

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Daniel Golubkov 232655IVCM

**Vulnerability Assessment and Penetration
Testing of Authentication and Authorization
Protocols**

Master's thesis

Supervisor: Edmund Laugasson
MSc

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Daniel Golubkov 232655IVCM

**Autentimis- ja autoriseerimisprotokollide
haavatavuse hindamine ja läbitungimise
testimine**

Magistritöö

Juhendaja: Edmund Laugasson
MSc

Tallinn 2025

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Daniel Golubkov

18.05.2025

Abstract

Authentication and authorization are the core user identification and access control processes of modern systems. With the constant rise of cyberthreats of various complexities, it becomes necessary to evaluate which of the relevant attacks are successful against authentication and authorization protocol implementations. It is additionally required to identify which techniques can be implemented to prevent said attacks. The technical contribution of this master's thesis is the practical evaluation of attacks and prevention techniques identified in the relevant research literature – such work can help service providers stay up to date with the most severe threats as well as the relevant prevention techniques. The conducted work emphasised the usage of a semi-automated evaluation process in a testing environment. More than half of the identified attacks were successful, and single implementations of prevention techniques were found to be lacking at times. Most of the successful attacks targeted the overall implementation and not the protocol. Hardening web services as well as staying up to date with recent security threats remain as essential tasks in ensuring service provider security.

This thesis is written in English and is 69 pages long, including 5 chapters, 33 figures and 3 tables.

Annotatsioon

Autentimine ja autoriseerimine on kaasaegsete süsteemide põhilised kasutajate tuvastamise ja juurdepääsu kontrollimise protsessid. Erineva keerukusega küberohtude pideva kasvuga seoses on vaja hinnata, millised asjakohastest rünnakutest on autentimis- ja autoriseerimisprotokollide rakendamiste vastu edukad. Lisaks on vaja kindlaks teha, milliseid tehnikaid saab nimetatud rünnakute ärahoidmiseks rakendada. Käesoleva magistritöö tehniline panus on vastavas teaduskirjanduses tuvastatud rünnete ja ennetustehnikate praktilisele hindamisele. Selline töö võib aidata teenusepakkujatel olla kursis kõige tõsisemate ohtude ja asjakohaste ennetustehnikatega. Läbiviidud töös on pigem rõhutatud poolautomaatse hindamisprotsessi kasutamist testimiskeskkonnas. Enam kui pool tuvastatud rünnakutest olid edukad ja kohati oli leitud, et üksikud ennetusmeetodid ei ole piisavad. Enamik edukaid rünnakuid oli suunatud üldisele rakendamisele ja mitte protokollile. Veebiteenuste tugevdamine ja viimaste turvaohutudega kursis olemine on jätkuvalt olulised ülesanded teenusepakkuja turvalisuse tagamisel.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 69 leheküljel, 5 peatükki, 33 joonist, 3 tabelit.

List of abbreviations and terms

ABAC	Attribute-Based Access Control
ACM DL	Association for Computing Machinery Digital Library
ARP	Address Resolution Protocol
API	Application Programming Interface
CA	Certificate Authority
CISA	Cybersecurity and Infrastructure Security Agency
CPU	Central Processing Unit
CSP	Content Security Policy
CSRF	Cross-Site Request Forgery
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DDoS	Distributed Denial of Service
DoS	Denial of Service
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HTML	Hypertext Markup Language
ID	Identifier
IdP	Identity Provider
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPS	Intrusion Prevention System
IT	Information Technology
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JSON Web Token
KDC	Key Distribution Center
KRBtgt	Kerberos Ticket Granting Ticket Account
LDAP	Lightweight Directory Access Protocol
MFA	Multi-Factor Authentication

MIT	Massachusetts Institute of Technology
MitM	Man-in-the-Middle Attack
NTLM	New Technology Local Area Network Manager
OASIS	Organization for the Advancement of Structured Information Standards
OAuth	Open Authorization
OIDC	OpenID Connect
OS	Operating System
OTP	One Time Password
OWASP	Open Worldwide Application Security Project
PKCE	Proof of Key Code Exchange
RAM	Random Access Memory
RSA	Rivest-Shamir-Adleman
SAML	Security Assertion Markup Language
SHA	Secure Hash Algorithms
SP	Service Provider
SSL	Secure Sockets Layer
SSO	Single Sign-On
TGS	Ticket Granting Service
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VAPT	Vulnerability Assessment and Penetration Testing
VM	Virtual Machine
WAF	Web Application Firewall
XML	Extensible Markup Language
XSS	Cross Site Scripting
XSW	XML Signature Wrapping
XXE	XML External Entity Attack
ZAP	Zed Attack Proxy

Table of contents

1 Introduction	13
1.1 Motivation	13
1.2 Research Problem Statement	13
1.3 Research Goal	14
1.4 Research Scope	15
1.5 Contribution Overview	15
2 Background.....	16
2.1 Authentication	17
2.2 Authorization	18
2.3 Single Sign-On	18
2.4 Protocols	19
2.4.1 SAML 2.0	19
2.4.2 OAuth 2.0	20
2.4.3 OIDC	21
2.4.4 LDAP	21
2.4.5 Kerberos	22
2.5 Vulnerability Assessment	23
2.6 Penetration testing	23
2.7 VAPT.....	24
2.8 Summary	24
3 Contribution.....	27
3.1 Methodology.....	27
3.1.1 Task Overview.....	27
3.1.2 Evaluation Process Overview	28
3.1.3 Research Methodology	29
3.1.4 Limitations.....	30
3.2 SAML 2.0 Protocol Implementation	30
3.2.1 XSW Attack.....	31
3.2.2 XXE Attack	32

3.2.3 Replay Attack	33
3.2.4 MitM Attack	34
3.2.5 Invalid Signature Attack	36
3.2.6 DoS/DDoS Attack	37
3.2.7 XSS Attack	40
3.2.8 CSRF Attack.....	43
3.3 OIDC Protocol Implementation.....	45
3.3.1 307-Redirect Attack.....	45
3.3.2 IdP Mix-up Attack	46
3.3.3 MitM Attack	46
3.3.4 DoS/DDoS Attack	47
3.3.5 XSS Attack	48
3.3.6 CSRF Attack.....	49
3.4 LDAP Protocol Implementation	49
3.4.1 Unauthorized Code Injection Attack	50
3.4.2 Brute Force Attack	50
3.4.3 MitM Attack	52
3.4.4 DoS/DDoS Attack	52
4 Discussion.....	54
4.1 Answers to Research Questions	55
4.2 Future Work.....	60
5 Conclusion.....	61
References	62
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis	69

List of figures

Figure 1. Evaluation process diagram.	29
Figure 2. Sample local configuration of trust relationship with an Okta IdP using the passport-saml library [16].	31
Figure 3. Sample XXE injected into a SAML response message [20].	32
Figure 4. Example of IssueInstant, NotBefore, and NotOnOrAfter elements in a SAML response.	33
Figure 5. Example of traffic forwarding through a generated endpoint to port 3000. ...	34
Figure 6. Retrieval of the authenticated user's session cookies with the help of Wireshark and ARP spoofing.	35
Figure 7. Example of setting the name, maxAge, and secure attributes for session cookies on the SP side [31] [32].	36
Figure 8. Strict transport security enablement on the SP side [32] [33].	36
Figure 9. Encrypted SAML assertion data.	36
Figure 10. Increased load of the target machine when under the 'SSL DOS' attack as seen on the system monitor.	38
Figure 11. Suricata rule to alert about excessive incoming ClientHello messages [41].	39
Figure 12. fail2ban jail configuration example for blocking IPs based on tracked Suricata logs [44] [45].	40
Figure 13. Blocked originating IPs after the 'SSL DOS' attack is repeated.	40
Figure 14. User's SAML 2.0 attribute rendering done by the SP.	41
Figure 15. Malicious code injected into the department user profile attribute on the IdP side.	41
Figure 16. Sanitization of the mutable IdP-sourced user profile attributes [50].	42
Figure 17. SP CSP statement for limiting content execution types and origins [51].	42
Figure 18. /logout endpoint request trigger through a button from a different website hosted on the adversary-controlled machine [52].	44
Figure 19. Anti-CSRF token protection implementation on the user welcoming page.	44

Figure 20. OIDC configuration on the SP side using the okta-oidc-middleware library [56].	45
Figure 21. Initial GET request to the IdP's authorization server by the SP.	45
Figure 22. GET request to the SP's callback, which contains the authorization code issued by the IdP.....	46
Figure 23. Retrieval of the OIDC authorization code with the help of Wireshark and ARP spoofing.	46
Figure 24. autocannon query for running many concurrent connections to the SP's callback route [60].	47
Figure 25. Test results generated by autocannon.....	48
Figure 26. Suricata alert rule used to catch excessive authorization callback requests [61].	48
Figure 27. XSS redirection script in the user_info object, coming from the IdP.....	49
Figure 28. GET request to the SP's callback, which contains the state parameter.....	49
Figure 29. LDAPS connection configuration on the SP side using passport-ldapauth [64].	50
Figure 30. Okta IdP LDAP interface authentication failure due to missing OTP code.	51
Figure 31. Successful brute force attack against a service user with a weak password.	51
Figure 32. Configuration of a secure password policy as per recommendations [72]. ..	51
Figure 33. Submitted form data by the end user caught with Wireshark.	52

List of tables

Table 1. Various attacks and prevention techniques on authorization and authentication protocol implementations.	26
Table 2. XSW attacks supported by the SAML Raider extension and corresponding result of each attack [19].	32
Table 3. Signature-based attacks and their corresponding results.....	37

1 Introduction

This chapter introduces the topic of the thesis, states the motivation for conducting the research work, and states the research problem and the research goal. This chapter also sets the scope and describes the overall contribution of the thesis.

1.1 Motivation

The motivation of this thesis is based on the importance of secure authentication and authorization processes and the analysis and evaluation of vulnerabilities introduced by incorrectly implemented authentication and authorization protocols in IT systems and applications with existing user management. Between 1st of November 2022 and 31st of October 2023, there has been a 180 percent increase in attacks focused on exploiting web application vulnerabilities, making them as relevant as ransomware attacks and taking one-third of all recorded data breaches. Furthermore, around 40 percent of all recorded data breaches involved credentials-based attacks [1]. During a 2021 survey, identification and authentication failures as well as broken access control were in the top 10 of OWASP's web application security risks [2]. Other research publications on the topic are either limited to theoretical descriptions of identified vulnerabilities or conduct sparse practical showcases of the attacks based off such vulnerabilities and are possibly outdated [3]. The author of this thesis will focus on filling this gap by conducting a practical evaluation of the stated attacks, which are based on trying to exploit sample implementations of authentication and authorization protocols, in a structured and expanded manner.

Moreover, the topic of this thesis is indirectly associated with the author's professional background in IAM and serves personal interest.

1.2 Research Problem Statement

Authentication and authorization are the core user identification and access control concepts of IT systems and web applications. With the constant rise of various cyberthreat

dangers, it becomes necessary to analyse which of the existing attacks and prevention strategies are successful against implementations of various authentication and authorization protocols. Such contribution would in turn allow for identification of the most up-to-date attacks and their severity, as well as prevention techniques of the existing vulnerabilities, which could help SPs increase protection and prevent reputational, monetary, and structural harm caused by possible adversaries.

Based on the problem statement, the corresponding research questions are the following for this thesis:

- [RQ1] What are the attacks on implementations of modern authentication and authorization protocols?
- [RQ1.1] Which of the stated attacks are successful against sample authentication and authorization protocol implementations?
- [RQ 1.2] How severe are the vulnerabilities that the successful attacks exploit?
- [RQ2] What are the prevention techniques of attacks on sample implementations of modern authentication and authorization protocols?
- [RQ2.1] Which of the stated prevention techniques are successful at preventing existing attacks?

1.3 Research Goal

The main goal of this thesis is to determine how successful the attack and prevention techniques are on sample implementations of modern authentication and authorization protocols.

Tasks supporting the research goal are the following:

- Setup of a testing environment to simulate attacks and prevention techniques on implementations of modern authentication and authorization protocols.
- Analysis of the effectiveness of the attacks in the current time by determining their applicability in a testing environment.
- Analysis of the effectiveness of prevention techniques correlating to successful attacks in a testing environment.

1.4 Research Scope

The scope of this thesis will be limited to the simulation of attacks and prevention techniques on sample implementations of modern authentication and authorization protocols in a testing environment. There will be no attack simulation done on live environments. The attacks and prevention techniques taken into account are found in the existing research literature. In-development authentication and authorization protocols are out of the scope of this research, as it is currently not possible to determine the risks associated with said protocols (due to them not being used in enterprise/live environments yet).

1.5 Contribution Overview

The contribution of the thesis is the following:

- Systematic literature review on the topic of authentication and authorization protocols, vulnerability assessment, and penetration testing, which is available on GitHub [3].
- Structured and hands-on analysis of various attacks on implementations of authentication and authorization protocols.
- Severity calculation of successful attacks.
- Analysis of various prevention techniques on successful attacks conducted on implementations of authentication and authorization protocols.

2 Background

For conducting the literature review, various digital databases were used for finding both practical and theoretical research literature. Databases used include Scopus, ACM DL, IEEE Xplore, and ScienceDirect. The following keywords were used for conducting a more directed search of the relevant literature: *authentication protocols, authorization protocols, vulnerability assessment, vulnerability analysis, security assessment, security analysis, security evaluation, penetration testing*. The oldest publication year of the queried literature was set to 2017, and the newest publication year was set to 2025 – this year range was selected to retrieve both the time-verified publications and to give way for more innovative and experimental publications. Exclusion and inclusion criteria were stated for choosing the most suitable publications. As an example, publications not written in English were excluded from the review [3].

The goals of the literature review were the following:

1. To understand the current state of the art in the topic of authentication and authorization, more specifically to identify which are the currently used authentication and authorization protocols,
2. To understand what the known attacks on authentication and authorization protocol implementations are and what the possible prevention techniques of these attacks are.
3. To understand which tools can be used for conducting vulnerability assessments and penetration testing of authentication and authorization protocols.

Relevant research literature highlights:

Thapa et al., “Security Analysis of User Authentication and Methods” [4] describes various authentication and authorization protocols, such as SAML 2.0, OAuth 2.0, LDAP, and states known conducted attacks. No practical showcase or relevancy analysis of the stated attacks has been done.

Maidine et al., “Cloud Identity Management Mechanisms and Issues” [5] showcases various authentication and authorization protocols, such as SAML 2.0, OAuth 2.0, OIDC, and describes various attacks on insecure implementations of these protocols. Provides limited examples of the described attacks. Prevention techniques are described but are not evaluated.

Naik et al., “Securing digital identities in the cloud by selecting an apposite Federated Identity Management from SAML, OAuth and OpenID Connect” [6] provides a more technical description of the SAML 2.0, OAuth 2.0, OIDC protocols and various attacks but does not explicitly showcase these attacks in a practical manner.

Indu et al., “Identity and access management in cloud environment: Mechanisms and challenges” [5] explains the main points of IAM processes, discusses authentication and authorization protocols and various relevant attacks, but does not showcase them in a hands-on approach and does not state how practical they are at the current time.

Sharma et al., “Security Analysis of OAuth 2.0 Implementation” [7] describes the OAuth 2.0 protocol, highlights some vulnerabilities that can arise from incorrect implementation, and showcases them through a practical approach. The work is limited to OAuth 2.0.

Motero et al., “On Attacking Kerberos Authentication Protocol in Windows Active Directory Services: A Practical Survey” [8] describes the flow of the Kerberos authentication protocol and explains various attacks on the protocol through a hands-on approach. The work is limited to Kerberos.

During the literature review, it was identified that most research papers do not provide practical evaluations of conducting attacks and applying prevention techniques on various implementations of authentication and authorization protocols.

Attacks showcased, such as in the case of OAuth 2.0 and Kerberos, are considered as sufficient by the author for the current case, and thus the practical evaluation of these protocols is excluded from the scope of this thesis. Moreover, OAuth 2.0’s capability is limited to authorization, and Kerberos provides only authentication – it was instead decided to focus on the protocols that provide both functionalities [7] [8].

2.1 Authentication

Authentication is the process of digital identity verification and is used in both on-premises and cloud systems. There are many available methods for an individual person to verify that the stated digital identity belongs to them. The amount and the complexity of these methods depend on the configured access controls of the relevant system. As an example, more basic authentication methods include identity verification through entering a valid set of credentials (i.e., username and password), while more advanced methods include the usage of biometrics [4] [5] [9].

In the current time, usage of only a set of basic credentials for user authentication is discouraged due to the proneness of this method to brute force attacks; simpler authentication methods should be hardened by additional usage of multifactor authentication. In modern enterprise settings, user authentication is handled by authentication protocols. Currently popular authentication protocols are: SAML 2.0, OIDC, LDAP, and Kerberos [5] [8].

2.2 Authorization

Authorization is the process of managing an authenticated user's access to various resources. Regular users should have limited access to a system's resources, while privileged users (i.e., users with the role of administrator) can have full access to manage a system. There exist various methods and models for improving authorization processes; one of the many access control models – ABAC, assigns appropriate levels of access to users based on their profile attributes in the IdP. Authorization is supported by OAuth 2.0, SAML 2.0, and LDAP in addition to authentication [9].

2.3 Single Sign-On

In an enterprise setting, users usually require access to multiple SPs for conducting work-related tasks. SSO allows users to authenticate to SPs without having to provide valid credentials to each. Instead, a centralized system, called an IdP, stores all the user identities and relevant authorization information like user attributes, access group memberships, and SP authorization grants, which in turn allows for enterprise-level centralized IAM. A user is required only to authenticate to the IdP for accessing trusted SPs through SSO. This approach greatly simplifies the authentication process for end-users and improves security, as it removes the need for end-users to manage different credentials for each SP [4] [5] [10].

A trust relationship must be set up between the IdP and the SP(s) for SSO to work. How this relationship is set up depends on the authentication protocol, but it usually involves an exchange of some kind of metadata (i.e., an XML metadata document for SAML and client ID and client secret for OIDC) between the IdP and the SP. The end goal that must be reached is that the IdP and the SP need to be able to securely communicate, and interchange authenticated user's session information [10].

2.4 Protocols

This paragraph describes the various authentication and authorization protocols identified through the relevant research literature.

2.4.1 SAML 2.0

SAML 2.0 is an authentication and authorization open standard protocol that allows user agent data (SAML assertion) transmission in the format of XML messages for conducting web-based SSO. The protocol can be made access control-compliant by adding various user attributes to SAML assertions. The trust relationship between the IdP and the SP is set up by exchanging SAML metadata files, which contain identifiable information of the providers. SAML metadata files are not considered confidential, as they contain only public key-based information and can be shared between the providers through open channels. The assertions are signed by the IdP, which holds the signing certificate containing the public and private key pair. The SAML assertion signatures are verified by the SP based on the information shared through the IdP's metadata file. The protocol supports both IdP-initiated and SP-initiated authentication, as well as basic user provisioning through initial authentication to the SP. SAML 2.0 is the latest version of the protocol, which was approved back in 2005 by OASIS [4] [5] [6] [11].

SAML 2.0 is vulnerable to XSW attacks because of its use of XML for transmission of SAML messages, which allows their modification while keeping the signatures valid. Prevention of such an attack can be achieved by implementing message integrity, tampering checks, and signature validation. Another possible XML-based attack is XXE, which allows for remote code execution by exploiting XML parsers. The most relevant prevention technique for XXE is to disable XML external entities [5].

A good security practice is to include validity timestamps in SAML messages, which state for how long these messages are valid – if this feature is not implemented, then the messages never expire. This means that such messages become susceptible to malicious reuse through MitM attacks and possible timestamp manipulations. SAML messages should be signed by a genuine CA; otherwise, they become susceptible to replay attacks. Encryption of the SAML messages through secure channels like HTTPS greatly helps in mitigating message timestamp manipulation and replay attacks [4].

Like any web-based service, SAML authentication and authorization implementations are prone to DoS/DDoS attacks. If any of the participating parties are overloaded with

requests, then they won't be able to process any SAML messages at the time, causing SSO to fail. Firewalls and DoS/DDoS protection services can be used to mitigate the possibilities of a successful DoS/DDoS attack. Various implementations of SAML 2.0 by SPs may also be vulnerable to XSS and CSRF attacks if they are not following secure session management, input validation, and CSRF protection techniques; however, this issue is universal for all web-based SSO protocols [4] [5] [6].

2.4.2 OAuth 2.0

OAuth 2.0 is a protocol for managing user authorization flows and sharing user data between providers through standardized APIs in web and mobile (native) applications. User authorization data is transmitted through authorization codes, which are then exchanged for user access tokens. Access tokens are usually either in JSON, JWT, or XML format and are generated by the authorization server (IdP's side) on behalf of the client (SP). Once the access token is retrieved by the client, it can be used to fetch protected resource data from the resource server. Access tokens are usually configured to expire in a short period of time, and a new access token can be retrieved by querying the authorization server with a valid refresh token. It should be noted that in the case of OAuth 2.0, the IdP does not necessarily have to be centralized to authorize access on behalf of the user to some SP. Popular services like Google, Facebook, etc., usually take on the role of an IdP as an example, the most important aspect of this is that there must be an existing user data source already present. As such, this protocol cannot be considered as pure-SSO compared to the other protocols described in this paragraph. OAuth 2.0 is one of the newer protocols and was released in 2012 [4] [5] [6] [7] [9].

Usage of the 307 HTTP status code for redirecting during the authorization flow can possibly leak user data because this method of redirection preserves payloads from previous requests to the IdP, which may contain user credentials [5] [7]. This issue can be mitigated by using the 303 HTTP status code instead, as this method of redirection does not preserve payloads from previous requests [5].

An IdP mix-up attack is possible when a malicious IdP catches the authorization requests made to a legitimate IdP authorization server and sends a response to the SP. The SP then makes a token request to the malicious IdP's authorization server with the client secret value. This attack is only possible when the SP uses the same redirect URIs for multiple IdPs and the adversary can intercept requests between the SP and the legitimate IdP authorization server [5] [7]. Implementation of data encryption through HTTPS and usage

of different redirect URIs for different IdPs, as well as authorization server whitelisting with request data validation, mitigates this attack [5].

Similarly to SAML 2.0, OAuth 2.0 implementations can be prone to XSS, DoS/DDoS, and CSRF attacks if not implemented following best security practices for web application development [5] [6]. MitM attacks can be mitigated by configuring PKCE for the OAuth 2.0 authorization code flow [7].

2.4.3 OIDC

OIDC is an authentication protocol for SSO built on top of OAuth 2.0 to facilitate both user authentication and authorization flows. In addition to having an access token, a new token type is introduced, which is in the format of JWT and is regarded as an ID token. ID tokens are digitally signed and contain identifiable user information. IdP- and SP-initiated authentication is supported; once an authenticated session is established, the IdP's authorization server forwards the access and ID tokens to the SP when presented with a valid authorization code. Client ID and client secret (provided by the IdP) are used as secret information for establishing a trusted relationship between the IdP and the SP. OIDC was released in 2014 [5] [6] [7] [11].

Since OIDC is closely tied to OAuth 2.0, it is prone to the same type of attacks, including the 307-redirect attack, IdP mix-up, DoS/DDoS, CSRF, MitM, and XSS [5] [7].

2.4.4 LDAP

LDAP is a centralized active directory access protocol that can support both authentication and authorization. During authentication, a bind request is created to the LDAP server by an LDAP client with the authentication and authorization details of a user. If the data is correct, then the server responds with a successful bind response to the client and authenticates the user. Compared to other provided protocols, LDAP is often used for implementing IAM in on-premises services and networks. One limitation of on-premises LDAP implementation is that it cannot be used as a universal solution for SSO, as there are too many various requirements for SSO protocols in the current time, and it is not possible to configure all of them on an on-premises active directory [4] [5] [11] .

Depending on the way handling of LDAP requests is done by the client application, it can become prone to LDAP injections. Sanitizing and validating inputs on the application's side mitigates the possibility of such an attack becoming successful [4] [5].

Additionally, LDAP is prone brute to force attacks, and the protocol by itself does not support MFA. However, configuring rate-limiting for LDAP bind requests and enforcing secure password policies can help in preventing this attack [4] [5].

As already mentioned in previously reviewed protocols, LDAP implementations can as well be prone to DoS/DDoS and MitM attacks if no prevention techniques are present [4] [5].

2.4.5 Kerberos

Kerberos is an SSO authentication protocol used mostly in Windows-based networks and domains. One of the relevant security aspects of this protocol is that it allows for client-server identity verification over insecure communication channels by using symmetric cryptography. A centralized authentication server (KDC) is used for verifying user identities based on the correctness of the submitted credentials and their presence in the domain's active directory, which in turn forwards data tokens labelled as "tickets". Initially, the authentication server responds with a ticket-granting ticket and a session key. The ticket-granting ticket contains data about the ticket itself and the user's identity. This ticket specifically can be used to retrieve a service ticket from the ticket-granting server by the client. A service can then verify whether the user is authorized to access its resources based on the service ticket and the session key; if so, it will fulfil the client's request. This protocol was developed by MIT, and its current version is v5 - its last update was in 2005 [8].

The pass the ticket attack allows for extraction of a service ticket or a ticket-granting ticket from the memory of a compromised machine in the domain, which in turn allows for bypassing regular authentication. Enabling credential guard, limiting the number of administrator users, and enforcing secure password policies help in mitigating this attack [8].

During the overpass the hash attack, an adversary obtains the NTLM password hash of a user. The hash is then used to retrieve a valid ticket-granting ticket. The adversary can then use the ticket to authenticate to some service. Usage of credential guard and enforcement of group policies can help with the mitigation of this vulnerability [8].

A golden ticket attack happens when an adversary steals the KRBTGT user's NTLM password hash. This hash allows an adversary to sign and create ticket-granting tickets, which in turn can be used to access services. Enforcement of secure password policies and limiting privileges of users mitigates the attack [8].

Another ticket-based attack is the silver ticket attack, which, when successful, can grant access to certain domain-specific services by forging a valid TGS. The adversary can retrieve the NTLM hash of a compromised service account and use that hash to forge TGS. Enforcing secure password policies and end-user privilege control can help in mitigating the silver ticket attack [8].

‘Kerberoasting’ is a TGS-targeted attack that allows retrieving service user passwords. The adversary requests a service ticket, and as service tickets are encrypted with the service user’s NTLM hash, the adversary can crack the NTLM hash through brute force. Enforcing secure password rules, using only group-managed service users, and disabling the usage of deprecated protocols such as NTLM in the domain mitigate this attack [8]. Other types of attacks are possible, such as unrestricted delegation attacks, restricted delegation attacks, resource-based restricted delegation attacks, and bronze bit and brute force attacks – each focusing on various security misconfigurations that can happen within a domain [8].

2.5 Vulnerability Assessment

Vulnerability assessment is the process of identifying vulnerabilities and security flaws in specific (or all) components of a given system. Enterprises comply in conducting vulnerability assessments of their systems on a regular basis to find corresponding weaknesses and to strengthen their security posture by remediating the identified vulnerabilities. Various components of the system are scanned for known CVEs. Vulnerability assessments can be conducted either by an external third party specializing in providing such services or by internal auditors. Once the vulnerabilities are identified, they also must be prioritized based on their severity. Vulnerability assessment is conducted using various tools and can be automated and/or manual [12] [13].

2.6 Penetration testing

Penetration testing extends the process of vulnerability assessment and conducts practical exploitation of the identified vulnerabilities for security evaluation. Additionally, false positives from automated vulnerability scanning tools can be revealed during penetration testing, and the possible impact of attacks on vulnerabilities can be simulated. Penetration

testing methods can be white, black, or grey box, depending on the knowledge about the system [12] [13].

2.7 VAPT

VAPT combines both processes of vulnerability assessment and penetration testing and ranges from initial information gathering about the system to conducting exploitation-based attacks [12].

Various VAPT tools exist, including NMAP, Nessus, Burp Suite, Accuentix, Metasploit, The Harvester, Wireshark, Zed Attack Proxy, Beef, and SQLMAP [12].

2.8 Summary

Modern enterprise environments rely extensively on authentication and authorization protocols such as SAML 2.0, OAuth 2.0, OIDC, LDAP, and Kerberos for both cloud-based and on-premises systems. Due to various security weaknesses in basic credential-based authentication, best practices depend on enhancing user authentication and identification processes through multifactor authentication and centralized identity management solutions such as IdPs and various SSO configurations. Implementations that follow security practices of these protocols, as well as configuration of security policies in the domain, are needed to mitigate various attack vectors that target authentication and authorization protocols. Each attack or vulnerability needs specific mitigation or prevention strategies to ensure security. Security analyses encompass various methodologies, including vulnerability assessment, which identifies and reports vulnerabilities without active exploitation, and penetration testing, which involves practically exploiting identified vulnerabilities to assess their possible impact and detect false positives from automated tools. Combining both vulnerability assessment and penetration testing into a unified process named VAPT allows for comprehensive evaluation - from initial information gathering about a system to exploitation-based assessments. For conducting VAPT, tools such as NMAP, Nessus, Burp Suite, Metasploit, etc., are used.

Table 1 summarizes the identified attacks and corresponding prevention techniques on authentication and authorization protocol implementations.

Attack	Affected protocol(s)	Prevention(s)
--------	----------------------	---------------

XSW	SAML 2.0	Implementation of message integrity and tampering checks.
XXE	SAML 2.0	Disabling of XML external entities.
Replay attack	SAML 2.0	SAML message validity timestamp setting. Implementation of HTTPS/TLS.
MitM attack	OAuth 2.0, SAML 2.0, OIDC, LDAP	HTTPS/TLS implementation, IdP whitelisting, implementation of PKCE for the authorization code flow.
Invalid signature attack	SAML 2.0	Genuine CA signing of SAML messages and validity checking. Implementation of HTTPS/TLS.
DoS/DDoS attack	OAuth 2.0, SAML 2.0, OIDC, LDAP	IP address filtering, request limiting and validation, WAF and/or DoS/DDoS prevention service implementation.
XSS	SAML 2.0, OAuth 2.0, OIDC	Input validation, sanitization, usage of the content security policy, usage of <code>HttpOnly</code> flag.
CSRF	OAuth 2.0, SAML 2.0, OIDC	Usage of state property with user-linked information, secure session policies and CSRF protection implementation.
307-redirect attack	OAuth 2.0, OIDC	Usage of HTTP 303 instead of HTTP 307 for user redirection.
IdP mix-up attack	OAuth 2.0, OIDC	Usage of HTTPS/TLS, authorization server whitelisting, redirect URI validation.
Unauthorized code injection attack	LDAP	Input validation and sanitization.
Brute force attack	LDAP	Configuration of rate-limiting for LDAP bind requests and enforcement of secure password policies.
Pass the ticket attack	Kerberos	Enablement of Credential Guard, limiting of the number of administrator users and

		enforcement of secure password policies.
Overpass the hash attack	Kerberos	Enablement of Credential Guard and enforcement of group policies.
Golden ticket attack	Kerberos	Enforcement of secure password policies and limiting privileges of users (access control).
Silver ticket attack	Kerberos	Enforcement of secure password policies and limiting privileges of users (access control).
'Kerberoasting' attack	Kerberos	Enforcement of secure password rules, usage of only group-managed service users (access control) and disabling the usage of deprecated protocols.

Table 1. Various attacks and prevention techniques on authorization and authentication protocol implementations.

3 Contribution

This chapter expands on the author's thesis contribution – explanation of how the identified protocols were implemented in a testing environment and the results of conducting various attacks on these implementations. Introduction of relevant prevention techniques was executed depending on whether any of the attacks were successful or not.

3.1 Methodology

This paragraph describes the chosen methodology for the thesis contribution: an overview of the research task, relevant research methods used, and the research process description along with stated limitations.

3.1.1 Task Overview

The main task of this thesis focuses on the practicality evaluation of the attacks and corresponding prevention methods on implementations of authentication and authorization protocols, both of which were identified in the relevant research literature. For a successful task resolution, a testing environment is required to be provisioned along with a basic SP-IdP relationship to enable SSO. The testing environment was decided to be of a virtual nature due to its simplicity and ease of use. The testing (practical evaluation of attacks and prevention techniques) process is semi-automated, consisting of manual evaluation by the author along with the use of VAPT tools for automated scanning and execution of known attacks. The basic SPs are built on the example of open-source authentication libraries and are simple in nature – the main functional goal of the SP in this case is to enable the SSO flow for evaluating the attacks and prevention techniques on authentication and authorization protocol implementation. The IdP was chosen based on its current popularity and the author's experience with the product – it is a paid, cloud-based provider that also openly provides a payment-free tenant that was used for this task.

3.1.2 Evaluation Process Overview

The testing environment is provisioned with the use of Oracle VM VirtualBox – three virtual machines, each with bridged network adapters on the 192.168.0.1/24 network, 2 GB of RAM, and 2 CPU cores of the host machine:

- Kali Linux (attacker/adversary), OS version: 2025.1a, IP: 192.168.0.134.
- Debian Linux (SP), OS version: 12.10.0, IP: 192.168.0.133.
- Ubuntu Linux (client), OS version: 20.04.6, IP: 192.168.0.132.

The cloud-based IdP tenant is available on the internet through a generated URL. The author has administrative access to the tenant for conducting tasks necessary for the SP-IdP connection set up.

Per iteration, the authentication and authorization protocol implementations are configured on the SP virtual machine and is available on port 3000. Additionally, a malicious web server on port 3000 of the adversary's virtual machine is running to demonstrate some upcoming proof-of-concepts.

Each identified attack is conducted either on the SP or the client virtual machine(s) by the adversary's virtual machine, depending on the attack scenario – the results are then recorded. The attacks are conducted in a mixed-way method with the usage of both automated tools and manual evaluation, depending on the attack. The tools are introduced throughout the chapter. If an attack was successful – then the relevant severity is assessed using the CVSS score [14] – then the identified prevention method(s) is implemented, and the attack is once again conducted – the results are then compared for a success/failure rate identification. For each new type of attack, the previously existing prevention techniques are disabled; the final version of the SP has all the previously described prevention techniques implemented.

Figure 1 shows the evaluation process diagram per each attack.

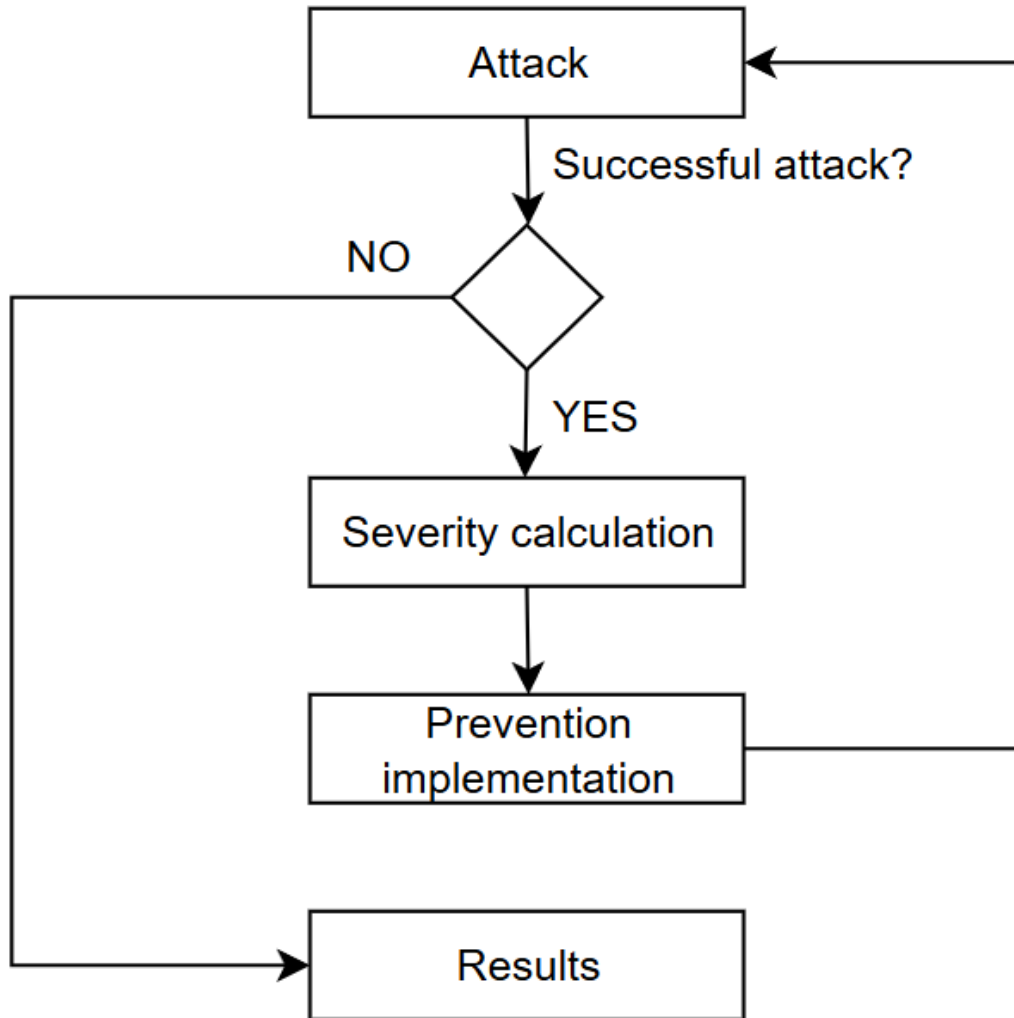


Figure 1. Evaluation process diagram.

3.1.3 Research Methodology

The evaluation process primarily uses an experimental-empirical type of research approach, complemented by descriptive case studies. The following research methods are used:

- Case study:
 - Provisioning of SP-IdP sample implementation(s) in a virtual environment for authentication and authorization protocols.
- Experimental:
 - Hands-on execution of various attacks using known VAPT tools and manual evaluation of the implementation(s) in a white box manner. Incremental implementation of prevention techniques in a virtual environment.

- Empirical:
 - Recording of attack success/failure results before vs. after implementing prevention techniques. Vulnerability scoring on successful attacks is calculated by using CVSS version 4.0 [14].

3.1.4 Limitations

During the evaluation process, the following limitations are considered:

- Since some prevention techniques require the implementation of SSL, for the simulation purposes a payment-free version of an HTTPS proxy is used instead.
- Due to the vulnerability reporting policy of the IdP [15] and certain aspects of it, it is decided not to conduct any form of direct attacks on the IdP testing tenant.

3.2 SAML 2.0 Protocol Implementation

For the implementation of the SAML 2.0 protocol in a virtual testing environment, the `passport-saml` and `express.js` libraries for `node.js` were used to set up the SP running on port 3000. `passport-saml` is a SAML authentication extension for the `passport` middleware [16], which was chosen due to its popularity and ease of configuration. `passport` was ranked in 10th place among the top `node.js` libraries for backend development in 2024 [17]. The `express.js` library is used for handling user sessions and various endpoints on the SP side, which is considered the most popular `node.js` library [17].

Okta was used as an IdP for setting up the IdP-SP trust relationship for SAML SSO. Okta is being continuously recognized as one of the leading IAM platforms on the market [18] and provides a payment-free developer tenant for evaluation purposes [19].

Figure 2 shows how the trust relationship with an Okta IdP is set up on the SP side; the entry point value and SSO certificate are retrieved from the SAML 2.0 application created in the Okta environment.

```

passport.use(new SamlStrategy(
{
  path: '/login/callback',
  entryPoint: 'https://dev-██████.okta.com/app/dev-██████_samltestingapp_1/exko4j7f6z00KdAUH5d7/sso/saml',
  issuer: 'http://192.168.0.133:3000',
  cert: fs.readFileSync('app/okta-test-saml.cert', 'utf-8'),
},
function (profile, done) {
  return done(null, profile);
}
));

```

Figure 2. Sample local configuration of trust relationship with an Okta IdP using the passport-saml library [16].

For evaluating the applicability of the existing attacks on the protocol implementation, ZAP was used to scan for web-based vulnerabilities. Burp Suite Community Edition with the SAML Raider extension was also used, as it allows for built-in manipulation of web requests and SAML messages along with mimicking valid metadata certificates [20]. Other tools like Wireshark were used for conducting general attacks that target the overall protocol implementation and not some specific aspect of it.

3.2.1 XSW Attack

XSW attacks target the way that SAML messages are handled by the SP. Since an SP must both validate the signature and process the message itself, these two actions can be separated into two processes, depending on the implementation. During an XSW attack, the adversary expects that the signature validation and message processing are two separate procedures. The adversary manipulates the structure of the SAML message by injecting specifically chosen elements/objects that still pass the signature validation and are handled during message processing as if they were legitimate [21].

SAML Raider supports eight different types of XSW attacks; the description and result of each attack are described in **Table 2**.

Attack	Description	Result
XSW1	Clones a malicious SAML response object and injects it after the valid signature object.	Invalid signature error.
XSW2	Clones a malicious SAML response object and injects it before the valid signature object.	Invalid signature error.
XSW3	Clones a malicious SAML assertion and injects it before the valid assertion object.	Invalid signature and multiple assertions errors.
XSW4	Clones a malicious SAML assertion and injects it as a top level of the valid assertion object.	Invalid signature error.

XSW5	Injects a malicious assertion object into a valid SAML response object; the malicious assertion is injected as a top level of a valid signature object.	Invalid signature and multiple assertions error.
XSW6	Injects a malicious assertion object into a valid SAML response object; the malicious assertion is injected as a top level of a valid signature, and the valid assertion is moved as a part of a valid signature object.	Invalid signature error.
XSW7	Injects an “Extensions” object with a cloned malicious assertion to the SAML response object.	Invalid signature error.
XSW8	Injects an “Object” object that contains the copy of a valid assertion with its signature removed.	Invalid signature error.

Table 2. XSW attacks supported by the SAML Raider extension and corresponding result of each attack [19].

The results are expected, as the built-in `validateSignature` function in `passport-saml` rejects the malicious elements/objects by both validating the signature and checking the number of assertions in the SAML message.

3.2.2 XXE Attack

XXE attacks exploit the way SAML messages are processed by the SP’s XML parser. If the XML parser has document type definitions enabled and/or if the messages are accepted and processed without prior evaluation, then an XXE becomes possible. During a successful XXE attack, the adversary can force the parser to fetch the contents of various files on the machine hosting the application or to make arbitrary requests to malicious web resources [22].

To identify the possibility of conducting an XXE attack on the `passport-saml` SP, various types of XML entities were injected into the SAML response message, which was coming from the IdP to the SP. An endpoint `/validate` was hosted on the Kali Linux virtual machine, port 3000, for receiving successful XXE requests.

Figure 3 shows one of the attempted XXE injections into a SAML response message.

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE foo [ <!ENTITY % xxe SYSTEM
"http://192.168.0.134:3000/validate"> %xxe; ]>
```

Figure 3. Sample XXE injected into a SAML response message [20].

For proof-of-concept, the entity definition was altered multiple times, the `SYSTEM` element was replaced with `PUBLIC`, instead of using a URL - the `file:///` destination was used

instead, etc. However, none of the attempted attacks were successful – no requests to the specified endpoint were made and no files were fetched.

The unsuccessful XXE attacks are justified, as `saml-passport` uses the `xml2js` library [23] for parsing XMLs – this library ignores custom document type definitions and, as such, is not vulnerable to XXE attacks.

3.2.3 Replay Attack

Replay attacks become possible when there are no SAML message validity checks either on the SP, IdP, or both sides. An adversary can intercept valid SAML messages and reuse them for gaining unauthorized access to the SP [24]. SAML messages originating from Okta are valid for 5 minutes; this is defined by the `IssueInstant`, `NotBefore`, and `NotOnOrAfter` XML elements inside the message [25].

Figure 4 shows an example of how the elements are stated in a SAML response.

```
<?xml version="1.0" encoding="UTF-8"?><saml2p: . . . IssueInstant="2025-04-13T16:10:40.048Z" Version="2.0"
xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol">
. . .
<saml2:Conditions NotBefore="2025-04-13T16:05:40.724Z" NotOnOrAfter="2025-04-13T16:15:40.724Z" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
```

Figure 4. Example of `IssueInstant`, `NotBefore`, and `NotOnOrAfter` elements in a SAML response.

However, 5 minutes can be considered as quite a long validity period, and it is typically assumed that a message should be valid for only 1 minute [24]. The security requirements may vary depending on the scenario, however.

Moreover, an adversary on the same network can catch the base64-encoded SAML messages originating from Okta by using Wireshark or SAML Raider and easily decode and replay them – sending a replayed SAML response message allows for user impersonation if the message is still valid. The calculated CVSS score for this vulnerability is 6 (medium).

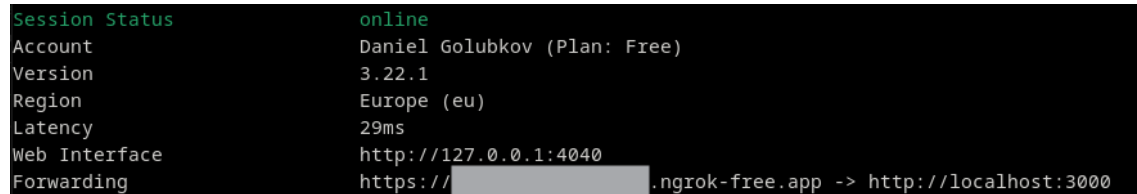
To harden the SAML message acceptance control on the SP's side, `saml-passport` supports the definition of `validateInResponseTo: true` and `acceptedClockSkewMs: -240000` attributes into the `Sam1Strategy` constructor. `validateInResponseTo` assures that the IdP's SAML response has the same `InResponseTo` identifier as the initial assertion coming from the SP; additionally, it prevents the same `InResponseTo` identifier reuse. `acceptedClockSkewMs` variable, when set to a negative number, lowers the number of

milliseconds for which the SAML response originating from the IdP is valid. A value of -240000 enforces the SP to only accept SAML responses that are not older than 1 minute [16].

Previously it was possible to intercept and resend SAML responses to the SP through SAML Raider, but it indeed becomes no longer possible with the newly stated attributes. When trying to reuse the same SAML response, the `InResponseTo is not valid` error is outputted by the SP, and when receiving a response that is older than 1 minute, the error `SAML assertion expired` is returned.

HTTPS can be implemented on the SP's side to prevent adversaries from catching SAML messages. A service called ngrok can be used to secure the connection between the SP and the IdP. ngrok is a reverse proxy that allows for secure transmission of requests through an ngrok agent to the SP from a generated endpoint with a specific identifier [26]. Thus, ngrok can be used to encrypt network traffic and hide it from potential adversaries. It should be noted that the `Sam1Strategy` constructor on the SP side and the SAML application settings on the IdP side should be updated to use the new URL.

Figure 5 shows an example of how an ngrok-generated endpoint can be used to forward network traffic to port 3000.



```
Session Status      online
Account            Daniel Golubkov (Plan: Free)
Version            3.22.1
Region             Europe (eu)
Latency            29ms
Web Interface      http://127.0.0.1:4040
Forwarding          https://[redacted].ngrok-free.app -> http://localhost:3000
```

Figure 5. Example of traffic forwarding through a generated endpoint to port 3000.

While ngrok is sufficient for the current case, in live environments TLS/HTTPS should be configured on the SP side using a valid certificate signed by a trusted CA [27].

3.2.4 MitM Attack

MitM is a type of attack where the adversary passively or actively intercepts the data transmission between two different parties and tries to conduct various attacks based on the retrieved information [28]. It can be assumed that so far, previously described attacks in this chapter can all be considered as various iterations of a MitM attack. Because in all the cases it is expected that the adversary gets in between the SP-IdP SSO flow and either alters data during transmission or tries various techniques to exploit possible vulnerabilities between the two parties, in example of SAML Raider. Additionally,

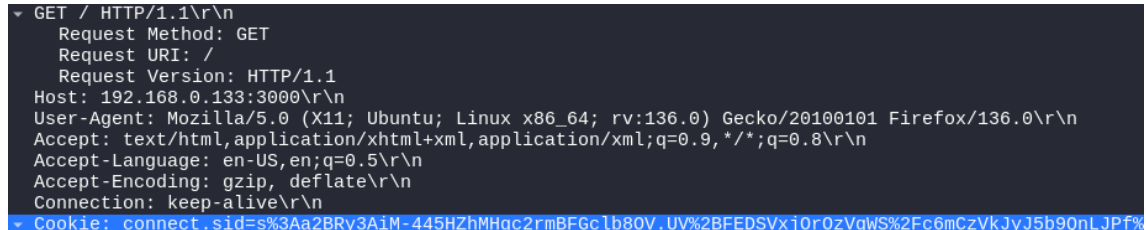
various measures are implemented on the SP side to reduce the possibility of a successful MitM attack. These include signature validation, ignoring of custom document type definitions, InResponseTo and SAML message timestamp validation, and implementation of HTTPS through the ngrok proxy.

Perhaps one of the more well-known MitM attacks is session hijacking, where an adversary steals session cookies of an authenticated end user and impersonates them. In 2023 an adversary managed to hijack sessions of multiple Okta customers by gaining access to the Okta support system through a social engineering attack [29].

Wireshark can be used by the adversary to successfully get the victim's plaintext connect.sid cookie value after receiving a valid SAML response and hijack their session if TLS/HTTPS is not set up. The calculated CVSS score for this vulnerability is 6 (medium).

A prerequisite for such an attack is that the adversary must conduct an ARP spoofing attack through their machine. For the current case, the arpspoof command bundled in the dsniff package with Kali Linux [30] was used for conducting the ARP spoofing attack.

Figure 6 shows the example of session cookie retrieval by listening to the network traffic between the victim and target SP.



```
GET / HTTP/1.1\r\n
Request Method: GET
Request URI: /
Request Version: HTTP/1.1
Host: 192.168.0.133:3000\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:136.0) Gecko/20100101 Firefox/136.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n
Cookie: connect.sid=s%3Aa2BRy3AiM-445HZhMHGc2rmBFGc1b80V.UV%2BFEDSVxj0r0zVqWS%2Fc6mCzVkJyJ5b9QnLJPf%
```

Figure 6. Retrieval of the authenticated user's session cookies with the help of Wireshark and ARP spoofing.

To decrease the possibility of a successful session hijacking attack, stricter authentication policies can be configured on the IdP side, meaning that the user would need to re-authenticate with their credentials when navigating to sensitive content.

On the SP side, secure session handling techniques can be implemented. Setting the lifetime of a cookie is beneficial, as then the end-user would need to re-authenticate after a set amount of time. A typical workday is around 8 hours, so the value of maxAge: 28800000 (milliseconds) can be set for the cookie object inside the session constructor. To prevent the establishment of an authenticated session through the insecure HTTP, secure: true can be set [31] [32]. Additionally, the default session cookie name should be changed to follow the best security recommendations [27], in the current case, it was changed to Session_ID.

Figure 7 shows an example of setting the session cookie name and the previously stated `maxAge` and `secure` attributes on the SP side.

```
app.set('trust proxy', 1);  
.  
.  
.  
app.use(session({ name: Session_ID, cookie: { maxAge: 28800000, secure: true  
})));
```

Figure 7. Example of setting the name, `maxAge`, and `secure` attributes for session cookies on the SP side [31] [32].

It is also useful to enable strict transport security on the SP side to enforce only HTTPS connections [32]. The `max-age` variable is set to 63072000, making the strict transport security valid for 2 years as per security recommendations [33].

Figure 8 shows how the strict transport policy can be enabled on the SP side.

```
app.use((req, res, next) => {  
    res.setHeader('Strict-Transport-Security', 'max-age=63072000');  
    next();  
});
```

Figure 8. Strict transport security enablement on the SP side [32] [33].

In a case where configuring HTTPS on the SP side is not possible, Okta also supports configuring SAML assertion encryption on transmission to prevent reading and tampering by possible adversaries [34]. The `openssl` library can be used to generate the RSA public/private keypair and a self-signed certificate [35]. The public key is used for encryption; the private key is used for decryption and creation of the self-signed certificate. The private key is added to the `SamlStrategy` constructor on the SP side as the `decryptionPvk` variable [16] and the self-signed certificate is uploaded to Okta [34]. With this configuration the `saml2:Assertion` object is replaced with `saml2:EncryptedAssertion`, and the contained assertion data is encrypted inside the `CipherValue` object.

Figure 9 shows how the assertion data is encrypted inside the SAML message.

```
<xenc:CipherValue>QeReh. . .I8WH1</xenc:CipherValue>
```

Figure 9. Encrypted SAML assertion data.

3.2.5 Invalid Signature Attack

Signature attacks are possible when signatures are not correctly validated by the SP, which in turn allows for assertion data manipulations and signature forging [24]. As an

example, if the signature is not validated at all, then the adversary can modify the NameID value in the assertion and impersonate any user that exists inside the SP. SAML Raider has the capability to conduct various signature-based attacks, and they were tested against the SP.

Table 3 showcases various signature-based attacks and the corresponding results of each attack.

Attack	Description	Result
Signature removal	Removal of the signature from the SAML assertion.	Invalid signature error.
Signature alteration	Alteration of the signature in the SAML assertion.	Invalid signature error.
Signature re-use	Re-use of the previously accepted signature linked to a different SAML assertion.	Invalid signature error.
SAML assertion re-sign	Re-signing of the assertion with a different signature from the trusted Okta certificate.	Invalid signature error.

Table 3. Signature-based attacks and their corresponding results.

Indeed, all of the specified attacks have failed. It was already identified in **3.2.1** that `passport-saml` conducts signature validation and identifies any type of external tampering.

Okta is a genuine and trusted CA that follows all the current security practices in handling SAML assertions. In the current state, the SAML signing certificate's [36] signature algorithm is set to SHA-2.

3.2.6 DoS/DDoS Attack

During a Dos attack, the adversary floods the targeted machine with network-based requests. DDoS is an improvement of such an attack, and the adversary utilizes multiple machines to conduct the attack instead of just one. One of the goals of DoS/DDoS attacks is to cause the target machine to become too busy with handling the malicious requests and make it unable to process legitimate requests in a timely manner. The alternative goal of such attacks is to cause the target machine to crash because of the high system resource utilization from handling a large number of requests [37].

To simulate a DDoS attack for this case, clones of the Kali Linux virtual machine were created; the total number of malicious machines was thus 18. For conducting the attack,

a unified collection of various bash scripts, called pentmenu, was used. pentmenu supports 12 different DoS attacks, most of which utilize various network flooding techniques [38]. Throughout the simulation, the script was run on all 18 machines and was set to target the Debian Linux machine running the SP implementation. Out of all the attacks, the ‘SSL DOS’ attack on the node.js port proved to be the most effective; this attack utilizes the openssl [39] library to create continuous TLS session establishments with the target machine. Even though there is no TLS configured on the node.js port at the current time, the target machine still must process the ClientHello message and reply with an HTTP 400 Bad Request response. When run on all the malicious machines at the same time, this attack caused the usage of one of the CPU cores on the target machine to spike to 98.8%. Also, the processing speed of the target machine decreased drastically, causing even simple actions such as navigating throughout the operating system difficult.

Figure 10 shows the increased load on the target machine when under the ‘SSL DOS’ attack.

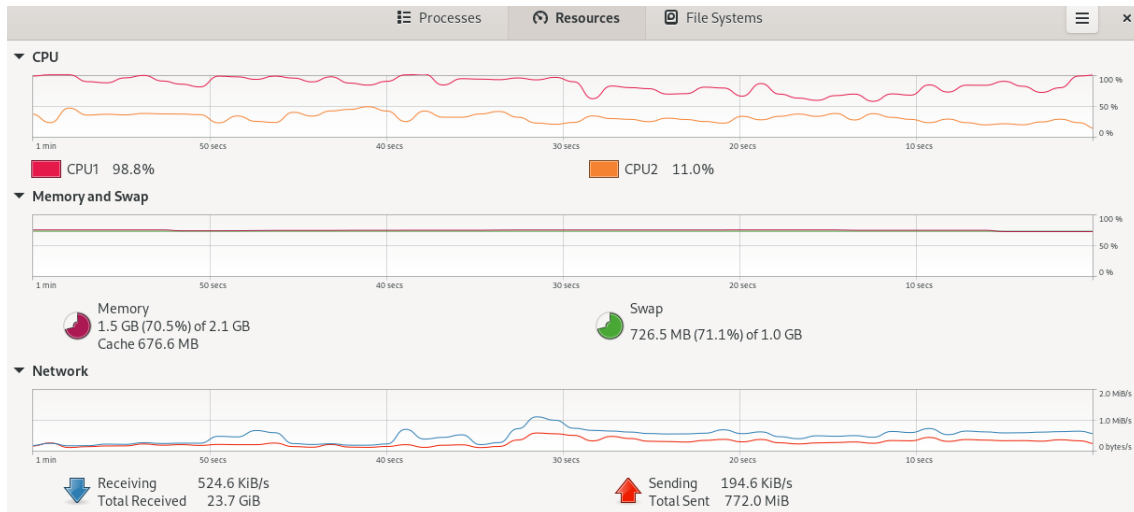


Figure 10. Increased load of the target machine when under the ‘SSL DOS’ attack as seen on the system monitor.

Furthermore, the server response time has increased significantly. When accessing the SP URL by a client machine, 14.21 seconds were needed to load the initial sign-in page on average. It is assumed that with more machines participating in the attack the SP machine would halt entirely. The calculated CVSS score for this vulnerability is 8.2 (high).

To prevent such an attack from becoming successful, a threat detection service can be implemented, which would allow for network traffic analysis and alerting of incoming attacks. For this case, Suricata was chosen for its availability and ease of setup. Suricata

allows for the setting up of threat alerts and even has its own IPS mode where it can drop malicious traffic [40].

In Suricata, network threat detection rules are set, which are similar to those of regular firewalls. To alert about excessive ClientHello incoming messages, a rule specified in **Figure 11** is set [41].

```
alert tls any any -> 192.168.0.133 3000 (msg: "EXCESSIVE CLIENTHELLO
ATTEMPTS!"; content: "|16 03|"; offset: 0; depth: 2; content: "|01|"; offset:
5; depth: 1; threshold: type threshold, track by_src, count 5, seconds 10;
priority: 1; sid: 1000003; rev:1;)
```

Figure 11. Suricata rule to alert about excessive incoming ClientHello messages [41].

This rule checks incoming packets for header bytes 0x16, 0x03 (in record header), and 0x01 (in handshake header), which are specific to a ClientHello message as per the TLS protocol specification [42]. ClientHello is sent by the client machine only during the initial TLS session establishment, and then there isn't normally a justified reason why more than five concurrent ClientHello messages should be processed from the same client in the span of 10 seconds. And thus, the rule states that if there are more than 5 such messages in the span of 10 seconds, then an alert is generated in Suricata logs.

The rule can also be edited to drop the traffic instead of just sending alerts. Malicious IPs can be blocked for some specific time, which is a standard traffic handling practice during DoS/DDoS attacks.

While IP blocking is chosen to be configured for this case, in a real-world scenario it is up to the SP to determine how to handle malicious traffic - various other protection methods can be implemented, like geo-blocking [43].

A basic Suricata setup is limited in the possibilities of handling traffic and does not allow for time-based IP blocking. Hence, a server protection service called fail2ban can be configured to read the specified Suricata alert from a log file and block the malicious IPs for a set time using a filter/jail setup [44] [45].

A filter with a failregex value of `^."src_ip":"."."signature_id":1000003\b.*$` is created – this filter explicitly states that fail2ban should look for "signature_id":1000003 in the log file, which corresponds to the sid stated for the Suricata rule. A sample of the jail configuration file for blocking IPs based on Suricata logs is also provided and shown in **Figure 12** with the name `suricata-tls-dos` [44] [45].

```

. . .
logpath = /var/log/suricata/eve.json
. . .
action = iptables[name=TLS-DOS, port=3000, protocol=tcp, blocktype=DROP]

```

Figure 12. fail2ban jail configuration example for blocking IPs based on tracked Suricata logs [44] [45].

This jail configuration adds a firewall DROP rule for every IP that triggered the Suricata alert, which in turn was added to the JSON Suricata log file `eve.json`. For this case, the IP block timer is set to 5 minutes (300 seconds).

With Suricata and fail2ban running and all the stated configurations enabled, the ‘SSL DOS’ attack is repeated. The malicious traffic is successfully identified, and the originating IPs are blocked.

Figure 13 shows the result of the attack – all IPs of machines participating in the attack were blocked.

```

Status for the jail: suricata-tls-dos
|- Filter
|   |- Currently failed: 0
|   |- Total failed:    146
|   '- File list:      /var/log/suricata/eve.json
'- Actions
    |- Currently banned: 18
    |- Total banned:    27
    '- Banned IP list:  192.168.0.136 192.168.0.145
                        168.0.149 192.168.0.157 192.168.0.139 192.168.0.140

```

Figure 13. Blocked originating IPs after the 'SSL DOS' attack is repeated.

While there was a momentary CPU usage spike when the attack began, as soon as the alert requirements were met, the IPs were blocked and the CPU usage never exceeded 20%. For this case, Suricata and fail2ban were configured to specifically prevent SSL DoS attacks; these tools can be further tweaked to possibly prevent any type of DoS/DDoS attack.

3.2.7 XSS Attack

XSS attacks occur when the adversary can inject malicious code into a target web service, most often by using the standard JS `<script>` tags, which is then processed and executed. Code injections can be of two types: browser-based injections and server-based injections. Usually, this happens when user inputs are not properly sanitized or when the content security policy is not implemented. XSS attacks can be stored, meaning that the

malicious code resides on the server hosting the web service and is executed during fetching by the user. Additionally, the attacks can be categorized as reflected, which means that the adversary modifies a valid URL to include their code and shares it with a user [46] [47]. Some examples of successful XSS attacks include session cookie hijacking and user redirections to malicious websites [46].

It should be noted that session cookie hijacking is not possible for this case, as express-session by default has the `HttpOnly` value set to `true` for session cookies, which prevents their retrieval through XSS [31].

The ZAP scanner was used to scan the routes of the SP to check for possible XSS attack vectors, but none were found. However, a general alert with a medium severity level for 10038 Content Security Policy (CSP) Header Not Set was outputted. Since the confirmation exists that CSP is indeed not set, then an XSS attack can be tested against the only mutable aspect of the rendering done by the SP – SAML 2.0 user attributes. It is common for SPs to display various information to users based on their profile in the IdP, including name, email, etc.

For this case, the default `express.js` HTTP response method `res.send()` [48] is used to display profile information to an authenticated user, more specifically – their identifier (email in this case) and their department. The values of both attributes are handled by the IdP. The authenticated user's profile attribute rendering by the SP is shown in **Figure 14**.

```
res.send(`<h1>Hello,      ${req.user.nameID}!      Your      department      is:
${req.user.department}</h1><a href="/logout"><button>Logout</button></a>`);
```

Figure 14. User's SAML 2.0 attribute rendering done by the SP.

It should be noted that it is not possible for the adversary to modify these attributes inside the SAML 2.0 assertion and send it to the SP, as that would invalidate the existing signature. However, the SP will accept any value of the attribute provided by the IdP. In a possible scenario where the IdP becomes compromised and the adversary can manipulate user profile attributes; malicious code can be injected instead of a typical string representing a user's department or any other similar attribute.

Figure 15 shows the assumed malicious code that was injected into the user's department profile attribute on the IdP side – this code will redirect a user to a different website, which is controlled by the adversary running on the Kali Linux machine.

```
<script>window.location.href='http://192.168.0.134:3000/evil'</script>
```

Figure 15. Malicious code injected into the department user profile attribute on the IdP side.

The `<script>` tags are retained during the whole SAML 2.0 flow, and once they reach the SP, the code inside the tags is executed, causing the authenticated user to be redirected to the malicious website. However, for this attack to be viable the adversary must have administrative access to the IdP.

This is only a proof-of-concept showcase – in a real-world scenario, the adversary can modify the script to execute any type of malicious code. Such an attack is possible when there is no input sanitization on the IdP and SP sides, and the SP lacks CSP. The calculated CVSS score for this vulnerability is 5.7 (medium).

To prevent the stated XSS attack from succeeding, sanitization along with CSP will be implemented on the SP side. The combination of both methods is considered an effective method in handling all types of XSS attacks. However, for form data and genuine input implementations, validation should also additionally be configured as per recommendations [49].

Sanitization is the method of removing potentially malicious logic and characters from inputs to prevent XSS attacks and unwanted behaviours [49]. For this case, an `express.js` sanitization library named `express-xss-sanitizer` is used for sanitizing the IdP-sourced user profile attribute values.

Figure 16 shows the sanitization of the `req.user.nameID` and `req.user.department` attribute values [50].

```
. . .  
req.user.nameID = sanitize(req.user.nameID);  
req.user.department = sanitize(req.user.department);  
. . .
```

Figure 16. Sanitization of the mutable IdP-sourced user profile attributes [50].

CSP defines what types of resources a target website can load and from what origin [51]. A custom CSP is set for `express.js` that restricts the execution of the inline and external scripts, which is seen on **Figure 17**.

```
res.setHeader('Content-Security-Policy', "default-src 'self'; frame-ancestors  
'self'; script-src 'self'; connect-src 'self'; img-src 'self'; style-src  
'self'; base-uri 'self'; form-action 'self';");
```

Figure 17. SP CSP statement for limiting content execution types and origins [51].

With the following implementation, XSS attacks will be either blocked in the backend by the sanitization library or on the browser side by the CSP.

3.2.8 CSRF Attack

CSRF attacks are used to make authenticated end users unknowingly perform actions on behalf of the adversary inside the target web service. During a CSRF attack, a malicious URL, which points to the adversary-controlled website, is usually delivered to the user by some type of phishing approach – if the user chooses to access the URL, then whichever malicious code is on the adversary-controlled website is executed. This is possible because requests made through the browser automatically contain session data, like cookies, which is added during cross-site requests.

The adversary can target various sensitive resources on the target web service, and the attack is less effective for low-privilege end users. As an example, the adversary can cause the end user to perform a transaction if the targeted web service is an online banking service without CSRF protection. Some of the most popular methods of CSRF attack prevention are the usage of SameSite cookies and implementation of anti-CSRF tokens [52] [53].

ZAP was once again used to scan for any alerts related to CSRF, and 10054-1 Cookie without SameSite Attribute with low severity, was outputted by the scanner – this confirms that the SameSite cookie is not set on the SP side. While the SAML 2.0 flow itself in the current implementation cannot be targeted by CSRF due to bound signature validation, other state-changing elements of the SP can still be.

The most straightforward method of hardening a web service against CSRF is to implement the SameSite attribute into user session cookies. It is recommended to either set this attribute to strict or lax, where the first variant prohibits all types of cross-site cookie-based navigation and the second variant allows cookie-based navigation from external sites if they contain a URL pointing to the origin web service [32] [54]. SameSite: "lax" attribute is set in the cookie constructor for the SP. It should be noted that the "strict" value cannot be set for this implementation as SAML 2.0 flow functions on the logic of cross-site requests; setting the attribute to strict will break the flow, and the users will not be able to SSO into the SP.

For conducting the proof-of-concept, an HTML page is hosted on the Kali Linux machine, which contains a button [52] that, once pressed, makes a request to the SP's /logout route. When successful, this action will cause the currently authenticated user to be signed out and prove that a CSRF vulnerability exists on the SP side.

Figure 18 shows the button configuration, which will make a GET request to the SP's /logout endpoint.

```
<form action="192.168.0.133:3000/logout" method="GET">
<input type = "submit" value ="logout"/>
</form>
```

Figure 18. /logout endpoint request trigger through a button from a different website hosted on the adversary-controlled machine [52].

However, even with SameSite: "lax" set, the attack is still successful because "lax" allows for the session cookie to be sent on URL-based requests, as mentioned previously [32] [54]. The calculated CVSS score for this vulnerability is 2.1 (low).

A more robust method for preventing CSRF attacks is using anti-CSRF tokens, which are usually some types of unpredictably generated data that is bound to the current user session. Trying to perform an anti-CSRF token-protected request from a different website would fail because the adversary cannot provide the user's CSRF token [53]. Additionally, implementation of the InResponseTo validation in the SAML response will help prevent login-based CSRF if the adversary would modify their website to send SAML responses linked to malicious user identities – this was described in 3.2.3.

For the anti-CSRF token implementation, a library named `lusca` [55] is added on the SP side, and the accepted method for /logout is changed from GET to POST, as the CSRF token needs to be included as a hidden value on the user welcoming page [53]. It must be noted that anti-CSRF token protection needs to be disabled on the /login/callback route, as the IdP can never provide the anti-CSRF token to the SP.

Figure 19 shows the anti-CSRF token protection implementation on the user welcoming page which also contains the link to the /logout route.

```
<input type='hidden' name='_csrf' value='${req.csrfToken()}'>
```

Figure 19. Anti-CSRF token protection implementation on the user welcoming page.

With the implementation complete, the user navigates to the malicious website and clicks the button to navigate to the /logout route; however, now the user is not logged out, and an error 'CSRF token missing' is outputted by the SP. Thus, the CSRF attack prevention techniques are successfully configured.

3.3 OIDC Protocol Implementation

The OIDC protocol is implemented on the SP side by using the `okta-oidc-middleware` [56] along with the `express.js` library, which is a similar setup to that described in paragraph 3.2. The `okta-oidc-middleware` library is chosen because it easily integrates with the Okta IdP and is considered a suitable evaluation subject for determining whether it is prone to known attacks.

Figure 20 shows the sample SP configuration using the `okta-oidc-middleware` library [56], the client ID and client secret values are retrieved from a sample OIDC application, which is created in Okta with the authorization code grant type.

```
const oidc = new ExpressOIDC({
  issuer: 'https://dev-████████.okta.com/oauth2/default',
  client_id: '00aop6fw3cIzXH8VP5d7',
  client_secret: '████████████████████████████████████████████████████████████████████████████████',
  appBaseUrl: 'http://192.168.0.133:3000',
  scope: 'openid profile email'
});
```

Figure 20. OIDC configuration on the SP side using the `okta-oidc-middleware` library [56].

The authorization callback route (sign-in redirect URI) is present at `/authorization-code/callback`.

3.3.1 307-Redirect Attack

For conducting the 307-redirect attack, Burp Suite along with the ‘intercept’ function was used for modifying the requests to the IdP and SP and identifying whether the 307 vulnerability exists or not.

In addition to the possible user data leakage described in 2.4.2, an adversary can potentially conduct redirect URI manipulations to redirect the authenticated user to a malicious website [57].

During OIDC authentication, an initial GET request is made by the SP to the IdP’s authorization server – this request contains the `redirect_uri` value, which is shown in **Figure 21**.

```
GET /oauth2/default/v1/authorize?. . .
redirect_uri=http%3A%2F%2F192.168.0.133%3A3000. . .
```

Figure 21. Initial GET request to the IdP’s authorization server by the SP.

Attempting to modify the value of `redirect_uri` causes Okta IdP to respond with HTTP 400 – Bad Request, which proves that redirect URI validation is occurring and thus it is not possible to cause malicious redirection this way.

When the IdP forwards the user's authorization code to the SP, another GET request is made; this is shown in **Figure 22**.

```
GET 192.168.0.133:3000/authorization-code/callback?code=25_ . . .
```

Figure 22. GET request to the SP's callback, which contains the authorization code issued by the IdP.

Modifying the host value for this request also does not yield any successful results from the adversary's point of view, as the requests are only allowed to be directed to the redirect URI value set in the Okta OIDC application's settings.

Furthermore, no user-identifying attributes are leaked during the requests, and thus the 307-redirect attack is not successful for this implementation.

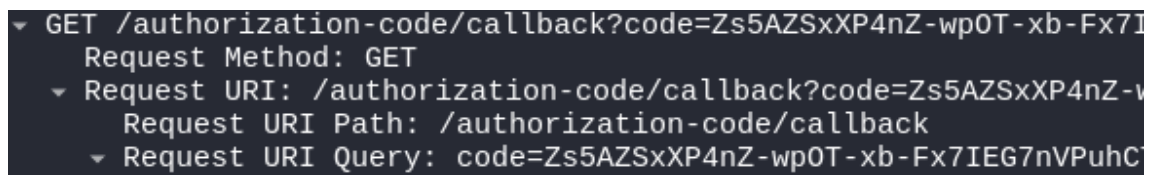
3.3.2 IdP Mix-up Attack

The `okta-oidc-middleware` library only allows for a single `ExpressOIDC` constructor definition [56] and thus it is not possible to configure connections to multiple IdPs – hence it is also not possible to test this attack in the current configuration.

3.3.3 MitM Attack

Similarly to the case described in 3.2.4, it is possible to use Wireshark to track the traffic coming to an ARP-spoofed client machine and get the authenticated user's session cookies.

Also, the adversary can retrieve the authorization code from the network traffic, which is shown in **Figure 23**.



```
▼ GET /authorization-code/callback?code=Zs5AZSxXP4nZ-wp0T-xb-Fx7I
  Request Method: GET
  ▼ Request URI: /authorization-code/callback?code=Zs5AZSxXP4nZ-w
    Request URI Path: /authorization-code/callback
    ▼ Request URI Query: code=Zs5AZSxXP4nZ-wp0T-xb-Fx7IEG7nVPuhC
```

Figure 23. Retrieval of the OIDC authorization code with the help of Wireshark and ARP spoofing.

However, this code is not enough to retrieve the user's ID and access tokens, as the client secret is also required, and the validity period of this code is only 300 seconds. If the adversary still tries to fetch the ID/access tokens through the IdP's `/oauth2/v1/token` endpoint, the error `invalid client` will be returned as a response [58]. Reusing an authorization code is also not effective, as then the SP responds with the `authorization code is invalid or expired` error.

The only scenario where the retrieved authorization code can be used for impersonating a user is if the `GET /authorization-code/callback` request, containing the authorization

code value, is dropped for a targeted user during the OIDC flow and the code value is retrieved. Then the adversary can start the OIDC flow with their own user, and during the callback request, replace their user's authorization code with the still valid, hijacked authorization code.

This scenario is highly unlikely, as the prerequisites are quite specific:

- The adversary must have the credentials to an existing user in the IdP.
- The adversary must have the ability to listen to, modify and drop traffic between the SP and IdP.

The calculated CVSS score for this vulnerability is 6 (medium).

The prevention methods, including traffic encryption through a TLS/HTTPS proxy and session cookie hijacking prevention techniques, were described in **3.2.4**.

Implementation of PKCE, which adds additional verification steps in the form of PKCE code generation and evaluation [59], is not possible on the SP side, as `okta/oidc-middleware` does not support it in its current state [56]. Instead of PKCE, implementation of TLS/HTTPS can be considered as a more effective MitM prevention method in the current case, as it introduces data transmission confidentiality, which invalidates the previously described attack scenario.

3.3.4 DoS/DDoS Attack

To test whether the implementation is vulnerable to DoS/DDoS, the SP's `/authorization-code/callback` route was targeted with dummy data. To conduct the evaluation for this case, the autocannon benchmarking tool was used for creating many concurrent requests to the SP. Figure 24 shows the used query, which specifies that 2500 concurrent connections need to be created, and the test is run for the duration of 120 seconds [60].

```
autocannon -c 2500 -d 120 http://192.168.0.133:3000/authorization-  
code/callback?code=bFBvfvpFfnQ2vMgz53DSwHgoKiyJ18cr2BduAkELsQNA&state=qWl0pyza  
NIWmMnk_ti8nXis3Jz40gEruCSmnRjcgRx8
```

Figure 24. autocannon query for running many concurrent connections to the SP's callback route [60].

Each dummy request to the callback route causes the SP to respond with the `did not find expected authorization request details in session error`.

Running the command on only a single Kali Linux virtual machine caused the SP's resource consumption to increase greatly, with CPU1 and CPU2 showing 79.7% and 84.7% load percentages correspondingly. Also, the website response times increased

significantly – averaging 15 seconds. The calculated CVSS score for this vulnerability is 8.2 (high).

Figure 25 shows the results of the test, which was generated by autocannon - the average latency during the test was 5 seconds, and there were 3000 timeouts.

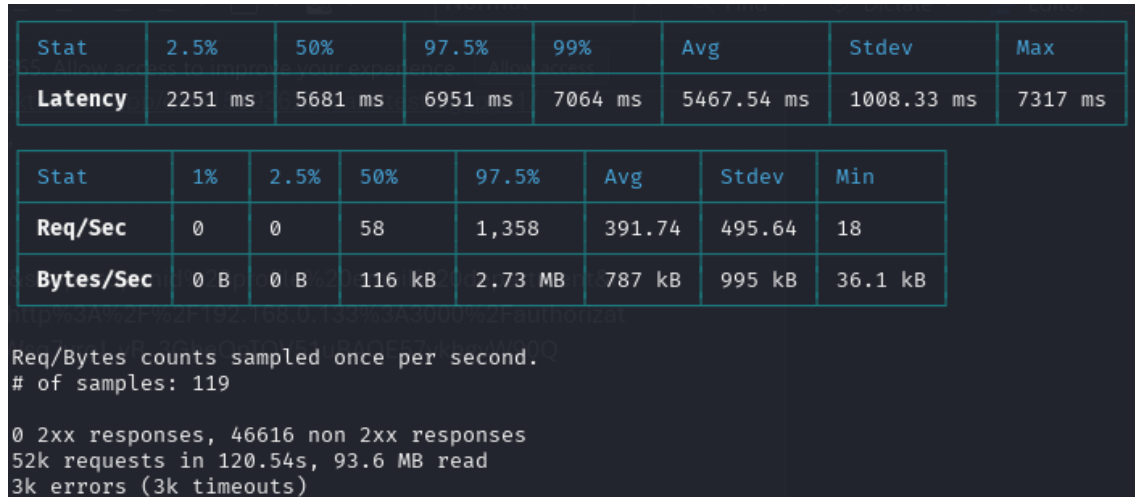


Figure 25. Test results generated by autocannon.

Similarly to 3.2.6, Suricata along with fail2ban was used to block IPs from where the malicious requests were originating from.

Figure 26 shows the new rule used for catching the excessive requests [61], it is highly unlikely that a genuine client would make 20 requests to the callback route in the span of 5 seconds.

```
alert http any any -> 192.168.0.133 3000 (msg: "EXCESSIVE AUTHORIZATION CALLBACKS!"; flow:established,to_server; http.method; content: "GET"; http.uri; content: "authorization-code/callback"; threshold: type threshold, track by_src, count 20, seconds 5; priority: 1; sid: 1000004; rev:1;)
```

Figure 26. Suricata alert rule used to catch excessive authorization callback requests [61].

3.3.5 XSS Attack

In a similar manner to 3.2.7, it can be assumed that the IdP may allow for injection of <script> tags into various user attributes. For this case, a custom OIDC claim was created named department, which would allow the transmission of the IdP user's department value to the SP. During the OIDC flow, the value is retained inside of the user_info object, and once it reaches the SP, the stated code is executed. The calculated CVSS score for this vulnerability is 5.7 (medium).

Figure 27 shows sample user attributes received by the SP; it is seen that the XSS redirection script from 3.2.7 is successfully transmitted during the flow.


```
given_name: 'Daniel',  
family_name: 'Golubkov',  
zoneinfo: 'America/Los_Angeles',  
updated_at: 1747343234,  
email_verified: true,  
department: "<script>window.location.href='http://192.168.0.134:3000/evil'</script>"
```

Figure 27. XSS redirection script in the user_info object, coming from the IdP.

As was previously stated in **3.2.7**, implementation of sanitization techniques prevents this attack from succeeding.

3.3.6 CSRF Attack

The OIDC flow can be prone to CSRF attacks if the state parameter, which is some type of unguessable data string, is not implemented and checked during authorization [62]. As an example, an SP could potentially accept an authorization callback request, which contains the adversary's authorization code. If the SP does not protect against CSRF attacks, then the targeted user could instead be logged in with an adversary-controlled identity [63].

However, it can be seen from the request/response analysis that the state parameter exists during the flow – this can be seen in **Figure 28**.

```
GET /authorization-code/callback?code=. .  
&state=mobvEA4UsWwRLrTT7uKr0JK7EvN0x46kHxTg2JUYUQM
```

Figure 28. GET request to the SP's callback, which contains the state parameter.

The did not find expected authorization request details in session error is returned when the request to the route does not contain the correct state parameter.

As `okta-oidc-middleware` controls the state parameter even during logouts [56], conducting a CSRF attack on the SP's `/logout` route is unsuccessful – `Unauthorized` is returned.

Hence, the following implementation is not vulnerable to CSRF attacks.

3.4 LDAP Protocol Implementation

For the LDAP implementation, the `passport-ldapauth` [64] library was used for providing client authentication to the Okta IdP's LDAP interface [65]. Okta provides access to all its public-facing certificates, and the `_.ldap.okta.com.crt` can be used to support the LDAPS connection on port 636 [66] [67]. A service user on the Okta IdP side with read-only administrator permissions is added to the configuration for LDAP binding

and retrieval of IdP identities through the SP. The full configuration for the LDAPS connection is seen in **Figure 29**.

```
passport.use(new LdapStrategy({
  server: {
    url: 'ldaps://dev-[REDACTED].ldap.okta.com:636',
    bindDN: 'uid=[REDACTED],ou=users,dc=dev-[REDACTED],dc=okta,dc=com',
    bindCredentials: '[REDACTED]',
    searchBase: 'ou=users,dc=[REDACTED],dc=okta,dc=com',
    searchFilter: '(uid={{username}})',
    tlsOptions: {
      ca: [
        fs.readFileSync('app/_ldap.okta.com.crt', 'utf-8')
      ]
    }
  },
}));
```

Figure 29. LDAPS connection configuration on the SP side using passport-ldapauth [64].

It is worth mentioning that the /login route contains the login form for end-users to use for authentication.

3.4.1 Unauthorized Code Injection Attack

LDAP injections occur when user input is not properly sanitized and validated, giving potential adversaries the ability to inject malicious code into LDAP queries. The consequences of such actions include leakage of user information, authentication bypass, and privilege escalation [68].

It is known that passport-ldapauth is a wrapper for ldapauth-fork [64], and it is found that this library has built-in sanitization for user inputs to prevent LDAP injection attacks [69]. Indeed, trying to inject various queries (for example, uid=*, uid=someadmin@mail.com, etc. [68]) into the username field causes the SP to reject them and does not yield any positive results for the adversary.

3.4.2 Brute Force Attack

During a brute force attack, an adversary uses a method of exhaustive username/password pair guessing to attempt to access user accounts with a weak level of credential security. There are various ways of brute-forcing LDAP authentication – for example, a tool called hydra can be used with the following command and username/password list files set as arguments: `hydra -L users.txt -P passwords.txt ldap3://dev-{id}.ldap.okta.com:636` [70]. By default, Okta enforces MFA on user authentication [71]. This is also valid for the LDAP interface. Even if the adversary correctly guesses a

username and the password, authentication will fail without the provided OTP code – this is seen in **Figure 30**.

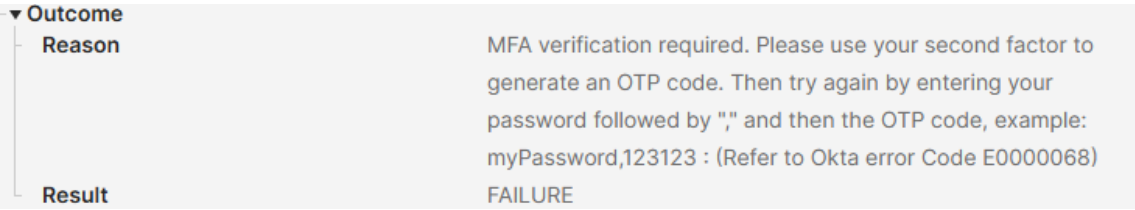


Figure 30. Okta IdP LDAP interface authentication failure due to missing OTP code.

However, the adversary could still possibly guess the credentials of the LDAP service user, which does not have MFA enforced.

Figure 31 shows a successful brute force attack against a service user with a weak password using hydra.

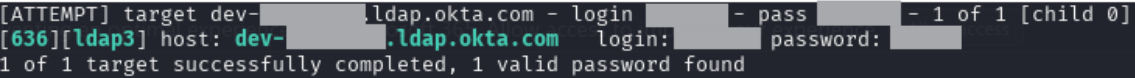


Figure 31. Successful brute force attack against a service user with a weak password.

The calculated CVSS score for this vulnerability is 8.2 (high).

As per security recommendations published by CISA, it is recommended to use long passwords with mixed-case letters along with introduced randomness.

Figure 32 shows the configuration of a secure password policy on the Okta IdP side [72].

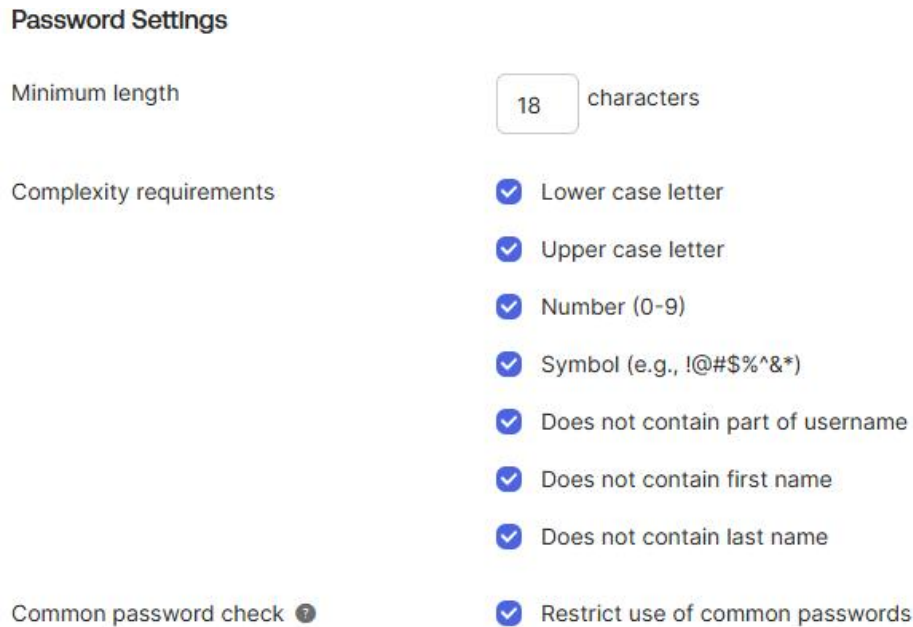


Figure 32. Configuration of a secure password policy as per recommendations [72].

Currently, it is computationally infeasible to brute force a complex password [73]; however, technology is constantly evolving, and brute-forcing complex passwords may become a reality in the near future – hence it is highly recommended to implement MFA solutions wherever possible.

On the IdP side, rate limits on the LDAP interface [74] will lower the speed of the brute force; however, they do not prevent the attack from succeeding.

3.4.3 MitM Attack

As LDAPS is used for data transmission between the SP and the IdP's LDAP interface, an adversary cannot read it due to encryption. However, it is still possible to get session cookie data along with the values of "username" and "password" when the form is submitted by the end-user because TLS/HTTPS is not implemented on the SP side. As was described in 3.2.4 and 3.3.3, ARP spoofing needs to be conducted on the end user's machine.

The submitted form data by the end-user, which was caught with Wireshark, is seen in **Figure 33**.

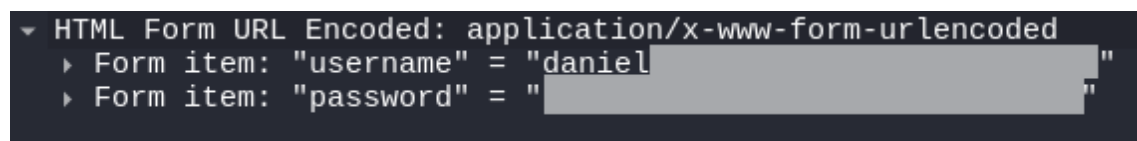


Figure 33. Submitted form data by the end user caught with Wireshark.

The calculated CVSS score for this vulnerability is 6 (medium). Consequently, the adversary can read user credentials and easily impersonate any user who submits their credentials through the login form if MFA is not enforced.

As previously discussed, implementation of TLS/HTTPS, secure session handling techniques, and strict transport security on the SP side will prevent this attack from occurring.

3.4.4 DoS/DDoS Attack

For simulating a DoS attack, the autocannon tool was once again used on the adversary's virtual machine – the configuration is similar to that stated in 3.3.4. The SP's /login route was targeted with GET requests for proof-of-concept. It was decided to exclude any type of potential user account data and POST requests to avoid pressuring the IdP's LDAP interface.

The CPU1 and CPU2 cores once again showed increased load – 85.2% and 76.5%, correspondingly. Unlike the previous tests described in **3.3.4** and **3.2.6**, during this test the SP was unresponsive to a client machine for the whole duration of the test – so it can be stated that the attack was successful. The calculated CVSS score for this vulnerability is 8.2 (high).

Similarly to previous cases, Suricata, along with fail2ban, can be used to block IPs that exceed some specific number of requests at a given timeframe.

4 Discussion

For this thesis, a practical evaluation of the identified attacks and prevention techniques was conducted. Authentication and authorization protocols that were evaluated are SAML 2.0, OIDC, and LDAP. Each protocol was implemented in a virtual testing environment through the usage of open-source libraries and a cloud-based IdP.

The attacks and prevention techniques were tested using automated tools and manual techniques.

From the eighteen existing attacks, eleven were successful. It was found that the implementation of a single prevention technique is at times not enough to fully prevent attacks (3.2.8). Rather, a combination of the existing prevention techniques is required (3.2.4, 3.2.6). Many of the commonly known attacks on authentication and authorization protocols are prevented by built-in security features of the implemented authentication libraries (3.2.1, 3.3.1, 3.4.1) or by the stated prevention techniques. In multiple cases of successful attacks, the adversary was able to impersonate the targeted user (3.2.3, 3.2.4), so it can be stated that these attacks are effective at compromising the ‘confidentiality’ aspect of SP implementations. Most of the successful attacks on existing vulnerabilities are of medium severity as per the CVSS score evaluation.

The importance of following secure web development practices for preventing successful attacks in live environments along with periodical security evaluations of web services cannot be underestimated. Even though this thesis provides practical evaluation of most of the identified attacks and prevention techniques, web developers and enterprises providing SP services still must stay up to date with recent threats and thoroughly analyse documentation about authentication services that they are implementing. As an example (as stated in 3.2.3), the `validateInResponseTo` attribute is disabled by default in the `passport-saml` library; however, it is an important addition for preventing replay attacks. During the evaluation process, it was additionally identified that data transmission encryption through HTTPS/TLS plays an important role in preventing various types of MitM attacks (3.2.4, 3.3.3, 3.4.3). Additionally, DoS/DDoS attacks remain universally effective against web services without configured protection (3.2.6, 3.3.4, 3.4.4). A few techniques, that were not identified in research literature were additionally implemented to prevent various attacks (3.2.3, 3.4.2).

The limitations of this thesis are that all the tests/practical evaluations were conducted in a virtual environment, and some of the conducted attacks can be considered as naïve – for example lack of HTTPS/TLS is quite rare in live enterprise environments. Some conducted attacks are rather targeted at web services than specific authorization and authentication protocol implementations (3.2.4, 3.2.8). The same statement is valid for the prevention techniques (CSP, secure cookie configuration).

As the author of this thesis is not a professional penetration tester nor a web developer, it is quite possible that some attack methods/vectors were missed. Additionally, it was not possible to conduct some attacks due to built-in limitations of protocol implementations (as in 3.3.2) or due to legal limitations stated by the IdP's policies.

The discussed SAML 2.0 [75], OIDC [76], and LDAP [77] implementations with configured prevention techniques are available on GitHub.

4.1 Answers to Research Questions

Answers to the research questions stated in 1.2 are the following:

- **[RQ1]** What are the attacks on implementations of modern authentication and authorization protocols?

The attacks on implementations of modern authentication and authorization protocols include:

- a) SAML 2.0: XSW attack, XXE attack, replay attack, MitM attack, invalid signature attack, DoS/DDoS attack, XSS attack, CSRF attack.
- b) OIDC/OAuth 2.0: 307-redirect attack, IdP mix-up attack, MitM attack, DoS/DDoS attack, XSS attack, CSRF attack.
- c) LDAP: unauthorized code injection attack, brute force attack, MitM attack and Dos/DDoS attack.
- d) Kerberos: pass the ticket attack, overpass the hash attack, golden ticket attack, silver ticket attack, 'kerberoasting' attack.

Table 1 provides a more detailed summary.

- **[RQ1.1]** Which of the stated attacks are successful against sample authentication and authorization protocol implementations?

The following attacks were found to be successful against sample authentication and authorization protocol implementations:

- a) SAML 2.0: replay attack, MitM attack, DoS/DDoS attack, XSS attack, CSRF attack.
- b) OIDC: MitM attack, DoS/DDoS attack, XSS attack.
- c) LDAP: brute force attack, MitM attack, DoS/DDoS attack.

As was stated in chapter 2, Kerberos and OAuth 2.0 were removed from the practical evaluation.

- **[RQ 1.2]** How severe are the vulnerabilities that the successful attacks exploit?
The severity of each vulnerability that the successful attacks exploit per implementation is the following:

- a) SAML 2.0:
 - i. Replay attack – 6 (medium).
 - ii. MitM attack – 6 (medium).
 - iii. DoS/DDoS attack – 8.2 (high).
 - iv. XSS attack – 5.7 (medium).
 - v. CSRF attack – 2.1 (low).
- b) OIDC:
 - i. MitM attack – 6 (medium).
 - ii. DoS/DDoS attack - 8.2 (high).
 - iii. XSS attack - 5.7 (medium).
- c) LDAP:
 - i. brute force attack – 8.2 (high).
 - ii. MitM attack – 6 (medium).
 - iii. DoS/DDoS attack - 8.2 (high).

- **[RQ2]** What are the prevention techniques of attacks on implementations of modern authentication and authorization protocols?

The prevention techniques of attacks on implementations of modern authentication and authorization protocols are the following:

- a) SAML 2.0:
 - i. XSW attack - implementation of message integrity and tampering checks.
 - ii. XXE attack - disabling of XML external entities.
 - iii. Replay attack - SAML message validity timestamp setting and implementation of HTTPS/TLS.

- iv. MitM attack - HTTPS/TLS implementation.
- v. Invalid signature attack - genuine CA signing of SAML messages and validity checking. Implementation of HTTPS/TLS.
- vi. DoS/DDoS attack - IP address filtering, request limiting and validation, WAF and/or DoS/DDoS prevention service implementation.
- vii. XSS attack - input validation, sanitization, usage of the content security policy, usage of `HttpOnly` flag.
- viii. CSRF attack - secure session policies and CSRF protection implementation.

b) OIDC/OAuth 2.0:

- i. 307-redirect attack - usage of HTTP 303 instead of HTTP 307 for user redirection.
- ii. IdP mix-up attack - usage of HTTPS/TLS, authorization server whitelisting, redirect URI validation.
- iii. MitM attack - HTTPS/TLS implementation, IdP whitelisting, implementation of PKCE for the authorization code flow.
- iv. DoS/DDoS attack - IP address filtering, request limiting and validation, WAF and/or DoS/DDoS prevention service implementation.
- v. XSS attack - input validation, sanitization, usage of the content security policy, usage of `HttpOnly` flag.
- vi. CSRF attack - usage of state property with user-linked information.

c) LDAP:

- i. Unauthorized code injection attack - input validation and sanitization.
- ii. Brute force attack - configuration of rate-limiting for LDAP bind requests and enforcement of secure password policies.
- iii. MitM attack - HTTPS/TLS implementation.
- iv. Dos/DDoS attack - IP address filtering, request limiting and validation, WAF and/or DoS/DDoS prevention service implementation.

d) Kerberos:

- i. Pass the ticket attack - enablement of credential guard, limiting of the number of administrator users and enforcement of secure password policies.
 - ii. Overpass the hash attack - enablement of credential guard and enforcement of group policies.
 - iii. Golden ticket attack - enforcement of secure password policies and limiting privileges of users (access control).
 - iv. Silver ticket attack - enforcement of secure password policies and limiting privileges of users (access control).
 - v. 'Kerberoasting' attack - enforcement of secure password rules, usage of only group-managed service users (access control) and disabling the usage of deprecated protocols.
- **[RQ2.1]** Which of the stated prevention techniques are successful at preventing existing attacks?

From the stated prevention techniques, the following were considered as successful at preventing existing attacks:

a) SAML 2.0:

- i. XSW attack - implementation of message integrity and tampering checks.
- ii. XXE attack - disabling of XML external entities.
- iii. Replay attack - SAML message validity timestamp setting and implementation of HTTPS/TLS.
- iv. MitM attack - HTTPS/TLS implementation.
- v. Invalid signature attack - genuine CA signing of SAML messages and validity checking. Implementation of HTTPS/TLS.
- vi. DoS/DDoS attack - IP address filtering, request limiting and validation, WAF and/or DoS/DDoS prevention service implementation.
- vii. XSS attack - input validation, sanitization, usage of the content security policy, usage of HttpOnly flag.
- viii. CSRF attack - secure session policies and CSRF protection implementation.

b) OIDC:

- i. 307-redirect attack - usage of HTTP 303 instead of HTTP 307 for user redirection.
- ii. MitM attack - HTTPS/TLS implementation, IdP whitelisting.
- iii. DoS/DDoS attack - IP address filtering, request limiting and validation, WAF and/or DoS/DDoS prevention service implementation.
- iv. XSS attack - input validation, sanitization, usage of the content security policy, usage of HttpOnly flag.
- v. CSRF attack - usage of state property with user-linked information.

c) LDAP:

- i. Unauthorized code injection attack - input validation and sanitization.
- ii. Brute force attack - enforcement of secure password policies.
- iii. MitM attack - HTTPS/TLS implementation.
- iv. Dos/DDoS attack - IP address filtering, request limiting and validation, WAF and/or DoS/DDoS prevention service implementation.

It should be noted that additional prevention techniques were implemented throughout the practical evaluation, these include:

a) SAML:

- i. Replay attack – inResponseTo attribute validation in SAML responses.
- ii. MitM attack – signature validation, disabling of XML external entities, inResponseTo and SAML message timestamp validation, SAML assertion encryption.

b) OIDC: 307-redirect attack - redirect_uri validation.

c) LDAP: brute force attack – MFA implementation.

For all protocols – a combination of implementing strict authentication policies, secure session handling techniques and strict transport security implementation has shown to be an effective addition at preventing MitM attacks.

4.2 Future Work

As many of the conducted tests were either semi-automated with the help of tools like ZAP and Burp Suite or manual, future work may involve creating a centralized and automated tool for conducting attacks on implementations of authentication and authorization protocols, which could also automatically implement the relevant prevention techniques. As an extension, the automated tool can be based on large language models, or more open-source libraries can be taken in for further analysis.

5 Conclusion

The main objective of this thesis was to evaluate various attack and prevention techniques on authentication and authorization protocol implementations in a practical manner. For all these protocols, open-source web-based libraries were used to integrate sample SPs into a virtual testing environment, where the practical evaluation was conducted.

The attacks and prevention techniques used during the practical evaluation were obtained from the relevant research literature and only targeted the sample SPs to avoid potential legal issues. The practical evaluation involved both manual and automated techniques. Throughout the practical evaluation, code samples and attack/prevention technique examples are provided for better understanding of how various attacks and prevention methods function. Additionally, the severities of successful attacks were evaluated.

While many protocol-specific attack prevention techniques are already built into open-source libraries, more than half of the existing attacks were successful on average. Most of the successful attacks targeted the web service aspect of the implementation and not the protocol. Furthermore, configuration of single prevention techniques is at times not enough; hence, it is beneficial to combine them for an increased level of service hardening.

Continuously reviewing related documentations, analysing potential security threats, and hardening web services remain as important aspects in ensuring service provider security.

References

- [1] Verizon Business, “2024 Data Breach Investigations Report,” 2024. [Online]. Available:
<https://www.verizon.com/business/resources/reports/dbir/2023/summary-of-findings/>.
- [2] The OWASP Foundation, “Top 10 Web Application Security Risks,” 2021. [Online]. Available: <https://owasp.org/www-project-top-ten/>.
- [3] D. Golubkov, “Vulnerability Assessment of Modern Authentication and Authorization Protocols; Literature review report,” 2024. [Online]. Available: https://github.com/dangol9/auth-authz-lit-rev/blob/main/lit_rev.pdf.
- [4] A. Thapa, C. S. Dhapola and H. Saini, “Security Analysis of User Authentication and Methods,” *Proceedings of the 2022 Fourteenth International Conference on Contemporary Computing (IC3-2022)*, p. 564–572, 2022.
- [5] K. Maidine and A. El-Yahyaoui, “Cloud Identity Management Mechanisms and Issues,” *2023 IEEE 6th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*, pp. 1-9, 2023.
- [6] N. Naik and P. Jenkins, “Securing digital identities in the cloud by selecting an apposite Federated Identity Management from SAML, OAuth and OpenID Connect,” *2017 11th International Conference on Research Challenges in Information Science (RCIS)*, pp. 163-174, 2017.
- [7] S. Sharma and J. KP, “Security Analysis of OAuth 2.0 Implementation,” *2023 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pp. 1-8, 2023.
- [8] C. D. Motero, J. R. B. Higuera, J. B. Higuera, J. A. S. Montalvo and N. G. Gómez, “On Attacking Kerberos Authentication Protocol in Windows Active Directory Services: A Practical Survey,” *IEEE Access*, vol. 9, pp. 109289-109319, 2021.
- [9] I. Indu, P. R. Anand and V. Bhaskar, “Identity and access management in cloud environment: Mechanisms and challenges,” *Engineering Science and Technology, an International Journal*, vol. 21, no. 4, pp. 574-588, 2018.

- [10] Y. Sadqi, Y. Belfaik and S. Safi, "Web OAuth-based SSO Systems Security," *In Proceedings of the 3rd International Conference on Networking, Information Systems & Security (NISS '20)*, vol. 69, pp. 1-7, 2020.
- [11] D. Pöhn and W. Hommel, "New Directions and Challenges within Identity and Access Management," *IEEE Communications Standards Magazine*, vol. 7, no. 2, pp. 84-90, 2023.
- [12] K. Patel, "A Survey on Vulnerability Assessment & Penetration Testing for Secure Communication," *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 320-325, 2019.
- [13] H. M. Z. A. Shebli and B. D. Beheshti, "A study on penetration testing process and tools," *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1-7, 2018.
- [14] IBM, "What is the Common Vulnerability Scoring System (CVSS)?," 21 February 2025. [Online]. Available: <https://www.ibm.com/think/topics/cvss>.
- [15] Okta, "Vulnerability Reporting Policy," n.d.. [Online]. Available: <https://www.okta.com/vulnerability-reporting-policy/>. [Accessed 2025].
- [16] Passport.js, "Passport-SAML," n.d.. [Online]. Available: <https://www.passportjs.org/packages/passport-saml/>. [Accessed 2025].
- [17] Technostacks, "Top NodeJS Libraries For Backend Development," 04 January 2024. [Online]. Available: <https://technostacks.com/blog/nodejs-libraries/>.
- [18] Okta, "2024 Gartner® Magic Quadrant™ for Access Management," Gartner, December 2024. [Online]. Available: <https://www.okta.com/resources/gartner-magic-quadrant-access-management/>.
- [19] Okta, "Okta Developer," n.d.. [Online]. Available: <https://developer.okta.com/signup/>. [Accessed 2025].
- [20] R. Bischofberger, E. Duss and T. Hort-Geiss, "SAML Raider - SAML2 Burp Extension," Compass Security, n.d.. [Online]. Available: <https://github.com/CompassSecurity/SAMLRaider>. [Accessed 2025].
- [21] B. Risher, "How to Hunt Bugs in SAML; a Methodology - Part II," 13 March 2019. [Online]. Available: <https://epi052.gitlab.io/notes-to-self/blog/2019-03-13-how-to-test-saml-a-methodology-part-two/>.

- [22] The OWASP Foundation, “A4:2017-XML External Entities (XXE),” 2017. [Online]. Available: [https://owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_\(XXE\)](https://owasp.org/www-project-top-ten/2017/A4_2017-XML_External_Entities_(XXE)).
- [23] M. Kubica, “node-xml2js,” n.d.. [Online]. Available: <https://github.com/Leonidas-from-XIV/node-xml2js>. [Accessed 2025].
- [24] Vaadata, “SAML: How it Works, Vulnerabilities and Common Attacks,” 19 July 2024. [Online]. Available: <https://www.vaadata.com/blog/saml-how-it-works-vulnerabilities-and-common-attacks/>.
- [25] Okta, “Handling Replay Attacks and Validating Timestamps in SAML Assertions with Okta,” 16 August 2024. [Online]. Available: https://support.okta.com/help/s/article/okta-service-has-protection-against-replay-attacks?language=en_US.
- [26] ngrok, “How does ngrok work?,” n.d.. [Online]. Available: <https://ngrok.com/docs/how-ngrok-works/>. [Accessed 2025].
- [27] OpenJS Foundation, “Production Best Practices: Security,” n.d.. [Online]. Available: <https://expressjs.com/en/advanced/best-practice-security.html>. [Accessed 2025].
- [28] J. Martinez, “Man-in-the-Middle (MITM) Attack: Definition, Examples & More,” StrongDM, 3 January 2025. [Online]. Available: <https://www.strongdm.com/blog/man-in-the-middle-attack>.
- [29] D. Bradbury, “Unauthorized Access to Okta's Support Case Management System: Root Cause and Remediation,” Okta, 3 November 2023. [Online]. Available: <https://sec.okta.com/articles/harfiles/>.
- [30] OffSec Services, “dsniff,” n.d.. [Online]. Available: <https://www.kali.org/tools/dsniff/#arpspoof>. [Accessed 2025].
- [31] OpenJS Foundation, “express-session,” n.d.. [Online]. Available: <https://expressjs.com/en/resources/middleware/session.html>. [Accessed 2025].
- [32] MDN, “Secure cookie configuration,” n.d.. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Practical_implementation_guides/Cookies. [Accessed 2025].

- [33] MDN, “Strict-Transport-Security,” n.d.. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Strict-Transport-Security>. [Accessed 2025].
- [34] Okta, “Application Integration Wizard SAML field reference,” n.d.. [Online]. Available: <https://help.okta.com/en-us/content/topics/apps/aiw-saml-reference.htm>. [Accessed 2025].
- [35] S. Brady, “Creating RSA Keys using OpenSSL,” 05 August 2020. [Online]. Available: <https://www.scottbrady.io/openssl/creating-rsa-keys-using-openssl>.
- [36] Okta, “About certificates,” n.d.. [Online]. Available: <https://help.okta.com/oie/en-us/content/topics/integrations/about-certificates.htm>. [Accessed 2025].
- [37] Cloudflare, “What is a DDoS attack?,” n.d.. [Online]. Available: <https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/>. [Accessed 2025].
- [38] C. Spillane, “pentmenu,” n.d.. [Online]. Available: <https://github.com/GinjaChris/pentmenu>. [Accessed 2025].
- [39] The OpenSSL Project Authors, “openssl-s_client,” n.d.. [Online]. Available: https://docs.openssl.org/master/man1/openssl-s_client/. [Accessed 2025].
- [40] The Open Information Security Foundation, “Suricata,” n.d.. [Online]. Available: <https://suricata.io/>. [Accessed 2025].
- [41] Open Information Security Foundation, “SSL/TLS Keywords,” n.d.. [Online]. Available: <https://docs.suricata.io/en/latest/rules/tls-keywords.html>. [Accessed 2025].
- [42] M. Driscoll, “The Illustrated TLS 1.3 Connection,” 2018. [Online]. Available: <https://tls13.xargs.org/#client-hello>.
- [43] DDOS-GUARD, “Popular Traffic Filtering Methods,” 23 November 2023. [Online]. Available: <https://ddos-guard.net/blog/traffic-filtering-methods>.
- [44] D. Crombie, “Optimising your Fail2Ban filters,” 29 September 2015. [Online]. Available: <https://www.the-art-of-web.com/system/fail2ban-filters/>.
- [45] “Suricata and fail2ban,” 23 March 2023. [Online]. Available: <https://blog.xentoo.info/2023/03/23/suricata-and-fail2ban/>.

- [46] K. S, “Cross Site Scripting (XSS),” The OWASP Foundation, n.d.. [Online]. Available: <https://owasp.org/www-community/attacks/xss/>. [Accessed 2025].
- [47] MDN, “Cross-site scripting (XSS),” n.d.. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/XSS>. [Accessed 2025].
- [48] OpenJS Foundation, “5.x API - express(),” n.d.. [Online]. Available: <https://expressjs.com/en/api.html#res.send>. [Accessed 2025].
- [49] J. Maury, “How to Use Input Sanitization to Prevent Web Attacks,” 6 February 2025. [Online]. Available: <https://www.esecurityplanet.com/endpoint/prevent-web-attacks-using-input-sanitization/>.
- [50] A. Adel, “Express XSS Sanitizer,” January 2025. [Online]. Available: <https://www.npmjs.com/package/express-xss-sanitizer>.
- [51] MDN, “Content Security Policy (CSP),” n.d.. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP>. [Accessed 2025].
- [52] K. S, “Cross Site Request Forgery (CSRF),” The OWASP Foundation, n.d.. [Online]. Available: <https://owasp.org/www-community/attacks/csrf>. [Accessed 2025].
- [53] MDN, “Cross-site request forgery (CSRF) prevention,” n.d.. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Practical_implementation_guides/CSRF_prevention. [Accessed 2025].
- [54] MDN, “Set-Cookie,” n.d.. [Online]. Available: <https://developer.mozilla.org/ru/docs/Web/HTTP/Reference/Headers/Set-Cookie>. [Accessed 2025].
- [55] krakenjs, “lusca,” n.d.. [Online]. Available: <https://github.com/krakenjs/lusca>. [Accessed 2025].
- [56] Okta, “okta-oidc-middleware,” n.d.. [Online]. Available: <https://github.com/okta/okta-oidc-middleware>. [Accessed 2025].
- [57] M. Paktiti, “Defending OAuth: Common attacks and how to prevent them,” 12 March 2025. [Online]. Available: <https://workos.com/blog/oauth-common-attacks-and-how-to-prevent-them>.

- [58] Okta, “Implement authorization by grant type,” n.d.. [Online]. Available: <https://developer.okta.com/docs/guides/implement-grant-type/authcode/main/#exchange-the-code-for-tokens>. [Accessed 2025].
- [59] Okta, “OAuth 2.0 and OpenID Connect overview,” n.d.. [Online]. Available: <https://developer.okta.com/docs/concepts/oauth-openid/#choosing-an-oauth-2-0-flow>. [Accessed 2025].
- [60] M. Collina, “autocannon,” n.d.. [Online]. Available: <https://github.com/mcollina/autocannon>. [Accessed 2025].
- [61] Open Information Security Foundation, “HTTP Keywords,” n.d.. [Online]. Available: <https://docs.suricata.io/en/latest/rules/http-keywords.html>. [Accessed 2025].
- [62] Okta, “Prevent Attacks and Redirect Users with OAuth 2.0 State Parameters,” n.d.. [Online]. Available: <https://auth0.com/docs/secure/attack-protection/state-parameters>. [Accessed 2025].
- [63] “CSRF protection for oauth authorization,” 2023. [Online]. Available: <https://security.stackexchange.com/questions/266526/csrf-protection-for-oauth-authorization>.
- [64] Passport.js, “passport-ldapauth,” 2023. [Online]. Available: <https://www.passportjs.org/packages/passport-ldapauth/>.
- [65] Okta, “Set up and manage the LDAP Interface,” n.d.. [Online]. Available: <https://help.okta.com/en-us/content/topics/directory/ldap-interface-main.htm>. [Accessed 2025].
- [66] Okta, “okta-pki,” n.d.. [Online]. Available: <https://github.com/okta/okta-pki>. [Accessed 2025].
- [67] Okta, “LDAP Interface connection settings,” n.d.. [Online]. Available: <https://help.okta.com/oie/en-us/content/topics/directory/ldap-interface-connection-settings.htm>. [Accessed 2025].
- [68] A. Dizdar, “Complete Guide to LDAP Injection: Types, Examples, and Prevention,” Bright Security, 2 June 2021. [Online]. Available: <https://www.brightsec.com/blog/ldap-injection/>.

- [69] T. Mick, J. Marneweck and V. Poikajärvi, “node-ldapauth-fork,” 2011. [Online]. Available: <https://github.com/vesse/node-ldapauth-fork/blob/master/lib/ldapauth.js#L60>.
- [70] K. Maina, “Ldap Pentesting,” 3 June 2022. [Online]. Available: <https://cyberkhalid.github.io/posts/ldap/>.
- [71] Okta, “Configure a global session policy and authentication policies,” n.d.. [Online]. Available: <https://developer.okta.com/docs/guides/configure-signon-policy/main/>. [Accessed 2025].
- [72] Cybersecurity & Infrastructure Security Agency, “Require Strong Passwords,” n.d.. [Online]. Available: <https://www.cisa.gov/secure-our-world/require-strong-passwords>. [Accessed 2025].
- [73] O. Duboust, “How long does it take a hacker to crack one of your passwords in 2024?,” 11 May 2024. [Online]. Available: <https://www.euronews.com/next/2024/05/11/how-long-does-it-take-a-hacker-to-crack-a-password-in-2024>.
- [74] Okta, “Rate limits overview,” n.d.. [Online]. Available: <https://developer.okta.com/docs/reference/rate-limits/>. [Accessed 2025].
- [75] D. Golubkov, “passport-saml-thesis,” 2025. [Online]. Available: <https://github.com/dangol9/passport-saml-thesis>.
- [76] D. Golubkov, “okta-oidc-thesis,” 2025. [Online]. Available: <https://github.com/dangol9/okta-oidc-thesis>.
- [77] D. Golubkov, “passport-ldap-thesis,” 2025. [Online]. Available: <https://github.com/dangol9/passport-ldap-thesis>.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis¹

I Daniel Golubkov,

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Vulnerability Assessment and Penetration Testing of Authentication and Authorization Protocols”, supervised by Edmund Laugasson:
 - 1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
 - 1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

18.05.2025

¹ The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.