute is synchronised between LDAP and AD as the `ntUniqueId` LDAP attribute.

Using this information we can also compute the same value of the anchor for the authentication process. Unfortunately, Keycloak only allows computing regular SAML attributes using JavaScript scripts, or extracting user attributes without any transformations for the `NameID` attribute, but not deriving the `NameID` programmatically. Consequently, we took advantage of the modular design of Keycloak and wrote a plugin that is derived from the code that handles regular SAML attributes and used it to export a computed `NameID`.

This requirement is somewhat specific to our deployment, since we keep the user information in LDAP, and the Active Directory connection is only used to synchronise user information with the Microsoft services in read-only mode. Members of the community have reported that they managed to connect[10] the Microsoft authentication with Keycloak using the regular *User Attribute Mapper for NameID* mapper that matches a static user attribute with the NameID, which is only possible if Keycloak is connected to the Active Directory directly.

The JavaScript code that handles the transformations can be seen in Listing 1. It first looks for a user attribute called `ntUniqueId` (which is extracted from LDAP), then converts the value from a hexadecimal encoded string (i.e., a hexadecimal dump of the string) to a byte array (i.e., the raw bytes of the representation of the original string) and finally transforms the array into a

[8] https://learn.microsoft.com/en-us/entra/identity/hybrid/connect/plan-connect-design-concepts#sourceanchor, [Online. Accessed: 24 July 2024]

[9] https://learn.microsoft.com/en-us/entra/identity/hybrid/connect/how-to-connect-fed-saml-idp#required-attributes, [Online. Accessed: 24 July 2024]

[10] https://github.com/keycloak/keycloak/discussions/12668#discussioncomment-9875689, [Online: Accessed 23 September 2024]

Listing 1. Compute immutableId from objectGUID

```
// Get encoder to convert JavaScript string to a
// base64-encoded Java-compatibe string.
var b64=Java.type("java.util.Base64").getEncoder();

// based on https://stackoverflow.com/a/34356351
// Convert a hex string to a byte array
var hexToBytes = function (hex) {
    var bytes = [];
    // Parse groups of two characters in the hex
    // string and convert them to integer values.
    // Save the results of the conversion in a byte array.
    //
    // e.g., "0c0a" is converted to an array containing the
    // values [12, 10]
    for (var c = 0; c < hex.length; c += 2)
        bytes.push(parseInt(hex.substr(c, 2), 16));
    return bytes;
};

// Get the users's "ntUniqueId" attribute from Keycloak.
var uuid = user.getFirstAttribute("ntUniqueId");
// Convert the "ntUniqueId" string to the equivalent array
// of bytes.
var uuidBytes = hexToBytes(uuid);
// Compute the base64 encoding of the array of bytes.
var immutableId = b64.encodeToString(uuidBytes);

// Return the base64-encoded value as the mapper's result.
exports = immutableId;
```

base64-encoded string. This is the same process that AD uses to compute the `immutableId` value, and thus we get the same value that can be sent as a response to an authentication request from the Microsoft SSO.

### 3.2. eduGAIN

eduGAIN[11] is an authentication inter-federation system for research and education organisations, that uses SAML and is managed by GEANT[12]. The various identity providers and service providers that are part of the system are managed by local federations (i.e., organisations from each participating country, like RoEduNet for Romania).

[11]https://edugain.org/, [Online. Accessed: 20 September 2024]
[12]https://geant.org/, [Online. Accessed: 20 September 2024]

According to their statistics[13], the system is currently composed of over 5700 identity providers and 3700 service providers. While it is not strictly necessary to authorize communications between all service providers and all identity providers, it is reasonable to expect a very large number of service providers that can connect to our identity provider.

The configurations of the entities in eduGAIN are provided as a large XML file (i.e., currently it is larger than 80 MB), that is processed by the participating federations and then made available to local entities. The list of entities in eduGAIN is dynamic, which means that an identity provider must be able to import and update the federation information periodically, and also be able to handle a very large number of client applications.

Since Keycloak did not natively support keeping a dynamic list of clients and updating their information (including removing them), we have opted to leverage simpleSAMLphp's *metarefresh* module, which can automatically download a file containing entity metadata and updating the entities registered in the service. In this case simpleSAMLphp acts as a SAML bridge, which is seen as a client from Keycloak's perspective, and as an identity provider from the perspective of the service providers in eduGAIN. When configured this way, the bridge does not actually authenticate the user; instead, user authentication and user attribute release is performed by Keycloak, and simpleSAMLphp forwards the responses to the service providers with minimal changes.

### 3.3. SSH login through SSO

Extending authentication to CLI tools like SSH is a more exotic use-case, since single sign-on protocols have mostly been designed for the web. Since SSH clients cannot display a web interface to allow users to interact with the single sign-on's login page, we cannot use the code authentication flow. Instead, we have to use one of the following authentication flows:

- resource owner password credentials flow;
- device authorisation flow.

The resource owner password credentials flow is a non-recommended authentication flow that places a lot of trust on the client service. Most authentication flows redirect the user to the login page of the single sign-on service, so only the SSO has access the user's credentials during the entire authentication process. The resource owner password credentials flow, however, works more similarly to local authentication, where the user inputs their username, password and second authentication factor into a login form on the service's login page, and then the service sends the credentials to the SSO in the user's stead. This approach is both more risky, since the user credentials can be leaked if the application (i.e., SSH) server is compromised, and provides less functionality since some authentication tokens like security keys only work with a browser.

[13]https://technical.edugain.org/entities, [Online. Accessed: 20 September 2024]

The device authorisation flow is a much better alternative in our case. This flow has been specifically created for devices without browser support or limited input (e.g., a TV) to achieve secure user authentication. The flow is actually composed of two flows: a flow between the user and the service (in our case, between SSH client and server), and a flow between the user and the SSO server through the browser. When users try to authenticate to the application service (i.e., SSH server), the service sends a special request to the SSO service and receives an authorisation code that the user must input in order to authorise the device. The user can then go to the SSO service's device authorisation page using their browser, enter the authorisation code they received from the SSH server (they will be required to authenticate if they are not) and the authorisation process is completed. After the process is completed, the service can connect to the SSO's token endpoint to get information about the user. This approach is much more secure, as it does not provide any user credentials to the SSH server, and can also leverage the familiar browser authentication flows for users.

For our implementation we have drew inspiration from the Okta Developer blog[14]. We also implemented this functionality as a PAM module library, but instead of C, we wrote the implementation in the D programming language in order to leverage its almost seamless inter-compatibility with C libraries (required for the PAM module library), and memory safety guarantees.

## 4. **Results**

GEANT provides a self-assertion test for attributes released to eduGAIN by identity providers. As seen in Figure 1, our simpleSAMLphp bridge correctly provides the expected attributes, and can consequently be used for authenticating to various connected services. We were also able to confirm that the released information is correct using services provided by the European Union that use eduGAIN to authenticate users, like the ERASMUS student ID card programme.

For the Microsoft integration the only available test was confirming that the authentication process correctly redirects users from Microsoft's single sign-on to our on-premise Keycloak deployment for authentication, and the response is accepted afterwards. As a note, users are not redirected to Keycloak for authentication often, as Microsoft's Entra ID single sign-on service allows for long-lasting user sessions, so re-authentication is not required in most cases. After we have confirmed that everything works as expected, there have not been any major changes that were required.

The SSH authentication integration with the single sign-on service using OpenID Connect is used on our front-end processor (*fep*) system. Users have the option to authenticate using SSH keys, or using interactive authentication

[14]https://developer.okta.com/blog/2021/08/20/cli-ssh-oauth-device-grant, [Online. Accessed 20 September 2024]

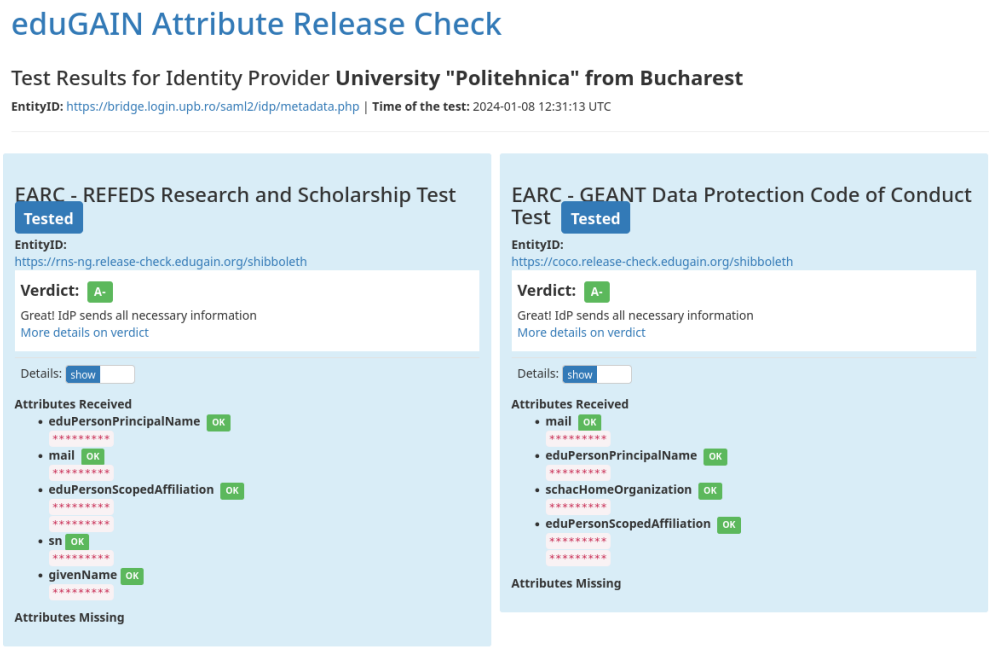


Fig. 1. eduGAIN attribute self-check

that requires them to authorise the connection using the SSO service. In the latter case, users receive a message asking them to log in using Keycloak and authorise the connection using an authorization code, as seen in Listing 2.

Listing 2. FEP interactive login prompt

```
Please login at https://login.upb.ro/auth/realms/UPB/device.
Then input code ABCD-EFGH

Press Enter to continue:
```

There have been some initial issues with some users because we did not use the correct base64 decode functions for the responses from the SSO service. The responses are sent using base64 URI-safe encoding without padding, which means that the "=" character that is normally used as padding for base64 encoded strings when the length is not a multiple of *4* was missing; this was not an issue for most users, but there were some users who reported that they could not log in, so we investigated the cause and fixed the issue within a few days.

As an additional caution, we have also modified the device authorisation flow to require an additional confirmation using the user's one-time password. We have done this as a preemptive measure to make some attack vectors harder. Because the device authorisation process in the browser is not a result of a redirect from a website the user has visited, if a malicious actor is able to convince the user to click on a link that inputs the authorisation code automatically (e.g., using a URL like `[...]/device?code=ABCD-EFGH`, although

this approach does not work in the current version), the SSO service could authorise access for the attacker without properly asking the user for confirmation. In this scenario, adding the requirement to confirm the user's identity using a one-time password should provide an opportunity for the user to figure out that the process is suspicious and interrupt it if it was not initiated by them.

## 5. Conclusions

In this paper we have presented an analysis of existing single sign-on solutions and the reasons why we have ultimately decided to use Keycloak. While it did not provide a turnkey solution for every use-case we had planned, we were able to leverage its modular architecture and extensibility to add the required functionality. By employing Keycloak, we have managed to achieve a good balance between better security for services, even beyond just web services, and convenience for our users.

Through the use of the simpleSAMLphp bridge and eduGAIN we have managed to extend the authentication service beyond our local services and provide an opportunity for our users to access many web applications and benefits even beyond the borders of our country. Additionally, we were able to use custom plugins to add the JavaScript mapper functionality for the *NameID* attribute that was required to make Keycloak the authoritative authentication server for Microsoft services in our tenant. Furthermore, through non-standard authentication flows in the OpenID Connect standard we have been able to add authentication support for applications that are not based on web services.

We have so far used these client configurations for over two years, throughout the university year (i.e., with increased loads near the exam sessions, or lower loads during the summer break) and besides the issues mentioned in this paper there have not been any performance or user satisfaction degradations after the clients have been configured fully.

## Acknowledgment

This work is funded under the SOC digital artifact analysis and threat intelligence sharing maturity buildout and operation in East Europe (Romania, Lithuania, and beyond) (SOCcare) Project, with the support of the European Commission and Digital Europe Programme (DIGITAL), under Grant Agreement No. 101145843. This project is funded by the European Union. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the European Cybersecurity Competence Centre can be held responsible for them.

# REFERENCES


[1] *Y. Sadqi and Y. Belfaik and S. Safi*, Web oauth-based SSO systems security, Proceedings of the 3rd International Conference on Networking, Information Systems & Security, 2020, 1–7

[2] *I. Nongbri and P. Hadem and S. Chettri*, A survey on single sign-on., Int. J. Creative Res. Thoughts , 2018, vol 6, nr 2, 595–602

[3] *F. Alaca and P. C. V. Oorschot*, Comparative Analysis and Framework Evaluating Web Single Sign-on Systems., ACM Comput. Surv. , Association for Computing Machinery (2020), 25–29

[4] *J. Anderson and K. Keahey*, Migrating towards single sign-on and federated identity, Practice and Experience in Advanced Research Computing, 2022, 1–8

[5] *M. A Christie and A. Bhandar and S. Nakandala and S. Marru and E. Abeysinghe and S. Pamidighantam and M. E Pierce*, Using keycloak for gateway authentication and authorization, 2017

[6] *A. Chatterjee and A. Prinz*, Applying spring security framework with KeyCloak-based OAuth2 to protect microservice architecture APIs: a case study, Sensors, MDPI, 2022, vol 22, nr 5, page 1703

[7] *Ö Aslan and S. S. Aktuğ and M. Ozkan-Okay and A. A. Yilmaz and E. Akin*, A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions, Electronics, MDPI, 2023, vol 12, nr 6, pages 1333

[8] *H. Hindy and D. Brosset and E. Bayne and A.K. Seeam and C. Tachtatzis and R. Atkinson and X. Bellekens*, A taxonomy of network threats and the effect of current datasets on intrusion detection systems, IEEE Access, IEEE, 2020, vol 8, 104650–104675

[9] *J.R. Almeida and A. Zúquete and A. Pazos and J.L. Oliveira*, A federated authentication schema among multiple identity providers, Heliyon, Elsevier, 2024, vol. 10, no. 7

[10] *J. Basney and P. Cao and T. Fleury*, Investigating root causes of authentication failures using a SAML and OIDC observatory, 2020 IEEE 6th International Conference on Dependability in Sensor, Cloud and Big Data Systems and Application (DependSys), IEEE, 2020, pages 119–126