A SAML 2.0 Authentication Middleware for ASP.NET Core

Jessy Kyalo Musyimi

Master in Information Security
Submission date: June 2018
Supervisor: Basel Katt, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology
Preface

This report represents the work done in partial fulfillment of my Master in Information Security, at the Norwegian University of Science and Technology (NTNU), department of Information Security and Communication Technology, in the spring semester of 2018.

The idea came when I was working for a Norwegian online pharmacy. Patient identity and prescription information was managed and protected by a SAML identity provider. The online pharmacy application had to be integrated with the SAML identity provider, in order to retrieve and show patient prescriptions on the online shop. To minimize the work required to integrate with a SAML identity provider, I searched for an open source solution to no avail, and ended up buying a SAML component that costed around 6000 USD.

The reader should be familiar with the SAML framework and programming. An understanding of the C# programming language, and the .NET Core framework would be helpful. An introduction to the SAML 2.0 standard and the ASP.NET Core framework is given.

01-06-2018
Acknowledgment

To my wife Jeddidah, thank you for your love and support. Thank you for all the nights, mornings and days I was away. Above all, thanks for taking care of our daughter Joan. Thank you for being my best friend. I owe you everything.

To my family, particularly my mother Serah, thank you for your support, prayers and words of encouragement during tough times. Thank you for your unwavering support and belief in me. I would never be the person I am today without you.

I owe great thanks to my supervisor Dr. Basel Katt, for the discussions, guidance and feedback. Thank you for being available to me whenever I needed you, not forgetting how quickly you replied my emails.

J.K.M
Abstract

The modern society is becoming more and more depended on information systems to run its critical services. Public infrastructure facilities, including the health services, commercial airlines and nuclear power plants depend on functional information systems to deliver secure and quality services to the society [1].

One way of building information systems is the use of web-based Internet applications. Web applications are software programs that run on a web server, and are accessed through a web browser [2]. They are accessible from any device or computer that is connected to the Internet. Considering the sensitivity and nature of personal information web applications store and give access to this days, they have to be built with security in mind. This includes, but not only limited to an effective authentication and authorization mechanism [3]. Effective authentication in web applications can be achieved using web application authentication protocols such as SAML and others [4].

Integrating a web application with a SAML identity provider is complex and time consuming for software developers [5] [6]. It requires a deep knowledge and understanding of XML, XML signatures and x509 certificates for encryption, decryption and signing of protocol messages [7] [8].

ASP.NET Core is the new framework developed by Microsoft for implementing web applications. At the moment, there are no free, open source SAML 2.0 libraries for ASP.NET Core. This thesis looks at how the SAML 2.0 authentication framework can be implemented in ASP.NET Core based web applications. It explores a way of making SAML 2.0 implementation friendly to software developers, by creating an open source, easy to configure, reusable, and flexible SAML 2.0 based authentication middleware for ASP.NET Core.
# Contents

Preface ................................................................. i
Acknowledgment ......................................................... iii
Abstract ................................................................. v
Contents ................................................................. vii
List of Figures ......................................................... xi
List of Tables ......................................................... xiii
List of Listings ......................................................... xv
Acronyms ................................................................. xvii
Glossary ................................................................. xix

1 Introduction ......................................................... 1
   1.1 SAML .......................................................... 1
   1.2 ASP.NET Core ............................................... 1
   1.3 ASP.NET Core middleware ........................................ 1
   1.4 Keywords .................................................... 1
   1.5 Problem description ........................................... 2
   1.6 Justification, motivation and benefits ......................... 2
   1.7 Research questions ........................................... 3
   1.8 Planned contributions ........................................ 3

2 Related work ....................................................... 5
   2.1 Background .................................................. 5
      2.1.1 .NET Framework .......................................... 5
      2.1.2 .NET Core and .NET standard ......................... 5
   2.2 ASP.NET Core ................................................ 6
      2.2.1 Porting a full .NET framework library (OIOSAML.NET) into .NET Core .................................................. 6
      2.2.2 Session management in ASP.NET Core web applications ........ 7
      2.2.3 Configuring ASP.NET Core applications ................ 9
      2.2.4 Diagnostics, tracing and logging ...................... 10
      2.2.5 ASP.NET Core middleware ................................ 11
      2.2.6 Handling authentication request-responses ............ 12
A SAML 2.0 Authentication Middleware for ASP.NET Core

2.3 SAML 2.0 .................................................. 15
   2.3.1 Web Single Sign-On (SSO) ......................... 15
   2.3.2 SAML 2.0 Components .............................. 16

3 Research methods .............................................. 27
   3.1 Exploratory case study as the research method ..... 27
   3.2 Literature review ....................................... 27
   3.3 Software development .................................. 28
      3.3.1 Methodology ...................................... 28
      3.3.2 Tools ............................................. 29

4 Saml2.Authentication.Core .................................... 31
   4.1 Requirements and specification ....................... 32
   4.2 Porting OIOSAML.NET library into ASP.NET Core .... 33
   4.3 Architecture .......................................... 34
   4.4 Solution structure ..................................... 35
   4.5 The SAML 2.0 authentication handler ............... 37
      4.5.1 Handling SAMLResponse .......................... 38
      4.5.2 Handling logout requests ......................... 38
      4.5.3 Handling authentication requests ............... 40
      4.5.4 Handling logout response ......................... 42
      4.5.5 Handling SAML authentication response (SAMLResponse) .... 44
      4.5.6 Handling sign-in (creating session cookie) .... 48
      4.5.7 Unique SAML request identifiers ............... 49
      4.5.8 Creating SAML 2.0 messages .................... 49
   4.6 Configuration ......................................... 50
      4.6.1 AuthenticationBuilder ............................ 50
      4.6.2 Startup services .................................. 51
      4.6.3 Bringing it all together .......................... 52
   4.7 Triggering authentication ............................... 55
   4.8 Triggering signout ..................................... 57
   4.9 Ethical and legal considerations ...................... 58

5 Results ...................................................... 59
   5.1 Configuration .......................................... 59
      5.1.1 Configuring salesforce as a SAML identity provider ... 59
      5.1.2 Configuring DemoWebApp as the service provider ... 60
      5.1.3 Managing identities ............................... 62
| H | IdentityProviderConfiguration | 129 |
| I | ServiceProviderConfiguration | 131 |
| J | SamlService | 133 |
| K | SamlProvider | 139 |
| L | Saml2MessageFactory | 143 |
| M | Saml2ClaimFactory | 147 |
| N | CertificateProvider | 149 |
| O | HttpArtifactBinding | 151 |
| P | HttpRedirectBinding | 153 |
| Q | Saml2StringBuilderExtensions | 161 |
| R | Saml2StringExtensions | 165 |
| S | Identity provider metadata | 169 |
| T | Base64 encoded SAMLResponse | 171 |
| U | Decoded SAMLResponse | 173 |
| V | Class diagram | 179 |
List of Figures

1. SAML 2.0 authentication middleware in the context of ASP.NET Core request pipeline .......................... 12
2. Basic SSO model .......................................................................................................................... 16
3. Structure of a SAML assertion ....................................................................................................... 17
4. Communication model using HTTP Redirect binding ................................................................. 20
5. Communication model using HTTP Artifact binding .......................................................... 22
6. SP initiated: Redirect $\rightarrow$ POST binding ........................................................................ 24
7. SP initiated: Redirect $\rightarrow$ Artifact binding .................................................................... 25
8. Architecture diagram .................................................................................................................... 34
9. Solution structure .......................................................................................................................... 36
10. Identity provider configuration in Salesforce ............................................................................. 60
11. Service provider configuration in Salesforce ............................................................................. 61
12. Users in salesforce ....................................................................................................................... 62
13. Authentication request URL ........................................................................................................ 64
14. User principal and claims ............................................................................................................. 70
15. LogoutRequest URL ..................................................................................................................... 72
16. LogoutResponse ........................................................................................................................... 73
List of Tables

1  Requirements and specifications ........................................... 32
2  Summary of .NET Core API portability analysis ......................... 33
3  Other threats associated with SAML-based systems .................. 76
List of Listings

1. Adding cookie middleware ........................................ 8
2. Extension methods to create (sign-in) and destroy (sign-out) session cookie .................................................. 9
3. Base AuthenticationHandler class ............................... 13
4. HandleRequestAsync method ...................................... 38
5. SignOutAsync method ................................................. 40
6. HandleChallengeAsync method .................................... 42
7. HandleSignOut method ................................................. 44
8. HandleSignin method .................................................. 45
9. HandleHttpArtifact method ......................................... 47
10. Signin method .......................................................... 48
11. CreateUniqueId method ................................................ 49
12. Configuring the authentication builder .......................... 50
13. Configuring start-up services ....................................... 51
15. Appsettings configuration example ............................... 54
16. Triggering authentication ............................................. 55
17. External authentication callback .................................... 56
18. Triggering signout ....................................................... 57
19. AuthnRequest ............................................................... 63
20. RelayState from authentication response ......................... 65
21. Validated assertion ....................................................... 65
22. LogoutRequest message .............................................. 71
23. Validated LogoutResponse message .............................. 74
24. Decrypting an encrypted assertion ............................... 77
25. Assertion signature check ............................................ 79
26. Replay attack check ..................................................... 80
Acronyms

**ACS** Assertion Consumer Service. 25, 26, 37, 41, 44, 60, 65, 75

**CLR** Common Language Run-time. 5

**FCL** .NET Framework Class Library. 5

**HTTP** Hypertext Transfer Protocol. 1, 7

**IDP** Identity Provider. 23, 24, 26, 32, 37, 40, 41, 52, 55

**JSON** JavaScript Object Notation. 9, 32, 53, 74

**LINQ** Language Integrated Query. 5

**SaaS** Software as a Service. 2

**SAML** Security Assertion Markup Language. iii, 1, 6, 15–18, 21, 23, 24, 27

**SAML 2.0** Security Assertion Markup Language version 2.0. iii, 3, 4, 15, 16, 27, 31, 32, 52, 53, 55, 59

**SLO** Single Logout. 32, 37, 75, 82

**SP** Service Provider. 23, 24, 26, 31, 32, 37, 52

**SSO** Single Sign-On. 5, 15, 16, 23, 24, 26, 81, 82

**URL** Uniform Resource Locator. 19

**USD** US dollars. 3

**XML** eXtensible Markup Language. iii, 2, 6
Glossary

**Assertion** A package of information about an identity issued by a SAML identity provider. 26

**Binding** Mapping from SAML request-response exchange, into the standard communication protocols like HTTP-GET, HTTP-POST and SOAP. 19

**Identity** Information about a person/subject. 62

**Identity Provider** An authority that can issues a SAML security assertion. 1, 15

**Profile** The combination of SAML protocols, Bindings and Assertions in order to satisfy a particular use case and enhance interoperability in applications. 23

**RelayState** A mechanism, whereby state data can be round tripped to an identity provider and back to the service provider or vice versa, as a parameter in the request-response exchange. 31, 38, 76

**SAML Artifact** A reference used for transferring authentication messages between a service provider and an identity provider. 21

**SAML Protocol** Defines rules that govern how an identity provider and a service provider, will handle different SAML requests and responses. 17

**SAMLart** A URL parameter for transferring a SAML artifact value between a service provider and an identity provider. 21, 37, 46

**SAMLRequest** An authentication request (samlp:AuthnRequest) or a logout request (samlp:LogoutRequest) generated by either a service provider or an identity provider. 2, 64, 77

**SAMLResponse** A response to an authentication (samlp:Response) or logout request (samlp:LogoutResponse). 2, 14, 65, 78
**Service Provider** Any entity that is in a position to accept a SAML security assertion, and use it for access control to provided services. 1, 15
1 Introduction

1.1 SAML

Security Assertion Markup Language is an open standard for exchanging authentication and authorization information between a Service Provider and an Identity Provider. The authentication information is exchanged in form of a security assertion, regarding a given identity, also known as a subject and typically a person. A service provider is any entity that is in a position to accept a security assertion, and use it for access control to provided services. An identity provider is the authority that issues the security assertion [7] [9].

1.2 ASP.NET Core

ASP.NET Core is the new framework created by Microsoft for building web applications. It is cross-platform and it runs on top of Microsoft .NET run-time, which can be installed on Windows, Mac or linux systems. This means that web applications created in ASP.NET Core can be hosted on either Windows or Linux web servers [10].

1.3 ASP.NET Core middleware

An ASP.NET Core middleware, is a software that can be plugged into a web application’s HTTP pipeline, to process HTTP requests and responses. It is set up when the web application starts, has access to all the incoming HTTP requests, can perform work on them and choose to pass them to the next component/middleware in the pipeline. ASP.NET Core middleware is suited for web application’s cross cutting concerns such as logging or authentication [11]. This project will utilize the ASP.NET Core middleware functionality, to create a SAML authentication handler for ASP.NET Core.

1.4 Keywords

Authentication, authorization, access control, web application security
1.5 Problem description

To software developers, SAML is hard to work with [5] [6]. It requires deep knowledge and understanding of XML, XML signatures and x509 certificates for encryption, decryption and signing of protocol messages. This makes integrating a web application with a SAML identity provider, complex and time consuming, especially if a developer has to write all from scratch [7] [12] [13].

Some identity providers have created SAML toolkits, which are tied to their solutions and therefore lacking reusability and flexibility. Other third-party companies have created proprietary SAML libraries that are quite expensive to use [14]. Different full .NET framework open source projects have tried to solve this by creating SAML libraries, that support only a subset of the SAML supported bindings, basically the HTTP redirect binding. This is due to lack of funding and support among other issues. Currently, there are no open source SAML libraries supporting ASP.NET Core or the HTTP Artifact binding [12] [7].

Clearly, some work needs to be done in order to bring SAML support into ASP.NET Core. The Danish Government, through Digitaliserdk, created an open source SAML library for the full .NET framework. The library is called OIOSAML.Net and it implements the SAML standard, in the context of the full .NET framework. The library does not support ASP.NET Core [15].

Is it therefore possible to fork this repository, port it into ASP.NET Core and, further build a full SAML authentication middleware for ASP.NET Core, that is reusable, flexible and easy to configure by software developers? One of the main challenges here will be initiating SAMLRequests, from the service provider, via the middleware and forwarding them to an identity provider. Another major challenge will be receiving and handling SAMLResponses from an identity provider, through the middleware and forwarding the result to the service provider application.

1.6 Justification, motivation and benefits

Despite the recent growth of new and developer friendly authentication standards like the OpenID Connect [8], SAML is still widely used for enterprise identity management. According to a survey by onelogin conducted in 2014, 67% of Software as a Service (SaaS) providers used SAML for SSO, while 19% were planning to implement SAML based services within the next 12 months [16].

SAML is relatively complex and time consuming for developers to implement.
A simple search in stackoverflow for SAML related questions gave over 9000 questions, of which a good number had not been answered [5].

ComponentSpace recently released a SAML 2.0 software component for ASP.NET Core. The cost for an enterprise licence with source code is barely 6000 USD with an additional 1099 USD for a subscription license [14]. Clearly, this is a lot of money for small and medium-sized enterprises who would like to use SAML as an authentication protocol in their applications.

When implementing SAML based authentication solutions, many developers are not willing to spend the time required to understand SAML or become experts. What they would like is a simple, free (open source), easy to adopt software package, which can be with a few lines of code configured to integrate with any SAML based identity provider [6] [8] [13]. With our solution, the software developer is free to focus their time and resources into developing features that deliver direct value to their organization.

1.7 Research questions

On the basis of the problem description given above, the following research questions are defined:

1. How to initiate and handle authentication requests?
2. How to receive and handle authentication responses?
3. How can the middleware, securely manage user sessions?
4. How to initiate and handle logout requests?
5. How to receive and handle logout responses?
6. How should the middleware be designed to offer, a reusable and flexible configuration, for any identity provider specific, or internal service provider runtime variables?
7. How to monitor, troubleshoot and diagnostically trace events within the middleware?

1.8 Planned contributions

The main contribution of this research is the created SAML 2.0 authentication middleware for ASP.NET Core (Saml2.Authentication.Core). This is available as a software package in github (https://github.com/jkmu/Saml2.Authentication.Core). The software is:

1. Open source under the Mozilla Public license [17]
2. Developer friendly (easy to adopt, configurable with a few lines of code)
3. Flexible and reusable (works to integrate with any SAML 2.0 identity provider using the HTTP Redirect binding and the HTTP Artifact binding)
4. Further work can be done to;
   - Support other SAML 2.0 bindings and profiles.
   - Enhance performance and usability.
   - Support multiple SAML 2.0 identity providers
2 Related work

This chapter discusses related work and introduces the .NET and ASP.NET Core frameworks for web applications development. It explores authentication in ASP.NET Core, while introducing background materials on creating custom authentication handlers. The chapter ends with a review of Single Sign-On (SSO) in the context of the SAML 2.0 framework and its components.

2.1 Background

2.1.1 .NET Framework

The .NET framework is a software development platform developed by Microsoft. It runs only on Windows machines, providing tools and technologies needed for developing Windows, Windows phone, Windows Server, Azure and web applications. The framework provides the compile time and run-time (execution environment for a managed program) necessary for building applications in any .NET based programming language [18]. Some of the main features of the .NET run-time are; automatic memory management, type safety, delegates and lambdas, generic types, Language Integrated Query (LINQ) and async programming [19]. The .NET framework has two main components; the Common Language Run-time (CLR) and .NET Framework Class Library (FCL). The CLR provides a run-time environment and services required for running programs, while the FCL contains a library of classes, interfaces and types that gives a developer access to the underlying system functionality [20].

2.1.2 .NET Core and .NET standard

.NET Core is a cross-platform implementation (port) of the .NET framework. Applications build in .NET Core can run on Windows, macOS, Linux machines including embedded and IOT devices. It is completely open source under the MIT licence. .NET Core run-time is flexible in the sense that it can be deployed together and inside the application, or it can be installed on the machine running the application. It is build on the same basis as the .NET Framework CLR [21].

.NET Standard is an interface specification, describing which features and
APIs are available in .NET implementations. Its purpose is to maintain uniformity and a standard in the .NET Base classes, giving developers the possibility of building software libraries that are flexible and reusable in all .NET implementations. Different versions of the .NET standard show how many APIs are available, with the latest version being .NET standard 2.0. Every .NET implementation (.NET/.NET Core) must advertise which .NET Standard it supports. [22].

Both the .NET framework and .NET Core support C-Sharp, F-Sharp and Visual Basic programming languages [22].

2.2 ASP.NET Core

As mentioned earlier, ASP.NET Core is a cross-platform and open source framework for building web applications, created and maintained by Microsoft. ASP.NET Core is made to run on the .NET Core, .NET or the mono run-time. It originated from ASP.NET which is a framework for building web applications that run on the full .NET framework. ASP.NET Core based applications are capable of running on Windows, Mac and Linux platforms [18] [10].

2.2.1 Porting a full .NET framework library (OIOSAML.NET) into .NET Core

As described in the problem description of this plan, SAML framework is XML based [12]. All protocol messages such as assertions, protocols, bindings and profiles are defined using a XML schema [23] [24]. Their integrity and confidentiality is guaranteed using XML signatures and XML encryption [25].

The full .NET framework uses the following APIs to create and verify XML digital signatures and XML encryption [15].

- System.Xml
- System.Security.Cryptography

These APIs have been ported over to .NET Core 2.0 with the System.Security.Cryptography.Xml being the latest and the last API to be ported. This means that as of now, .NET Core supports signing of XML documents [26] [27] [28].

Microsoft recommends the following 5 steps when porting existing .NET frame-
work libraries into .NET Core [29]

1. Identify all the library dependencies and assert if they are also supported in .NET Core
2. Update the target framework of the library to .NET Framework 4.6.2
3. With the .NET Portability Analyzer tool, analyze the library assembly [30]. The generated report can help find out which APIs the library is using that are not or are supported by .NET Core.
4. Port the library tests first
5. Port the library

2.2.2 Session management in ASP.NET Core web applications

Web applications use the HTTP protocol to transfer web pages from a web server to a client through a web browser. The web browser opens a connection to the web server, makes a request, and waits for a response. Web servers do not store any state between consecutive requests, making the protocol stateless [31]. Since there is no storage between subsequent requests, there is no way of remembering what the web server transferred to a client in the past, or what activities the user carried out in the previous request [32].

The intention of session management in web applications is to create an association between an authenticated user and a session [32]. Sessions represent long lasting authenticated requests and responses, between the same user (browser), and a web server, starting when the user signs in to the web server, until when the user explicitly signs-out, or the session expires [32]. Different user related variables (e.g. user-is-authenticated, username etc) can be added to the session, and these will be remembered by the web server as long as the session is valid.

There are multiple ways of implementing session management in web applications (HTTP). Some of these include the use of HTTP cookies, URL parameters and hidden form fields [33]. This project focuses on using HTTP cookies as the main method of keeping user sessions within the SAML authentication middleware and the service provider’s application.

HTTP cookies are pieces of text data that the web server sends to the client (browser) as part of the response. These are meant for the browser to store them on disk and send them as part of the next request to the web server. In this way, the web server can tell which HTTP requests belong to the same user (session). The web server will then read the contents of the cookie and populate the request
A SAML 2.0 Authentication Middleware for ASP.NET Core

Microsoft offers the cookie authentication middleware as a way of managing user sessions in ASP.NET Core web applications. This is implemented as part of the Microsoft.AspNetCore.Authentication.Cookies package. The middleware is initiated during the start-up of a web application. The following is an example code from Microsoft showing how to initiate the cookie middleware.

```csharp
services.AddAuthentication("Cookies").AddCookie();
```

Listing 1: Adding cookie middleware

ASP.NET Core 2.0 allows the use of multiple cookie instances, with each instance creating a new authentication handler, having the capability of authenticating the same user on different levels, and also using different identity providers. The above code will initiate the cookie middleware with a named cookie authentication handler (authentication scheme) called "Cookies". The handler will be responsible for storing the outcome of the authentication in a cookie called "Cookies". Signing in a user and creating the session cookie is accomplished by calling the cookie middleware extension method `SignInAsync` and passing the authentication scheme name e.g. "Cookies" from the above example and a `ClaimsPrincipal` (user data/claims/variables). The cookie is encrypted and attached to the current response. To sign out a user, the extension method `SignOutAsync` is called with the authentication scheme name as a parameter.
2.2.3 Configuring ASP.NET Core applications

In the full .NET framework, ASP.NET web applications use XML based configuration files. An XML configuration file called the **web.config** is placed at the root of the application. The **web.config** file has a section called **AppSettings** where application configuration variables can be added as key value pairs [37].

ASP.NET Core web applications, do not utilize **web.config** files for application configuration, but can read application configuration values from multiple external sources [37]. Each external source is implemented as a configuration provider for a given file format or source.

The following configuration sources are supported in ASP.NET Core [38].

1. File formats (**JSON**, **XML** and **INI**). This is the most commonly used configuration provider and it uses external json/xml/ini files placed in the application root.
2. Command-line arguments
3. Environment variables
4. In-Memory .NET objects
5. An encrypted user store
6. Azure key vault
7. Custom created or installed providers

ASP.NET Core’s in-build configuration API is used to read stored configuration values from the configuration files as key value pairs [37].
2.2.4 Diagnostics, tracing and logging

When troubleshooting or debugging applications, tracing makes a fundamental technique as it provides developers with an insight on what is happening under the covers of their application. Tracing records relevant events during application run-time and keeps them for analysis [39]. This is helpful when testing or troubleshooting a piece of code as it can identify system errors, pinpoint exceptions, warnings and code behaviour. [40]. Combined with logging, information collected when tracing can also be stored for later offline analysis [39].

Chiarreta and Lattanzi identifies logging as an important part of a web application which is difficult to implement. This is demonstrated by the fact that there are more than 1,900 different logging related frameworks available in the .NET package manager nuget [37].

Modern web applications utilize software packages from different vendors and authors. These can use different logging frameworks, making logging in applications complicated. Microsoft in the full .NET framework implemented the Common.Logging library as a means of solving this problem, by providing an abstraction and enabling switching between different logging frameworks e.g. log4net, NLog, Serilog [41] [37].

With ASP.NET Core, the Common.Logging library is not required anymore. The framework offers the same behaviour out of the box [37]. This is implemented as a built in logging API, supporting different logging providers (frameworks) and offering the possibility of adding custom third party logging frameworks. Logging providers can be configured to show a log message, e.g a console logger or store message in a file or database [42].

To create logs, the ILogger interface is injected into a class constructor. With the help of the inbuilt dependency injection in ASP.NET Core, this will create an ILogger object of a given category name. The category name in this context will be the class name. The ILogger interface expose different logging methods, depending on the required log level. A log level defines the degree of importance of a given log message [42]. The different log levels supported by ASP.NET Core are [42].

1. Trace. The lowest level indented for log messages useful only to developers. Should never enabled in production as they may contain sensitive information.
2. Debug. Intended for log messages that are useful during development and
debugging sessions

3. **Information.** For general log messages useful for tracing flow in the application

4. **Warning.** Warning log messages are used for indicating any error of which application did not stop but automatically recovered. These point to actions that needs more investigation.

5. **Error.** The error log level is indented for log massages indicating failure during a given operation.

6. **Critical.** Service level log messages that lead to service shutdown or prevent the service from starting

### 2.2.5 ASP.NET Core middleware

To create the SAML authentication middleware, it is important to understand the role ASP.NET Core middleware play in handling requests and responses in the application pipeline.

A middleware is a reusable class or a method that uses a request delegate to build the request pipeline. The Run, Map and Use methods are used to configure request delegates. When the application receives an incoming HTTP request, the request is passed through each middleware, (request delegate) in the pipeline. The main function of each middleware is to execute a given function or block of code on the request and determine whether to allow the the request to enter the next middleware in the pipeline or not. This is called short circuiting [11].
The figure below shows the SAML 2.0 authentication middleware placement in the context of other middlewares in an ASP.NET Core request-response pipeline [11].

![Figure 1: SAML 2.0 authentication middleware in the context of ASP.NET Core request pipeline](image)

### 2.2.6 Handling authentication request-responses

In ASP.NET Core 2.0, authentication middlewares are implemented as authentication handlers. The `AuthenticationHandler<TOptions>` class in the `Microsoft.AspNetCore.Authentication` name space is the base class under which new custom authentication handlers can be derived. The class contains the constructors, properties and methods necessary for processing each HTTP request in the HTTP pipeline [43].

The class is abstract, takes in one parameter `Options`, which represents the base Options model for all authentication middlewares. The `AuthenticationOptions` class is used to pass authentication middleware configuration values, into the authentication handler. These can be for example, the `AuthenticationScheme` property, which sets the logical name of the authentication middleware. This enables the possibility of using the same authentication middleware, more than once in the HTTP pipeline, with different authentication scheme names [44] [43].

The base authentication handler class offers different methods which should be implemented by any custom authentication handler. Some of these methods are discussed below.
public abstract class AuthenticationHandler<TOptions>
    : Microsoft.AspNetCore.Authentication.IAuthenticationHandler
        where TOptions
        : AuthenticationSchemeOptionsnew()

Listing 3: Base AuthenticationHandler class

HandleChallengeAsync()
This method should be implemented, to deal with unauthorized requests, and
user login challenge. It should contain logic that performs some kind of authen-
tication checks, and returns the appropriate response, either by modifying the
HTTP response headers, redirecting to a login page, or creating an authen-
tication request and redirecting to an external identity provider for authentication as
implemented in this project [45].

The method takes in one parameter of type AuthenticationProperties. This
is used to temporarily store state values, during an authentication session. The
state values should include a redirect URL. This denotes the endpoint where the
authentication handler will transfer control to after a successful authentication
challenge. State values are not limited to redirect URLs, but any data that the
consumer of the middleware needs to be restored, after a successful or failed
authentication attempt [46].

IAuthenticationRequestHandler interface
The IAuthenticationRequestHandler interface contains one method

public System.Threading.Tasks.Task<bool> HandleRequestAsync();

which is called on every request coming in through the HTTP pipeline. It re-
turns true if HTTP request processing should stop, or false if the HTTP request
should be passed on to the next middleware in the HTTP pipeline [47]. This
is useful especially for authentication handlers that redirect to external identity
providers for authentication, as they will need a way of receiving the authenti-
cation response, handling it and reporting to the HTTP pipeline that the request
has been handled.

One way of doing this, is by listening to HTTP requests to a particular applica-
tion path or endpoint. If the current HTTP request path equals to a pre-configured
value, then the authentication handler should handle the request, otherwise the handler should return false which forwards the HTTP request into the next middleware.

The method `HandleRequestAsync` should be implemented to handle different requests-response messages, based on different pre-configured application paths; for example handling `SAMLResponses` through pre-configured Assertion Consumer Service or Single Logout URLs [47].
2.3 SAML 2.0

SAML 2.0 is a general framework for transferring identity and security information about a user, also known as a subject, between a service provider and a SAML 2.0 based identity provider. [7]. Oasis, describes three major uses of SAML; Web Single Sign-On (SSO), Attribute-Based Authorization and Securing Web Services. This thesis looks at SAML 2.0 in the context of providing Web SSO.

2.3.1 Web Single Sign-On (SSO)

In access control, single-sign-on is the reuse of an existing session by an identity provider, for a given user. The session is usually set when the user tries to access resources, on any service provider, protected by the same identity provider. If a user tries to access resources on another service provider, protected by the same identity provider, and on the same browser, the identity provider will redirect back to the service provider, with an assertion from the existing session, as long as it is valid. The user will not have to re-supply their login credentials [7] [9].

Identity federation is the sharing of a user’s identity information between multiple identity providers. The same user identity information is used to identify, and give a user access to resources on different systems (service providers). The identity providers have to agree on which subset of identity information or attributes to federate [7] [9].

As described earlier, web SSO makes it possible for a user to authenticate on website A, using an Identity Provider A, and without an additional authentication, the user is allowed to access protected resources at website B, that uses identity provider A. A website here refers to a Service Provider.

SAML 2.0 web SSO, works by allowing the communication of the initial authentication assertion, that was issued to the user after authenticating in website A, to website B. If website B, trusts the origin of the assertion, website B, can then choose to sign in the user as if they were directly authenticated [7].
The figure below shows the basic web SSO model [7]. The identity provider authenticates users, which are subsequently recognized and allowed to access restricted resources at the service provider [7].

2.3.2 SAML 2.0 Components

The SAML 2.0 framework is defined by different components which are, assertions, protocols, bindings and profiles [48].

Assertions

SAML assertions represent packages of information about an identity, issued by an identity provider and transferred to the service provider as a response to an authentication request. Three types of assertion statements are defined in the framework; authentication, attribute and authorization decision. The three types can be created by any SAML identity provider. [48].

The authentication assertion statement, is generated by an identity provider, to show that a given subject was authenticated using a particular method at a given time. The attribute assertion statement shows that the given attributes are associated with the specified authenticated subject, while authorization decision statements show, whether requests to access a given resource at the service provider has been granted or not [7].
Structure

SAML assertions are XMLDocuments, and their structure is generic. Inside the XMLDocument, different XMLElements describe the authentication, attribute, authorization decision or any other custom defined statements.

The figure below from OASIS represents the high level structure of a SAML authentication assertion [7]

![Diagram of SAML assertion structure]

Figure 3: Structure of a SAML assertion

Protocols

SAML protocols define how an identity provider and a service provider, will handle different SAML requests and responses. Service providers send authentication (login) and logout requests to the identity provider and expect responses for the same in return [23].

Other SAML requests, that a service provider can send to an identity provider include; requests to authenticate a given principal and return an assertion, requests that a name identifier be registered, requests to terminate the use of a given identifier, requests to retrieve SAML Protocol messages by means of an artifact, requests to destroy related sessions, and requests for a name identifier mapping [7].

The following protocols are supported by the SAML standard; Assertion Query and Request Protocol, Name Identifier Management Protocol, Authentication Request Protocol, Artifact Protocol, Name Identifier Mapping Protocol and Single
Logout Protocol [23]. This project focuses on implementing the Authentication Request Protocol, Artifact Protocol and the Single Logout Protocol in the authentication middleware. Further work to support other protocols is outside the scope. The protocols are elaborated below, but their concrete usage and examples are given in the results chapter.

**Authentication Request Protocol**

The Authentication Request Protocol defines an authentication request <Authn-Request>, which is issued by a service provider to an identity provider, requesting for authentication of a given subject. The identity provider responds with a <Response>, containing one or more assertions of the authenticated subject [23].

**Artifact Protocol**

The Artifact Protocol provides a way of transferring authentication assertions by reference, also referred to as an artifact. During authentication, the identity provider creates the assertion, and assigns a reference/artifact to it. The SAML response to the service provider contains the artifact, which can be used to obtain the actual assertion through a back channel. [23].

**Single Logout Protocol**

The Single logout Protocol defines a request, issued by a service provider to an identity provider, allowing the destruction of all sessions associated with a given subject. The logout request can be initiated by either a service provider or an identity provider [23].

**Bindings**

SAML protocol Bindings, represent mappings from SAML request-response exchanges, into the standard communication protocols like HTTP-GET, HTTP-POST and SOAP. These define how SAML protocol messages can be communicated within HTTP-GET (redirect), HTTP-POST or SOAP messages. The aim of SAML bindings is to ensure that implemented SAML software can operate using standard messaging and communication protocols.

Some bindings define a **RelayState** mechanism, whereby state data can be round tripped to the identity provider and back as a parameter in the request-response exchange. This is necessary to preserve the application state before authentication, and to restore the state after authentication [7].

The standard defines the following bindings; SOAP Binding, HTTP-Redirect, HTTP-POST, HTTP-Artifact and URI Binding. This project looks at the HTTP-
Redirect and HTTP-Artifact bindings as implemented in the authentication middleware [7].

**HTTP Redirect Binding (urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect)**

The HTTP-Redirect Binding transmits SAML protocol messages, within browser URL parameters. SAML or the HTTP Protocol does not enforce a limitation on the resulting request URL length, but standard web browsers do. For example, the Microsoft Internet Explorer browser has a limit of 2,083 characters. According to Boutell, long URLs over 2,000 characters are discouraged as they will not work in all browsers [7] [49].

To carry SAML XML messages on a URL, special encoding have to be applied on the message. More complex messages should be transmitted using the HTTP POST or HTTP Artifact bindings [7].

State data can be transmitted within the protocol message as a RelayState parameter in the HTTP redirect binding. SAML enforces a maximum size of 80 bytes, and requires that the integrity of the state be protected by the creator of the state. The creator is either a service provider or an identity provider. The standard requires that, any SAML message that has RelayState within, the responder must return the same RelayState as received in the corresponding response using a binding that supports RelayState [7].

SAML protocol messages withing the HTTP Redirect binding are encoded using URL encoding techniques and transmitted using the HTTP GET method. The DEFLATE compression encoding is one of the URL encoding techniques used. This is identified as urn:oasis:names:tc:SAML:2.0:bindings:URL-Encoding:DEFLATE [7].

The following steps builds a signed authentication request URL using the DEFLATE encoding technique [7].

1. The XML request (without signature) is deflate encoded, then URL encoded and added to a parameter SAMLRequest.
2. The RelayState, if available is DEFLATE encoded and URL encoded and added as a parameter.
3. The signing algorithm is URL encoded and added as parameter.
4. The signature of the result is calculated using the correct hashing algorithm.
5. The signature is converted into a base 64 string, URL encoded, and added as a parameter.

The resulting URL would look like {destination}?SAMLrequest=request&
**RelayState=relaystate&SigAlg=alg&Signature=signature**

The diagram below from OASIS models the communication between a service and identity provider, using the HTTP Redirect binding. The user agent represents the user’s browser, while the SAML Requester or SAML responder can either be a service or an identity provider [7].

![Communication model using HTTP Redirect binding](image)

Figure 4: Communication model using HTTP Redirect binding
HTTP Artifact binding (urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact)

The HTTP Artifact binding represents a way of transmitting the SAML request, the SAML response, or both by a reference; also known as an artifact. The SAML requester creates the actual SAML message, assigns it a reference and stores it locally. The SAML requester then transmits the artifact to the SAML responder through either HTTP Redirect or HTTP POST binding. On receiving the artifact, the responder can then retrieve the actual message by the artifact using a back channel binding, such as the HTTP SOAP binding. The HTTP Artifact binding therefore uses multiple bindings to transmit request and response messages [7].

The authentication middleware created in this project supports transmitting the SAML request, using the HTTP Redirect binding, and receiving the SAML Response via HTTP Redirect or the HTTP Artifact binding.

The HTTP Artifact binding is recommended for cases where the service and identity provider must communicate via a HTTP user agent (browser), but not transmit the whole message through the front channel. This can be due to technical or security reasons. To resolve the actual message, an `<samlp:ArtifactResolve>` request is sent to the requester through a pre-configured communication path, basically an endpoint [7].

The binding supports transmitting of a RelayState together with the SAML Artifact, in the same manner as the HTTP Redirect binding. When transmitting using the HTTP Redirect binding, the artifact value is URL encoded, and added into a parameter known as SAMLart. If RelayState is present, it is URL encoded and added to the query, in a parameter called RelayState. If transmitting using the HTTP POST Binding, the SAMLart is form-encoded and placed in a form named SAMLart with an additional hidden field RelayState. The action of the form is set to the recipient’s assertion consumer service URL, with the method set to POST [7].
The following diagram from OASIS shows the system model for request-response SAML communication using the HTTP Artifact binding [7].

![Communication model using HTTP Artifact binding](image)

Figure 5: Communication model using HTTP Artifact binding
Profiles

SAML profiles specify how SAML components are combined to work together in a particular application enhancing interoperability [50]. Described below are some of the SAML profiles that are implemented in this project.

Web Browser SSO Profile

The Web Browser SSO Profile combines the Authentication Request Protocol with the HTTP Redirect, HTTP POST and the HTTP Artifact bindings to support Single Sign-On in web browsers [50].

The Profile defines four types of communication models grouped into two. These are the push or pull defined by how SAML assertions, are delivered to the Service Provider, while the Idp or SP initiated models are defined by how the communication flow is initiated. The pull model involves using the Artifact binding to send a message by a reference an allowing the receiver to 'pull' the actual message related to the reference. The push model is used to deliver SAML messages through HTTP redirects or HTTP POST [7].

IDP and SP initiated communication flows can be combined with the different SAML bindings to give six different use cases. Two of the actual use cases for this project are discussed below [7].
**SP initiated: Redirect → POST binding** This use case assumes an unauthenticated user trying to browse restricted resources in service provider www.abc.com which has its identities provided by identity provider www.xyz.com. The website www.abc.com will therefore create an `<AuthnRequest>` message and deliver it to www.xyz.com through HTTP Redirect while the identity provider in response will prepare and deliver a **SAML** response through the HTTP POST binding [7].

The figure below from OASIS illustrates the communication [7].

![Diagram](image.png)

**Figure 6: SP initiated: Redirect → POST binding**

Description of the process [7].
1. Anonymous user tries to access a restricted resource
2. **SP** creates and sends an `<AuthnRequest>` message and redirects to **IDP**
3. The **SSO** service decides if user has to login and challenges the user for credentials
4. User provides valid credentials
5. The **SSO** service POSTS a **SAML** response back to the browser
6. The browser issues an HTTP POST containing the SAML Response to the
Assertion Consumer Service (ACS)

7. ACS validates the digital signature on the SAML Response and issues an HTTP Redirect to the browser allowing the user to browse the protected resource

**SP initiated: Redirect → Artifact binding** This use case assumes an unauthenticated user trying to browse a restricted resource in service provider www.abc.com which has its identities managed by identity provider www.xyz.com. The service provider www.abc.com will therefore create an `<AuthnRequest>` message and deliver it to www.xyz.com through the HTTP Redirect binding. The identity provider www.xyz.com will in turn respond with a SAML Artifact through an HTTP POST message. The service provider can use the SAML Artifact to resolve the actual SAML response from the identity provider [7].

The figure below from OASIS illustrates this use case [7].

![Figure 7: SP initiated: Redirect → Artifact binding](image)

Description of the process [7].

1. Anonymous user tries to access a restricted resource.
2. **SP** creates and sends an `<AuthnRequest>` message and redirects to **IDP**.

3. The **SSO** service decides if user has to login and challenges the user for credentials.

4. User provides valid credentials.

5. The **SSO** service generates an assertion and an artifact for the user, and sends the artifact to the browser.

6. Browser POSTS the SAMLart to the **ACS** and extracts the source ID from the artifact to find the identity of the SAML responder (www.xyz.com).


8. www.xyz responds with an `<ArtifactResponse>` message containing the previously generated **Assertion** and establishes a session for the user.

9. The **ACS** redirects the browser to the protected resource with session cookies for access control.

**Single Logout Profile**

The Single Logout Profile defines how the Single Logout protocol is combined with other bindings (SOAP, HTTP Redirect, HTTP POST and HTTP Artifact) to support the destruction of SAML sessions for a particular subject [50].

**Artifact Resolution Profile**

The Artifact resolution Profile specifies the combination of the Artifact Resolution Protocol and other bindings like the SOAP binding [50].
3 Research methods

This chapter discusses the research methods used in the thesis.

3.1 Exploratory case study as the research method

To be able to answer the above research questions, an exploratory case study was used as the main research methodology. This was combined with the waterfall model for software development. According to Yin, case studies are the preferred research method strategy when "how" research questions are formulated as it is in this project [51].

Case study is a robust method of investigation that allows researchers to explore, and understand complex issues. It gives the ability to combine both qualitative and quantitative data, explaining the process and outcome of an experiment [52]. This thesis aims at investigating and understanding how to build a SAML 2.0 authentication middleware for the ASP.NET Core web application development framework. The projected is scoped to implement the HTTP Redirect and HTTP Artifact bindings of the SAML framework.


To validate the results, a demo web application that referenced the authentication middleware software package (Saml2.Authentication.Core) as a dependency was created. The demo web application was then configured using the authentication middleware, to use a real world SAML identity provider salesforce.com (https://www.salesforce.com), for authentication and managing identities.

3.2 Literature review

The study focused on related work in regard to implementation of SAML authentication in ASP.NET Core web applications. It was of importance to understand how SAML 2.0 authentication works, and how SAML 2.0 can be implemented into a middleware useful in ASP.NET Core web applications development. OASIS [53], which is the body behind SAML 2.0 has released a lot of literature and
technical documentation for SAML 2.0. This literature was studied, combined
with the OIOSAML.NET library [15] in order to understand; the inner workings
of the SAML 2.0 framework, the .NET framework classes that implement the dif-
ferent aspects of the framework, and the corresponding classes in the .NET Core
framework for easier porting.

The Microsoft documentation for ASP.NET Core was also studied in order
to answer the other research questions, namely; session management, config-
uration of identity provider's information and tracing for troubleshooting. The
stack overflow forum was utilized for searching of examples and asking soft-
ware development related questions, combined with standard search engines
such as www.google.com and others. Other search engines that are more related
to science like the www.scholar.google.com, Springerlink (link.springer.com),
Science Direct (www.sciencedirect.com) and IEEPlore (http://ieeexplore.
ieee.org) were also used.

Since there was limited literature on creating authentication middlewares/han-
dlers for ASP.NET Core, as it is not a task software developers do often, the Mi-
tree/dev/src) was studied. This gave different examples of authentication han-
dlers created by Microsoft for Facebook, Google authentication etc.

3.3 Software development

3.3.1 Methodology

When developing software, the choice of a good software development method-
ology is important for success. A good software development methodology forces
discipline on the software developer, helping them to increase the overall under-
standing of the problem, which will in turn improves the quality of the product
[54]

To develop the middleware, the waterfall method of software development
life-cycle was used. This is one of the oldest methods of software development,
proposed by Winston W. Royce in the 1970 [55]. The model is comprised of
several phases, one after the other and in a sequential manner, with the output
from one phase being the input into the next phase. The phases are analysis,
design, implementation, testing and maintenance [55].

The waterfall model was suitable for this project because, it is simple, easy to
understand, the project is short, and the requirements are clear. Since only one
person was working on the project, it was easy to set milestones while completing tasks from one stage/phase to the next.

3.3.2 Tools
This section shortly describes the software development tools used to create the SAML authentication middleware.

Programming language
C#
C# is an object oriented language for building different application types for the .NET framework. The language is type-safe with encapsulation, inheritance and polymorphism concepts built in. Its syntax is similar to C, C++ or Java programming languages making it easy for developers to switch to any of these languages with ease [56].

Integrated Development Environment (IDE)
An IDE is an application that helps to manage the development of other applications, combining all the required tools and features into one application. Some of the tools offered by an IDE include code editors, compilers, debuggers, automation tools e.t.c [57]

Visual Studio 2017
Created by Microsoft, Visual Studio provides all the tools necessary for building mobile, Windows, web and cloud applications with the possibility of installing only the components needed for the particular application type. The IDE contains code editing features, with an in built debugger and a code testing framework [58].

Resharper
Resharper is a code inspection extension for Visual Studio with the aim of improving code quality. Resharper analyzes code quality on the fly and offers suggestions where the code can be improved. It also enforces programming language specific code styling and formatting [59]

Source control
Also known as version control, or revision control, it is a component that manages changes in digital artifacts including documents or computer software as it is in this context. It helps software developers to work simultaneously on the same files, while helping to resolve any change, merging conflicts and keeping the
history or version of all changes to a given file [60].

**Git**

Git is a distributed version control system and it is one of the worlds most used [61]. It is a decentralized version control system (DVCS) as it keeps a full history of all the changes in each developers machine on the contrary to a centralized version control systems which only keeps a single copy with the full history on a centralized server [62].

**Github**

Github is a web based repository hosting service using git. It hosts both open source and business repositories and it will be used to host the software source code. [63]

**Git extensions**

Git extensions is a toolkit that is installed in Windows to work with git. It integrates with Windows explorer, Visual Studio and provides a user interface for git. Some of the git commands on git extensions include clone, commit, push pull and merge [64].

**Nuget package manager**

**Project planning**

A project planning tool was required to be able to plan and deliver the planned milestones effectively and efficiently. A trello kanban board was used to create, organize and prioritize the different tasks and user stories. The board kept a back log of user stories and tasks. It was configured to show the overall status and progress of the project, based on the status of the user stories in different columns (To do, On Going, Testing and Done) [65]
In this chapter we will discuss the middleware that was developed in this project. We start with listing the requirements, then we look at how parts of the middleware were ported from OIOSAML.NET to ASP.NET Core, and the different components making up the middleware. The chapter ends with a discussion of how the components are put together, work together as a whole, to provide a fully functioning SAML 2.0 authentication middleware for ASP.NET Core.

Saml2.Authentication.Core is the authentication middleware that was created during the thesis. It is a software application implementing the SAML 2.0 authentication protocol in the ASP.NET Core framework for web application development. The software brings SAML into the world of ASP.NET Core, by creating an authentication middleware/handler, that supports the HTTP Redirect and HTTP Artifact SAML bindings.

The middleware can easily be added into any ASP.NET Core web application (SP) as a dependency, and configured into the HTTP request processing pipeline with minimal effort. When used, the middleware takes over user challenging for authentication, creating SAML authentication requests (<AuthnRequest>) and redirecting to the configured identity provider for authentication.

After the user enters valid credentials in the identity provider’s login form, the middleware is capable of receiving the authentication response by HTTP Redirect or HTTP artifact binding, validating the SAMLResponse, getting the assertion, reading identity information (subject, username, name etc) from the validated assertion, setting a local session cookie, restoring the RelayState, and redirecting back to the service provider’s requested page, or resource while maintaining the integrity and confidentiality of the exchanged identity, and the communication itself.
4.1 Requirements and specification

As mentioned earlier, the waterfall method of software development was used when creating the software. During the analysis phase, the following specifications and requirements were uncovered.

<table>
<thead>
<tr>
<th>#</th>
<th>Requirement/Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Should have support for HTTP Redirect Binding</td>
</tr>
<tr>
<td>2.</td>
<td>Should have support for HTTP Artifact binding through SAMLResponse</td>
</tr>
<tr>
<td>3.</td>
<td>Should have support for SP-initiated Single Sign-On</td>
</tr>
<tr>
<td>4.</td>
<td>Should have support for SP-initiated SLO</td>
</tr>
<tr>
<td>5.</td>
<td>Should be easily configurable into the ASP.NET Core 2 HTTP pipeline</td>
</tr>
<tr>
<td>6.</td>
<td>Should be able of keeping user sessions in a secure way using ASP.NET Core 2 cookie authentication</td>
</tr>
<tr>
<td>7.</td>
<td>Should support adding configuration values for IDP and SP using JSON and ASP.NET Core 2 configuration options</td>
</tr>
<tr>
<td>8.</td>
<td>Should be able to trigger the middleware/authentication handler using challenge/challenge result</td>
</tr>
<tr>
<td>9.</td>
<td>Should be able to send user to the page they were in application after authentication using RelayState</td>
</tr>
<tr>
<td>10.</td>
<td>Should validate SAML requests and responses using digital signatures according the SAML 2.0 specification (protect message integrity)</td>
</tr>
<tr>
<td>11.</td>
<td>Should stop/detect automated/replay requests and attacks</td>
</tr>
<tr>
<td>12.</td>
<td>Should be able to ensure that contents of SAML messages are only accessible by the intended recipient (confidentiality)</td>
</tr>
</tbody>
</table>

Table 1: Requirements and specifications
4.2 Porting OIOSAML.NET library into ASP.NET Core

To reduce the amount of work in creating the middleware, parts of the open source, full .NET framework, SAML class library from Digitaliserdk was used as the base. These parts include, the core SAML protocol types, schema types and assertion validation types [15] [66]. Since these types were written targeting the full .NET framework, they had to be ported into the .NET Core framework.

The .NET API Portability Analyzer tool [30] was used to analyze these types. The aim was to identify the full .NET APIs they used, and their portability to .NET Core. The table below gives a summary of the results. The full API portability report is given in appendix A.

<table>
<thead>
<tr>
<th>Submission Id</th>
<th>07113ced-343d-4ecc-bdc3-5d4b50a7f384</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Summary of .NET Core API portability analysis</td>
</tr>
<tr>
<td>Targets</td>
<td>.NET Core,.NET Framework,.NET Standard</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assembly name</th>
<th>Target Framework</th>
<th>.NET Core</th>
<th>.NET Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>dk.nita.saml20 (OIOSAML.NET)</td>
<td>.NETFramework,Version=v4.7.1</td>
<td>81,85</td>
<td>70,53</td>
</tr>
</tbody>
</table>

Table 2: Summary of .NET Core API portability analysis

The report from the portability analysis (appendix A) gives information about supported and not supported APIs in .NET Core. The tool is supposed to be capable of giving some hints on how to fix the APIs flagged as not supported. In the case of partially porting OIOSAML.NET, the tool only managed to give recommendation for only about 1% of the used types.

From table 2 we can see that the OIOSAML.NET library targets the .NET framework version 4.7.1, and that, the analyzed target types to port where 81% portable to .NET Core, and 70% portable to .NET Standard. Theoretically, this meant that 81% of the code in the core SAML protocol types, schema types and assertion validation types, could be used in making the middleware. This did not end up being the case, as making the 19% changes to work led to changing more of the existing code. This transformed into roughly 40% of the code in these types, being used as the base for the SAML 2.0 authentication middleware.
4.3 Architecture

Figure 8 shows the architecture of the authentication middleware as placed within the ASP.NET Core HTTP pipeline. The middleware receives and handles only SAML authentication and logout HTTP requests. Other requests are passed on to the next middleware in the pipeline. There are three main components that make up the middleware; HandleChallengeAsync, SignOutAsync and HandleRequestAsync. Their functions are described below, while their concrete implementations are discussed in section 4.5.

![Architecture diagram]

**AuthnRequest** Authentication requests initiated by the service provider.

**SAMLResponse** Authentication response from identity provider to service provider.

**SAMLRequest** Authentication request from service provider to identity provider (external redirect).

**AuthnResponse** Authentication response from middleware to service provider (internal redirect).

**HandleChallengeAsync** Component for handling service provider authentication requests.
**SignOutAsync** Component for handling service provider logout requests.

**HandleRequestAsync** Component for handling authentication (HTTP redirect, HTTP Artifact) and logout SAMLResponse from identity provider.

### 4.4 Solution structure

The solution is made up of two projects; the middleware (**Saml2.Authentication.Core**), and a demo web application (**DemoWebApp**) which was used for testing the middleware. The classes making up the middleware are logically grouped into folders of related functionality. These includes, **Validation**; for classes handling validation of requests and responses, **Bindings**; for all binding related functionality, **Factories**; for classes responsible for creating authentication requests and responses, **Utils**; for utilities like digital signature, encryption and serialization helpers, **Extensions**; for classes that extend standard Microsoft types (strings, StringBuilder, ClaimsPrincipal, HttpRequest, HttpResponse) among others.
The figure below represent the solution as seen from Microsoft Visual Studio 2017 IDE. Appendix V shows the complete class diagram.

Figure 9: Solution structure
4.5 The SAML 2.0 authentication handler

The Saml2Handler is the core of the middleware. It is an ASP.NET Core authentication handler that coordinates the communication between the client (SP), and the IDP. It contains all the necessary methods and dependencies required for handling authentication challenges, authentication request-response exchanges, logout requests, and logout responses. To do the work, the handler depends on other services, factories and binding classes that are injected into the handler using the inversion of control mechanism. This is provided by the ASP.NET Core framework by default. Some of the dependencies are described below. Their full implementations are given in 8.2.

IOptionsMonitor<Saml2Options> An instance of the Saml2Options object, providing default middleware configuration options, including the Assertion Consumer Service and Single Logout URLs among other configuration options (Appendix F).

ILoggerFactory An instance of the LoggerFactory used for configuring the logging system.

UrlEncoder Used for URL character encoding.

ISystemClock An abstraction of the system clock.

ISamlService An instance of the SamlService which contains methods for creating SAML request URLs and handling SAMLResponses (Appendix J).

IHttpRedirectBinding An instance of the HttpRedirectBinding object containing HTTP Redirect binding related implementations. These includes, building authentication request URL and handling HTTP Redirect binding SAMLResponse (Appendix P).

IHttpArtifactBinding An instance of the HttpArtifactBinding object containing implementations to handle HTTP Artifact binding. This includes but is not limited to; getting SAMLart from SAMLResponse and resolving the assertion from the ArtifactResolutionService (Appendix O).

ISaml2ClaimFactory An instance of the Saml2ClaimFactory which receives a validated assertion, reads the transferred identity information and transforms the identity values into a list of claims (Appendix M).
The following sub sections gives an insight into the functionality provided by the authentication handler. The methods making up the handler are described. The full implementation of the handler is given in appendix B.

### 4.5.1 Handling SAMLResponse

The `HandleRequestAsync` method is called on every HTTP Request into the web application. It gives a way of performing logic on every request and determining if the HTTP Request should be handled by the middleware or not. The method returns true if the request has been handled and false if not, allowing the HTTP pipeline to forward the request to the next middleware in the HTTP pipeline [11]. Basically, this method handles SAMLResponses from the identity provider. These can either be authentication request responses (HTTP Redirect or Artifact binding) or logout responses. It uses three concrete methods to handle sign-in with HTTP Redirect, signout and sign-in using HTTP Artifact binding. These are described further down.

```csharp
public async Task<bool> HandleRequestAsync()
{
    if (await HandleSignIn())
        return true;

    if (await HandleSignOut())
        return true;

    return await HandleHttpArtifact();
}
```

Listing 4: HandleRequestAsync method

### 4.5.2 Handling logout requests

The `SignOutAsync` method handles SP-initiated logout requests. It takes in one parameter of `AuthenticationProperties` type, passed from the client. It contains the redirectURI where the middleware will redirect to after a successful authentication, among other `RelayState` values. The method calls the SamlService to create the logout request URL with the following parameters;

**LogoutRequestId** A unique identifier which should corresponds to the expected
InResponseTo value of the logoutResponse. The LogoutRequestId is kept in a protected cookie.

**SessionIndex** A value indicating the session on the identity provider that the service provider requests to be destroyed. This value is obtained from the previous assertion.

**Subject** A value indicating the session owner. This value is also obtained from the previous assertion.

**RelayState** A value indicating the state which should be restored after logout. This value is the AuthenticationProperties parameter which is confidentially protected using the data protection API [67].
The method transfers control to the Identity Provider by setting the HTTP Response to redirect to the created logout request URL. This triggers the logout on the identity provider.

```csharp
public Task SignOutAsync(AuthenticationProperties properties)
{
    _logger.LogDebug($"Entering {nameof(SignOutAsync)}", properties);

    var logoutRequestId = CreateUniqueId();
    var cookieOptions = Options.RequestIdCookie
        .Build(Context, Clock.UtcNow);
    Response.Cookies.Append(Options.RequestIdCookie.Name,
        Options.StringDataFormat.Protect(logoutRequestId),
        cookieOptions);

    var relayState = Options.StateDataFormat.Protect(properties);
    var sessionIndex = Context.User.GetSessionIndex();
    var subject = Context.User.GetSubject();

    var logoutRequestUrl = _samlService.GetLogoutRequest(
        logoutRequestId, sessionIndex, subject, relayState);

    _logger.LogDebug(
        $"Method={nameof(SignOutAsync)}.
        Redirecting to saml identity provider for SLO.
        Url={logoutRequestUrl}");

    Context.Response.Redirect(logoutRequestUrl, true);
    return Task.CompletedTask;
}
```

Listing 5: SignOutAsync method

**4.5.3 Handling authentication requests**

The HandleChallengeAsync method is the equivalent of the SignOutAsync method, but for handling authentication requests using the HTTP Redirect binding. It is
triggered by the client when user tries to access a restricted resource (how to trigger authentication is discussed further down). The method has one parameter of type `AuthenticationProperties` containing RelayState. This includes a redirectURI to redirect to, after authentication, among other state values. It uses the `SamlService` instance to get the actual authentication request URL with the following parameters:

**AuthnRequestId** A unique identifier which should corresponds to the InResponseTo value of the expected SAMLResponse. The AuthnRequestId is kept in a protected cookie.

**RelayState** A value indicating the state of the service provider that should be restored after authentication. This value is the `AuthenticationProperties` parameter which is confidentially protected using the data protection API [67].

**AssertionConsumerServiceUrl** Represents the URL for the Assertion Consumer Service, capable of receiving and handling a SAMLResponse from the Identity Provider. This value is added into the `<AuthenticationRequest>` as configured in the identity provider.

The method transfers control to the Identity Provider by setting the HTTP Response to redirect to the created authentication request URL. This triggers the login form on the identity provider.

```csharp
protected override Task HandleChallengeAsync(
    AuthenticationProperties properties)
{
    _logger.LogDebug($"Entering {nameof(HandleChallengeAsync)}",
        properties);

    var authnRequestId = CreateUniqueId();

    var deleteCookieOptions = Options.RequestIdCookie
        .Build(Context, Clock.UtcNow);
    Response.DeleteAllRequestIdCookies(
        Context.Request, deleteCookieOptions);

    var cookieOptions = Options.RequestIdCookie
```
.Build(Context, Clock.UtcNow);

    Response.Cookies.Append(Options.RequestIdCookie.Name,
        Options.StringDataFormat.Protect(authnRequestId),
        cookieOptions);

    var relayState = Options.StateDataFormat.Protect(properties);
    var requestUrl = _samlService
        .GetAuthnRequest(
            authnRequestId,
            relayState,
            $"{Request.GetBaseUrl()}
                /{Options.AssertionConsumerServiceUrl}"));

    _logger.LogDebug(
        $"Method={nameof(HandleChallengeAsync)}."
        Redirecting to saml identity provider for SSO.
        Url={requestUrl}");}

        Context.Response.Redirect(requestUrl, true);
        return Task.CompletedTask;

Listing 6: HandleChallengeAsync method

4.5.4 Handling logout response

The HandleSignOut() method receives and handles SP-initiated single logout responses from the identity provider.

It is a part of the HandleRequestAsync method, and it is called on every request into the web application. It compares the configured SingleLogoutServiceUrl and the HTTP Request path to decide whether to handle the HTTP request or not. It gets the SAMLResponse from the HTTP Request including the round tripped RelayState; gets the original LogoutRequestId from cookie and validates it against the InResponseTo value of the logout response; validates the SAMLResponse and uses the cookie middleware to logout the user from the ser-
vice provider. The RelayState is unprotected and the middleware returns control to the provided redirectURI.

```csharp
private async Task<bool> HandleSignOut()
{
    if (!Request.Path.Value.EndsWith(
        Options.SingleLogoutServiceUrl,
        StringComparison.OrdinalIgnoreCase))
    {
        return false;
    }

    _logger.LogDebug($"Entering {nameof(HandleSignOut)}");
    if (!_httpRedirectBinding.IsValid(Context.Request))
    {
        return false;
    }

    var uri = new Uri(Context.Request.GetEncodedUrl());
    var response = _httpRedirectBinding
        .GetResponse(Context.Request);
    var authenticationProperties =
        Options.StateDataFormat.Unprotect(response.RelayState)
            ?? new AuthenticationProperties();

    var initialLogoutRequestId = GetRequestId();
    if (!_samlService.IsLogoutResponseValid(
        uri, initialLogoutRequestId))
    {
        return false;
    }

    await Context.SignOutAsync(
        Options.SignOutScheme,
        authenticationProperties);

    var cookieOptions = Options.RequestIdCookie
        .Build(Context, Clock.UtcNow);
    Context.Response.DeleteAllRequestIdCookies(
        Context.Request, cookieOptions);

    _logger.LogDebug($"Method={nameof(HandleSignOut)}.");
}
Received and handled sp initiated logout response. Redirecting to {redirectUrl}“);

    var redirectUrl = GetRedirectUrl(authenticationProperties);
    Context.Response.Redirect(redirectUrl, true);
    return true;
}

Listing 7: HandleSignOut method

4.5.5 Handling SAML authentication response (SAMLResponse)

HTTP Redirect

The HandleSignin() method handles SAMLResponse for authentication requests. These are the HTTP Requests with the configured Assertion Consumer Service URL as the request path. This is compared with the configured value and any other request whose path does not match the configured AssertionConsumerServiceUrl value is ignored.

It uses the SamlService (HandleHttpRedirectResponse method) to; get the validated SAMLResponse; get and validate the assertion; read the transferred identity information as claims, and use the claims to sign-in the user for the configured SignInScheme. This creates a session cookie with a user principal containing the given claims. The RelayState is unprotected and the middleware returns control to the provided redirectURI (J). The service provider can then read the session cookie to get the transferred identity information from the identity provider as claims and use the session cookie for access control.

private async Task<bool> HandleSignIn()
{
    if (!Request.Path.Value.EndsWith(
        Options.AssertionConsumerServiceUrl,
        StringComparison.OrdinalIgnoreCase))
        return false;
    _logger.LogDebug($"Entering {nameof(HandleSignIn)}");
if (!_httpRedirectBinding.IsValid(Context.Request))
    return false;

var initialAuthnRequestId = GetRequestId();
var result = _httpRedirectBinding
    .GetResponse(Context.Request);

var base64EncodedSamlResponse = result.Response;
var assertion = _samlService.HandleHttpRedirectResponse(
    base64EncodedSamlResponse, initialAuthnRequestId);

var authenticationProperties =
    Options.StateDataFormat.Unprotect(result.RelayState)
    ?? new AuthenticationProperties();

await SignIn(assertion, authenticationProperties);

var cookieOptions = Options.RequestIdCookie
    .Build(Context, Clock.UtcNow);
Response.DeleteAllRequestIdCookies(
    Context.Request, cookieOptions);

_logger.LogDebug(
    $"Method={nameof(HandleSignin)}. Received and handled SSO redirect response. Redirecting to {redirectUrl}");

var redirectUrl = GetRedirectUrl(authenticationProperties);
Context.Response.Redirect(redirectUrl, true);
return true;

Listing 8: HandleSignin method
HTTP Artifact

The middleware supports the HTTP Artifact binding by SAMLResponse (SP initiated: Redirect → Artifact binding) as described in sub section (2.3.2). The method `HandleHttpArtifact()` handles SAMLResponses for authentication requests, and ignores all HTTP Requests whose request path is not equal to the configured `AssertionConsumerServiceUrl`.

It receives the artifact (SAMLart) and uses the `HandleHttpArtifactResponse` method of the SamlService (J) to resolve the artifact by sending an `<ArtifactResolve>` message to the source identity provider. The response is an `<ArtifactResponse>` message containing the respective assertion. The transferred identity information is read from the validated assertion by the Saml2ClaimFactory (M), and transformed into claims which are used to create a new ClaimsPrincipal. The ClaimsPrincipal is signed-in to a session cookie. The RelayState is unprotected and the redirectURI from the state is read.

The middleware transfers control to the service provider by setting the HTTP Response to redirect to the given redirectURI. As described in the `HandleSignin` method, the service provider can read identity information of the subject from the user principal claims and use the created session cookie for access control.

```csharp
private async Task<bool> HandleHttpArtifact()
{
    if (!Request.Path.Value.EndsWith(
        Options.AssertionConsumerServiceUrl,
        StringComparison.OrdinalIgnoreCase))
        return false;

    _logger.LogDebug($"Entering {nameof(HandleHttpArtifact)}");
    if (!_httpArtifactBinding.IsValid(Context.Request))
        return false;

    var initialAuthnRequestId = GetRequestId(); //TODO validate
    var assertion = _samlService
        .HandleHttpArtifactResponse(Context.Request);  
    var relayState = _httpArtifactBinding
        .GetRelayState(Context.Request); 
}```
The cookies responsible for keeping the unique request identifiers are deleted after the SAMLResponse is received.
4.5.6 Handling sign-in (creating session cookie)

The `SignIn` method is a helper used by the `HandleSignIn()` and `HandleHttpArtifact()` methods to read identity information from the assertion and transform it into claims. It also creates a new ClaimsPrincipal using the claims, and signs-in the created principal into a session cookie.

```csharp
private async Task SignIn(
    Saml2Assertion assertion,
    AuthenticationProperties authenticationProperties)
{
    var claims = _claimFactory.Create(assertion);
    var identity = new ClaimsIdentity(claims, Scheme.Name);
    var principal = new ClaimsPrincipal(identity);

    await Context.SignInAsync(
        Options.SignInScheme,
        principal,
        authenticationProperties);
}
```

Listing 10: Signin method
4.5.7 Unique SAML request identifiers

The helper method CreateUniqueId is used to create unique identifiers for each SAML request. These are necessary for the validation of received SAMLResponses. The value is validated against the InResponseTo value of the SAMLResponse, and the message is discarded if these do not match. This helps in protection against automated attacks, and the integrity of the request-response message exchange.

```
private static string CreateUniqueId(int length = 32)
{
    var bytes = new byte[length];
    using (var randomNumberGenerator
        = RandomNumberGenerator.Create())
    {
        randomNumberGenerator.GetBytes(bytes);
        var hex = new StringBuilder(bytes.Length * 2);
        foreach (var b in bytes)
            hex.AppendFormat("{0:x2}", b);

        return hex.ToString();
    }
}
```

Listing 11: CreateUniqueId method

4.5.8 Creating SAML 2.0 messages

The middleware creates three specific SAML 2.0 protocol messages during authentication and signout. These include authentication requests, logout requests and logout responses. Appendix L shows the Saml2MessageFactory class with the functions responsible for creating the protocol messages. Calling the function GetXml() on the resulting objects automatically creates the XML version of the request by serializing the values into XML strings.
4.6 Configuration

4.6.1 AuthenticationBuilder

The AuthenticationBuilder class gives an extension method AddScheme which is used to configure custom authentication handlers into the HTTP pipeline as authentication schemes. An authentication scheme represents a unique identifier for the current authentication middleware. This adds the custom authentication middleware into the ASP.NET Core authentication pipeline [68].

Listing 12 shows how the Saml2Handler is added into the application’s authentication middleware pipeline. The complete implementation is given in appendix D.

```
public static AuthenticationBuilder AddSaml(
    this AuthenticationBuilder builder,
    string authenticationScheme,
    string displayName,
    Action<Saml2Options> configureOptions)
{
    builder.Services.TryAddEnumerable(
        ServiceDescriptor.Singleton<
            IPostConfigureOptions<Saml2Options>,
            Saml2PostConfigureOptions>();

    return builder.AddScheme<Saml2Options, Saml2Handler>(
        authenticationScheme, displayName, configureOptions);
}
```

Listing 12: Configuring the authenticationbuilder

The IPostConfigureOptions offers a way of adding configuration values to the authentication handler after all other configurations have occurred. This is useful, especially when some configuration values have to be instantiated before use. Objects like the state DataProtectionProvider instance fall into this category [69]. Appendix E shows the post configuration of the DataProtectionProvider which is used in the authentication middleware to protect the confidentiality of cookies and RelaysState.
4.6.2 Startup services

The middleware requires several services to be instantiated during startup. These services are used by the authentication handler to perform specific functions. The authentication handler gets instances of these services from the dependency injection (DI) container which is configured during startup. DI is supported by default in ASP.NET Core and the framework exposes an IServiceCollection extension which is used to add services into the inbuilt DI container [70].

```csharp
private static void AddRequiredServices(
    this IServiceCollection services)
{
    services.AddOptions();
    services.TryAddSingleton(resolver =>
        resolver.GetRequiredService<
            IOptions<Saml2Configuration>>().Value);

    services.TryAddTransient<ISaml2Validator, Saml2Validator>();
    services.TryAddTransient<ISaml2ClaimFactory,
        Saml2ClaimFactory>();
    services.TryAddTransient<ISamlProvider, SamlProvider>();
    services.TryAddTransient<ISamlMessageFactory,
        Saml2MessageFactory>();
    services.TryAddTransient<ISignatureProviderFactory,
        SignatureProviderFactory>();
    services.TryAddTransient<IHttpRedirectBinding,
        HttpRedirectBinding>();
    services.TryAddTransient<IHttpArtifactBinding,
        HttpArtifactBinding>();
    services.TryAddTransient<ISamlService, SamlService>();
}
```

Listing 13: Configuring start-up services

Listing 13 shows how the required services are added into the DI container. Other dependencies that need to be configured during startup include but are not limited to signing certificates. The middleware requires signing certificates...
for both SP and IDP during runtime. These are used for digitally signing, encrypting and verifying protocol messages as required by the SAML 2.0 specifications [7]. Three different ways of reading and configuring the signing certificates are implemented. These can be added as a file in the file system, a thumbprint or certificate name from the certificate store. Appendix C gives a detailed overview of how start-up services are configured in the middleware.

4.6.3 Bringing it all together

The above sections have given and described the different components making up the authentication middleware. This section describes how these components are put together, work together as a whole, to provide a fully functioning SAML 2.0 authentication middleware for ASP.NET Core. It explains, the minimal configuration required for the middleware as used in the DemoWebApp.

ASP.NET Core applications use an application start-up class called Startup. This is the entry point of any ASP.NET Core based web application, and it runs once when the application starts. The class is made of of two methods ConfigureServices which is used to configure application services, and the Configure method, which is used to configure the application request processing pipeline (HTTP pipeline) [71].

The middleware is configured within the ConfigureServices method with a few lines of code. Listing 14, shows how to configure the middleware in Startup. A brief description for what each line does is given.

```csharp
public void ConfigureServices(IServiceCollection services)
{
    // Omitted code

    // Add Saml2.Authentication.Core configuration values
    services.Configure<Saml2Configuration>(
        Configuration.GetSection("Saml2");

    // Add Saml2.Authentication.Core services to the container
    services.AddSaml();

    // Add signing certificates by thumbprint
    services.AddSigningCertificates(Configuration
        ["Saml2:ServiceProviderConfiguration:
        SigningCertificateThumprint"],
```
Listing 14: Configuring Saml2.Authentication.Core in startup

In listing 14, we see that the SAML configuration values are read from a JSON configuration file, with a section called "Saml2". This is usually done in the appsettings.json file. The settings used during testing of the DemoWebApp are given in listing 15. See appendices G, H and I for all the possible SAML 2.0 configuration variables.
"a69d709f32f14b484f1a46b3d514b42388a0c3f6",
  "SingleSignOnService":
    "https://saml2login.my.salesforce.com/idp/endpoint
     /HttpRedirect",
  "SingleSignOutService":
    "https://saml2login.my.salesforce.com/services/auth/
     idp/saml2/logout",
  "ProtocolBinding":
    "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
},
"ServiceProviderConfiguration": {
  "EntityId": "https://localhost:44344",
  "Name": "Saml2.auth",
  "SigningCertificateThumprint":
    "83331e94db6a841ed921ff86e41624a6eb78c8d1"
}
}
4.7 Triggering authentication

The SAML 2.0 authentication middleware can be invoked by creating an action (IActionResult), that returns an instance of the ChallengeResult class. The authentication scheme name and properties (AuthenticationProperties) should also be supplied. This is called "challenging" the middleware. When challenged, the middleware triggers the SAML 2.0 authentication handler (HandleChallengeAsync method) as described in section 4.5 to handle the challenge. This will create a SAML authentication request (AuthnRequest) and redirect to the configured IDP for authentication [72].

The listing below gives an example of code "challenging" the middleware.

```csharp
public IActionResult ExternalLogin(string returnUrl = null)
{
    var provider = "AuthenticationSchemeName";

    // Request a redirect to the external login provider.
    var redirectUrl = Url.Action(nameof(ExternalLoginCallback),
                                "Account", new { returnUrl });

    var properties = _signInManager.ConfigureExternalAuthenticationProperties(
        provider,
        redirectUrl);

    return Challenge(properties, provider);
}
```

Listing 16: Triggering authentication

The action in listing 16 defines a redirectUrl variable which is added into the authentication properties object. The object is passed into to the middleware and it represents the RelayState value that is protected and round tripped to the identity provider. The redirectUrl is the callback action that the middleware will redirect to after authentication.
An example callback action is given below.

```csharp
public async Task<IActionResult> ExternalLoginCallback(
    string returnUrl = null)
{
    var info = await _signInManager.GetExternalLoginInfoAsync();
    if (info == null)
    {
        return RedirectToAction(nameof(Login));
    }

    // Sign in the user with this external login provider.
    var result = await _signInManager
        .ExternalLoginSignInAsync(
            info.LoginProvider,
            info.ProviderKey,
            isPersistent: false,
            bypassTwoFactor: true);

    if (result.Succeeded)
    {
        _logger.LogInformation("User logged in with
            {Name} provider.",
            info.LoginProvider);
        return RedirectToLocal(returnUrl);
    }
}
```

Listing 17: External authentication callback

From the example in listing 16, the middleware redirects to the **ExternalLoginCallback** action after authentication. The line

```csharp
var info = await _signInManager.GetExternalLoginInfoAsync();
```

reads the session cookie that was set by the authentication middleware. This contains a user principal object with the transferred identity information from the identity provider. This example goes on to sign-in the received user principal in to a new cookie which is used for access control.
4.8 Triggering signout

Triggering signout is done the same way as triggering authentication, except that the action triggering signout calls the `HttpContext.SignOutAsync` method. The authentication scheme name to signout from and the `AuthenticationProperties` object [73] are required. This triggers the `SignOutAsync` method of the authentication handler which creates a logout request for the currently logged-in subject and redirects to the identity provider. The following example shows an action that triggers signout using the SAM 2.0 authentication middleware.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task Logout()
{
    var redirectUrl = Url.Action(
        nameof(ExternalLogoutCallback),
        "Account");

    var properties = _signInManager.ConfigureExternalAuthenticationProperties("AuthenticationSchemeName",
    redirectUrl);

    await HttpContext.SignOutAsync("AuthenticationSchemeName",
    properties);
}
```

Listing 18: Triggering signout
4.9 Ethical and legal considerations

A part of this project uses classes ported from the open source project OIOSAML.Net 2.0 made by digitaliser.dk [15]. The OIOSAML.Net 2.0 [15] library is released under the Mozilla Public Licence. The SAML 2.0 authentication middleware for ASP.NET Core is also released under the same licence [66]. This license allows the author to grant modification rights to any contributor of the project. It allows for use, modification, sub licensing, distribution of both source code and the modified source code [17].
5 Results

This chapter analyses the results captured from testing the created authentication middleware to verify the research questions, requirements and specifications formulated during the project start. The results are based on an evaluation of the authentication middleware using a demo ASP.NET web application (DemoWebApp), configured to use a real world SAML identity provider (https://www.salesforce.com) for authentication and managing identities.

5.1 Configuration

Salesforce offers the possibility of creating and configuring it as a SAML 2.0 identity provider. This allows third party web applications to give users access to resources using identities created and maintained by Salesforce, and transferred to the web application using the SAML 2.0 framework. For this to work, configuration has to be done at salesforce.com, to enable Salesforce as a SAML identity provider, and also configure the service provider as an application connected to the identity provider [74].

5.1.1 Configuring salesforce as a SAML identity provider

To configure the identity provider, the domain name for the identity provider (https://saml2login.my.salesforce.com/) was created and a self-signed certificate with the SHA-256 signature algorithm was generated. These were added to the identity provider's configuration settings in salesforce.com [74].
Figure 10 shows the configured Salesforce identity provider with entity id (issuer) https://saml2login.my.salesforce.com. Appendix S shows the identity provider's metadata.

5.1.2 Configuring DemoWebApp as the service provider

The DemoWebApp was added in Salesforce as a service provider application, and connected to the configured SAML identity provider. All the required SAML 2.0 configuration values for service providers were added in the web app settings. Some of these are described below [75].

Start URL The web application’s start URL.
Entity Id The web application’s unique identifier.
ACS URL The web application’s Assertion Consumer Service URL.
Single Logout URL The web application’s single logout endpoint.
Issuer The SAML identity provider’s entity identifier.
Figure 11 shows the **DemoWebApp** application’s configuration as a service provider that is connected to the identity provider (issuer) with entity id https://saml2login.my.salesforce.com.

After configuring the **DemoWebApp** as a service provider in Salesforce, it was also necessary to configure the **DemoWebApp** application to use Salesforce as the SAML identity provider. This was done in the application settings file as seen in listing 15.
5.1.3 Managing identities

An identity with the username jkmu@XXXXXX.com, and federation id 123456789 was added and connected to the service provider DemoWebApp. This was done in the user management portal at Salesforce.

The figure below shows the Identity information of the added user.

![Figure 12: Users in salesforce](image-url)
5.2 Verification of results

The results were verified using case studies which were designed to cover the overall SAML 2.0 signin and signout using the authentication middleware, as configured in the DemoWebApp. The aim was to verify the fulfillment of the research questions identified in section 1.7. For each case study, the middleware was configured to use the HTTP redirect binding. A brief discussion of how the middleware solves each of the research question is given together with the achieved test results.

5.2.1 Case study 1: Initiating authentication (SAMLRequest)

This case study verifies research question 1. The service provider initiates authentication using an action that returns an instance of the ChallengeResult class. The class is instantiated with the authentication scheme name given to the authentication middleware, and a set of given properties. The HandleChallengeAsync component receives the challenge and handles it by creating an authentication request and redirecting to the external identity provider for login. This is discussed in sections 4.5.3 and 4.7.

Results

Triggering authentication using the middleware created an authentication request (AuthnRequest) with a unique identifier as shown in listing 19.

Listing 19: AuthnRequest

```xml
<?xml version="1.0"?>
<q1:AuthnRequest
  ID="b3dc5923fff09491d9f66ec6000c"
  Version="2.0"
  IssueInstant="2018-05-05T15:55:03.7250138Z"
  Destination="https://saml2login.my.salesforce.com/../HttpRedirect"
  ForceAuthn="false"
  IsPassive="false"
  ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
  AssertionConsumerServiceURL="https://localhost:44344/Saml2/AssertionConsumerService"
```
The middleware created an authentication request URL by concatenating the SAMLRequest (AuthnRequest), RelayState, SignAlg (signing algorithm), Signature and the destination URL (single sign-on endpoint of the identity provider). The SAMLRequest and the RelayState were deflate encoded and URL encoded. The SignAlg was URL encoded while the signature was converted into a base 64 string and URL encoded [7].

Figure 13 shows the created authentication request URL.

Figure 13: Authentication request URL

Setting the created authentication request URL as the location to redirect to in the ASP.NET Core HTTP response pipeline resulted to a redirect to the identity provider’s login page [76].

5.2.2 Case study 2: Handling authentication response (SAMLResponse)
This case study verifies research question 2. Authentication responses are handled by the HandleRequestAsync component 4.5.1. This component is called on
every HTTP request that comes through the middleware. To determine whether to handle the request, the component has a function `HandleSignIn` (8) which listens to HTTP requests with the configured Assertion Consumer Service URL as the request path. A match means that this is an authentication response. The method initiates verification of the response, reading of the identity data from the received assertion and signs in the user into a local session cookie.

**Results**

Upon entering the user credentials at the identity providers login page, the middleware received an authentication response in an embedded html form. The form contained the roundtripped RelayState value as added in the authentication request, and a base 64 encoded SAMLResponse. Appendix T shows the actual SAMLResponse as received while listing 20 shows the received RelayState, whose value is equivalent to the value in the authentication request URL in figure 13.

```
BMHGcoMgAADQL3JGY3A5ohgXXA16TeDCkC41dmLdaunX9z2/
R41TnMMOMvWkHyZSDm/EZu8o5acYXs6cyber2TQb/W8kD+
HDe83ofO10reuGo4ttsLxug6UFK0BR3Igud4t4Sawzuwok5Pvom7CJX+
zzsD1p938fHiC3n0TElyCpFQvFyoo0+daxd2keRoEO5kKcA/
m9Jot4RqJB9r6Fjfuba2Xbb/xPAeuyMrdrk4q+NAtw3KUqCMwp1FcjHasCZ2SfwAAAP/
/AwA=
```

Listing 20: RelayState from authentication response

The **SAMLResponse** was then decoded and validated. The result was the authentication response in form of an XML assertion. Appendix U, shows the decoded SAMLResponse while listing 21 shows the actual assertion.

Listing 21: Validated assertion

```
<saml:Assertion
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    ID="_f580b191d80f081b766398a5738360aa1525535780073"
    <saml:Issuer
        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
        https://saml2login.my.salesforce.com</saml:Issuer>
```
MDAwMDJDdjNyMRcwFQYDVQQKDA5TYWxlc2ZvcmNlLmNvbTEWMBQGA1UEBwwNU2FwIEZyYW5jaW5jbzELMA4GA1UEBwwNU2FsZXNmb3JjZS5jb20xFDAd</saml:Signature>

<saml:Subject>
  <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">jkmu@XXXXXXXX.com</saml:NameID>
  <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
    <saml:SubjectConfirmationData InResponseTo="b3dc5923fff09491d9f66ec6000c" NotOnOrAfter="2018-05-05T16:01:20.073Z" Recipient="https://localhost:44344/Saml2/AssertionConsumerService" />
  </saml:SubjectConfirmationData>
</saml:Subject>
<saml:AudienceRestriction>
</saml:AudienceRestriction>
<saml:Conditions>
    <saml:AuthnContext>
      <saml:AuthnContextClassRef>
        urn:oasis:names:tc:SAML:2.0:ac:classes:unspecified
      </saml:AuthnContextClassRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
  <saml:AttributeStatement>
    <saml:Attribute
      Name="userId"
      NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
      <saml:AttributeValue
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="xs:anyType">0051r0000089ZM2</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute
      Name="username"
      NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
      <saml:AttributeValue
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="xs:anyType">jkmu@XXXXXXXX.com</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute
      Name="email"
      NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
      <saml:AttributeValue
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="xs:anyType">jkmu@XXXXXXXX.com</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute
      Name="is_portal_user"
      NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
      68
    </saml:Attribute>
  </saml:AttributeStatement>
</saml:Conditions>
From the listing above, we can see that the identity information of the user was transferred as attribute statements in the assertion. The identity information included userId, username and email. This information was used to create a user principal object in ASP.NET Core, by transforming the attribute statements into claims. The **ClaimsPrincipal** (user) was then signed-in to create a session cookie (2).
5.2.3 Case study 3: Maintaining session
This case study verifies research question 3. The identity information received from the identity provider needs to be persisted into a session for access control. Case study 2 describes how the middleware creates a user from the received identity information. This is persisted into a local session cookie by the SignIn method (4.5.6).

Results
Figure 14 is an example of a callback endpoint where the authentication middleware returned control after a successful authentication. It shows how the service provider can read the transferred identity information as claims on the authenticated user principal. Note that, the claims have the same identity information as the attribute statements in the received assertion in listing 21.

Based on the individual requirements of the service provider, the developer can choose to continue using the already set session cookie, or, read the identity information from the session cookie, create a local identity, probably with extra information, connect these identities and sign-in the new user principal.

```csharp
[HttpGet] [AllowAnonymous] public async Task<IActionResult> ExternalLoginCallback(string returnUrl = null, string remoteError = null)
{
    if (remoteError != null)
    {
        var info = await _signInManager.GetExternalLoginInfoAsync();
        return RedirectToAction(nameof(login));
    }

    var identity = new ClaimsIdentity(
        new ClaimsPrincipal(new ClaimsIdentity(new ClaimsInfo(new Dictionary<string, object>()
        {
            [ClaimTypes.NameIdentifier] = "SamI",
            [ClaimTypes.Name] = "SamI",
        })), "SamI")
    );

    if (result.Success)
    {
        return Redirect(result.ReturnUrl);
    }

    // If the user does not have an account, then ask the user to create an account.
    var data = new { ReturnUrl = returnUrl, LogInProvider = _signInManager.Provider }, email = info.Principal.FindFirstValue(ClaimTypes.Email);
    return View("ExternalLogin", new ExternalLoginViewModel { email = email });
}
```

Figure 14: User principal and claims
5.2.4 Case study 4: SP-initiated logout

This case study verifies research question 4. Service provider logout requests are initiated by calling the `HttpContext.SignOutAsync` method from ASP.NET Core, with the authentication handler’s scheme name (4.8). This triggers the `SignOutAsync` component of the authentication middleware which creates a logout request and transfers control to the identity provider for logout (4.5.2).

Results

After clicking logout on the DemoWebApp application, the `SignOutAsync` method of the authentication handler was triggered. This created a `LogoutRequest` XML with a unique logout identifier. The current session identifier (SessionIndex) and subject to logout was added to the request. The resulting XML string was deflate encoded, URL encoded, signed (same as the authentication request) and added as a SAMLRequest parameter to the identity provider’s single logout endpoint. The resulting logout request URL was set as the location to redirect to in the ASP.NET Core HTTP response [76]. The full logout request URL is given in figure 15, while listing 22 shows the `LogoutRequest` XML message.

Listing 22: LogoutRequest message

```xml
<?xml version="1.0"?>
<q1:LogoutRequest
  ID="eb45015c9e787c7ad186626dbb90de"
  Version="2.0"
  IssueInstant="2018-05-05T16:30:52.0319493Z"
  Destination="https://saml2login.my.salesforce.com/..//logout"
  Reason="urn:oasis:names:tc:SAML:2.0:logout:user"

  xmlns:q1="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:urn:oasis:names:tc:SAML:2.0:assertion"

  https://localhost:44344
</Issuer>

>NameID xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
  jkmu@XXXXXXXX.com
</NameID>

<q1:SessionIndex>
_f404918957093a3e80346cf72542aadc1525537793123
</q1:SessionIndex>
```
Figure 15: LogutRequest URL
5.2.5 Case study 5: Handling SP-initiated logout response

This case study verifies research question 5. As described in section 4.5.4, the HandleSignOut() method handles logout responses. Being a part of the HandleRequestAsync component which is called on every request, it checks for requests that target the single logout service and disregards the rest. The method destroys the session cookie for valid responses by calling the ASP.NET Core method HttpContext.SignOutAsync with the specified SignOutScheme of the middleware (7).

Results

The logout request in case study 4 triggered a redirect to the identity provider, and a redirect back from the identity provider with a SAMLResponse (LogoutResponse). The middleware validated the LogoutResponse, removed the local session cookie and redirected back to the DemoWebApp application. Figure 16 shows the received logout response while listing 22 shows the validated logout response message.

```
<?xml version="1.0" encoding="UTF-8"?>
<sam1:Issuer
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:SAML:2.0:assertion http://docs.oasis-open.org/saml/v2.0/saml-core-2.0.xsd"
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">
  <saml:Issuer>
    <saml:IDPProfile URI="urn:oasis:names:tc:SAML:2.0:assertion:IDPProfile#SAML2.0-Inf"/>
  </saml:Issuer>
</sam1:Issuer>
```

Figure 16: LogoutResponse
A SAML 2.0 Authentication Middleware for ASP.NET Core

Listing 23: Validated LogoutResponse message

```xml
<ns2:LogoutResponse xmlns:ns2="urn:oasis:names:tc:SAML:2.0:assertion">
  <ns2:Issuer>https://saml2login.my.salesforce.com</ns2:Issuer>
  <ns2:Status>
  </ns2:Status>
</ns2:LogoutResponse>
```

5.2.6 Case study 6: Middleware configuration
This case study verifies research question 6. The middleware implements extension methods for adding it into the application's middleware pipeline (12), and instantiating the required services during start up (13). These are simplified into simple single functions which are called in the application's startup class (14). Runtime application variables (service and identity provider configuration values) are added as JSON values in a file called `appsettings.json` placed at the root of the application (15).

Results
See section 5.1 for discussion on how the DemoWebApp was configured during result verification.

5.2.7 Case study 7: Monitoring and tracing
This case study verifies research question 7. The middleware offers the possibility of tracing specific events within the middleware using logs. Debug log level messages are added to the authentication handler when entering and leaving different functions. These can be persisted to the file system or database based on the service provider requirements.
6 Discussion

6.1 Threat analysis

This section discusses some of the threats associated with SAML-based systems. These are studied before and authors in [25] list a comprehensive list of possible threats. In this thesis, we base our analysis on the aforementioned list.

6.1.1 Denial of service (DOS)

SAML-based systems are susceptible to denial of service attacks whereby an attacker can issue user agent redirects into the identity or service provider target endpoints. Handling SAML requests requires computing resources to parse XML, validate digital signatures and decrypt XML messages. An attacker can potentially generate "fake" requests faster than the system can handle causing the system to drown [25].

Oasis recommends several countermeasures for denial of service attacks in SAML-based systems [25]. These include;

- Requiring client authentication using client certificates which are traceable in case of DOS attacks.
- Requiring the use of signed SAML requests which increases the work required by the attacker to generate and embed digital signatures to the requests.
- Restricting access to the Assertion Consumer Service (ACS) and Single Logout (SLO) endpoints to known service and identity providers.
Other threats associated with SAML-based systems are given below [25].

<table>
<thead>
<tr>
<th>#</th>
<th>Threat</th>
<th>Description</th>
<th>Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Stolen Assertion</td>
<td>An eavesdropper copying a real user’s assertion and using it to impersonate the user</td>
<td>Confidentiality protection 6.2.1, reject expired assertions, short values for NotBefore and NotAfter attributes</td>
</tr>
<tr>
<td>2.</td>
<td>Man In the Middle Attack</td>
<td>A malicious site impersonating the user</td>
<td>Validate the Recipient value in the assertion</td>
</tr>
<tr>
<td>3.</td>
<td>Forged Assertion</td>
<td>Malicious altering of an assertion</td>
<td>Ensure message integrity by verifying the message signature</td>
</tr>
<tr>
<td>4.</td>
<td>Modification or exposure of state information</td>
<td>Altering of RelayState information</td>
<td>Ensure message confidentiality and integrity of RelayState using encryption. See 2.3.2 for more information</td>
</tr>
</tbody>
</table>

Table 3: Other threats associated with SAML-based systems

6.2 Security requirements
This section looks at some of the security requirements identified in section 4.1, and discusses the security controls implemented in the middleware to address them.

6.2.1 Confidentiality
Confidentiality is the ability to ensure that the contents of SAML messages send from either an identity provider to a service provider or vice versa are only accessible by the intended recipient. This requires the protection of the message while in transit and also on the message level [25].

Oasis recommends the use of secure network protocols such as TLS and IP Security protocols (IPSec) between service and identity providers to ensure the confidentiality of messages in transit [25].

On the message level, the confidentiality of SAML messages can be ensured by the use of XML Encryption of selected XML elements within the message [25]. The authentication middleware supports this by allowing identity providers to issue encrypted assertions. The middleware is capable of reading the encrypted assertion and decrypting it using the service provider’s private key.
Listing 24 shows how the middleware reads and decrypts the contents of an encrypted assertion.

```csharp
public XmlElement GetDecriptedAssertion(
    XmlElement xmlElement,
    AsymmetricAlgorithm privateKey)
{
    var encryptedList =
        xmlElement.GetElementsByTagName(
            EncryptedAssertion.ELEMENT_NAME,
            Saml2Constants.ASSERTION);
    var assertion = (XmlElement) encryptedList[0];
    if (assertion == null)
        throw new Saml2Exception("Missing assertion");

    var encryptedAssertion = new Saml2EncryptedAssertion(
        (RSA) privateKey);
    encryptedAssertion.LoadXml(assertion);
    encryptedAssertion.Decrypt();

    return encryptedAssertion.Assertion.DocumentElement;
}
```

Listing 24: Decrypting an encrypted assertion

### 6.2.2 Message Integrity

Message integrity is the ability to prove that a received SAML message has not been altered under transport. The integrity of SAML messages should be ensured when in transit using secure network protocols such as TLS and IPSec. To protect message integrity on message level, XML Signature is used to create a signature of the original SAML message. The signature is transported together with the original message, giving the receiver the ability to verify and confirm that the original message was not altered [25].

The middleware protects and verifies the integrity of both SAMLRequests and SAMLResponses. SAMLRequests are protected by creating and embedding the signature of the original SAMLRequest in the SAMLRequest, according to the
SAML specifications. See page 19 and case study 5.2.1 for more details.

The integrity of a received SAMLResponse is confirmed by verifying the signature that is embedded on the SAML message. Messages whose integrity cannot be verified are discarded. It is possible to configure the middleware to not check message signatures in appsettings using the provided OmitAssertionSignatureCheck configuration value.

Listing 25 shows how the middleware verifies the signature of a received assertion.

```csharp
public Saml2Assertion GetValidatedAssertion(
    XmlElement assertionElement,
    AsymmetricAlgorithm key,
    string audience,
    bool omitAssertionSignatureCheck = false)
{
    var keys = new List<AsymmetricAlgorithm> { key };
    var assertion = new Saml2Assertion(assertionElement, keys,
        AssertionProfile.Core, new List<string> { audience }, false);
    if (!omitAssertionSignatureCheck)
    {
        //TODO: This is checked automatically if autovalidation is on
        if (!assertion.CheckSignature(keys))
        {
            throw new Saml2Exception(
                "Invalid signature in assertion");
        }
    }
    if (assertion.IsExpired())
    {
        throw new Saml2Exception("Assertion is expired");
    }
    return assertion;
}
```

Note that, it is OPTIONAL to check and confirm the integrity and confiden-
tiality of SAML requests and responses according to the SAML specifications [7]. Service providers and identity providers should choose this depending on the security requirements of their SAML systems.

6.2.3 Replay attacks
Replay attacks are caused by re-submission of forms containing SAML responses, in order to gain access to protected resources. Oasis recommends the use of one-time-use values in assertions that can be validated by the service provider [25].

The middleware protects against replay attacks by creating and embedding a unique request identifier on each authentication or logout request. The value is kept on an encrypted local cookie. This value is expected to be echoed back in the SAML response inside the `InResponseTo` XML attribute. On receiving any SAML response, the value of the unique request identifier cookie is read, and compared to the `InResponseTo` attribute value. Messages whose values do not match are discarded. Listing 26 shows how the middleware checks for replay attacks in SAML responses.

```csharp
public void CheckReplayAttack(
    XmlElement element,
    string originalSamlRequestId)
{
    var inResponseToAttribute =
        element.Attributes["InResponseTo"];
    if (inResponseToAttribute == null)
    {
        throw new Saml2Exception(
            "Received a response message that did
            not contain an InResponseTo attribute");
    }
    var inResponseTo = inResponseToAttribute.Value;
    if (string.IsNullOrEmpty(originalSamlRequestId)
        || string.IsNullOrEmpty(inResponseTo))
    {
        throw new Saml2Exception(
            "Received a response message that did
            not contain an InResponseTo attribute");
    }
}
```
Listing 26: Replay attack check

6.2.4 Privacy consideration
SAML transfers the identity information about an authenticated subject, from an identity provider to a service provider as SAML statements with associated attributes. It is possible that at times the subject, the service provider or the identity provider would like to keep the accessibility of some of this identity information restricted. Example of such information might be attributes about the subject’s health or finances [25].

In this situation, Oasis recommends that, the privacy laws and regulations in the respective jurisdiction should be considered when deploying SAML-based systems. The identity provider and service provider should address any privacy concerns within their systems [25].

6.3 Advantages and disadvantages of the thesis
This section discusses the advantages and disadvantages of using the created SAML 2.0 authentication middleware for ASP.NET Core. Also given, are the benefits of using SAML 2.0 as an authentication framework.

6.3.1 Advantages
Usability The middleware is easy to use. It requires only a few lines of code to add it into the HTTP request processing pipeline.

Flexibility The middleware is reusable and flexible, it is designed to support any service and identity provider that implements the SAML 2.0 framework.
**Save development time and costs** The middleware will save software developers and enterprises the time needed to integrate with SAML identity providers, helping them to focus on developing features that give direct returns to their organizations.

**Extendable** Support for other SAML binding combinations (use cases) can be added with ease.

### 6.3.2 Disadvantages

**Testing** The middleware has not been thoroughly tested for features, performance and security. It is possible that the current implementation contains bugs and security problems. It is also not known the number of concurrent authentication requests the middleware can service in a busy environment.

**Partial support for SAML binding combinations** The middleware supports only two SAML binding combinations (SP initiated: Redirect → POST binding, SP initiated: Redirect → Artifact binding) and SP initiated logout with the HTTP redirect binding.

### 6.4 Benefits of SAML 2.0

Geyer, in the saml.xml.org journal gives several advantages of using SAML as an authentication protocol. Some of these are discussed below.

**Platform neutrality** SAML separates security and authentication from service applications, making security and application logic independent of each other [77].

**Improved online user experience** SAML Single Sign-On allows users to enter their credentials once at a trusted identity provider, and use them to access online services on different service providers [77].

**Reduced administrative costs for service providers** The cost of maintaining identities is handled by the identity provider [77].

**Transfer of risk** The risks around maintaining and properly protecting identity information is handled by the identity provider [77].
7 Conclusion

Saml2.Authentication.Core is a fork of the OIOSAML.Net repository. The SAML schema C# classes were copied, ported and used as the base for creating the Saml2.Authentication.Core middleware. The middleware is reusable; it can be used to integrate any service provider, with any real world SAML-based identity provider, using a few lines of code. It is flexible; it is possible to configure it to support variable service and identity provider configurations.

The middleware is capable of handling both authentication and logout request-response messages. It transfers identity information from an identity provider to a service provider through a session cookie. The information in the session cookie can be used to create and signin a local user or grant access to local resources depending on the requirements of the service provider.

The middleware supports;

- Service provider initiated authentication requests with;
  - HTTP redirect binding SAMLRequest and HTTP POST binding SAML-Response.
  - HTTP redirect binding SAMLRequest and HTTP artifact binding SAML-Response.

- Single Logout Profile
  - Service provider initiated logout

The middleware does not support identity provider initiated logouts. This has negative consequences in Single Sign-On environments as the service provider will not respond to Single Logout requests from the identity provider.

7.1 Future Work

This thesis represents a fairly working solution for bringing SAML 2.0 authentication framework into ASP.NET Core. The solution needs core software development improvements to make it production ready. Below are interesting areas for further work that were out of scope for this study:
• Code refactoring to restructure the existing code to improve readability and reduce complexity.
• Thorough manual and automated testing to find bugs.
• Testing with other SAML-based identity providers.
• Adding support for other SAML binding combinations.
• Packaging and releasing the software as a nuget package.
Bibliography


A SAML 2.0 Authentication Middleware for ASP.NET Core


8 Materials

8.1 Source code

8.2 Project planning
Project planning tool: https://trello.com/b/2cnwqV3
Appendices
A .NET Core API port analysis in the OIOSAML.NET assembly

Target types and member support

<table>
<thead>
<tr>
<th>Not supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>T:System.ServiceModel.BasicHttpBinding</td>
</tr>
<tr>
<td>T:System.ServiceModel.ChannelFactory'1</td>
</tr>
<tr>
<td>T:System.ServiceModel.Channels.IRequestChannel</td>
</tr>
<tr>
<td>T:System.ServiceModel.EndpointAddress</td>
</tr>
<tr>
<td>T:System.ServiceModel.HttpBindingBase</td>
</tr>
<tr>
<td>T:System.ServiceModel.ICommunicationObject</td>
</tr>
</tbody>
</table>
T:System.Web.HttpCookie
T:System.Web.HttpCookieCollection
T:System.Web.HttpRequest
T:System.Web.IHttpHandler
T:System.Web.SessionState.IRequiresSessionState
T:System.Web.UI.AttributeCollection
T:System.Web.UI.ClientScriptManager
T:System.Web.UI.Control
T:System.Web.UI.ControlCollection
T:System.Web.UI.CssStyleCollection
T:System.Web.UI.HtmlControls.HtmlControl
T:System.Web.UI.HtmlControls.HtmlHead
T:System.Web.UI.HtmlControls.HtmlInputControl
T:System.Web.UI.HtmlControls.HtmlLink
T:System.Web.UI.HtmlTextWriterStyle
T:System.Web.UI.LiteralControl
T:System.Web.UI.Page
T:System.Web.UI.WebControls.HyperLink
T:System.Web.UI.WebControls.Label
T:System.Web.UI.WebControls.Panel
T:System.Web.UI.WebControls.WebControl
Supported: 2.0+
namespace Saml2.Authentication.Core.Authentication
{
    /// <inheritdoc>
    /// <cref>AuthenticationHandler</cref>
    /// </inheritdoc>
    /// <summary>
    /// Saml2Handler class. Handles communication between the service provider and the identity provider.
    /// Handles authentication challenges, authentication responses, Logout requests and logout responses
    /// </summary>
    public class Saml2Handler : AuthenticationHandler<Saml2Options>,
                                IAuthenticationRequestHandler,
                                IAuthenticationSignOutHandler
    {
}
private readonly ISaml2ClaimFactory _claimFactory;
private readonly IHttpArtifactBinding _httpArtifactBinding;
private readonly IHttpRedirectBinding _httpRedirectBinding;
private readonly ILogger _logger;
private readonly ISamlService _samlService;

public Saml2Handler(
    IOptionsMonitor<Saml2Options> options,
    ILoggerFactory logger,
    Encoder encoder,
    ISystemClock clock,
    ISamlService samlService,
    IHttpRedirectBinding httpRedirectBinding,
    IHttpArtifactBinding httpArtifactBinding,
    ISaml2ClaimFactory claimFactory)
    : base(options, logger, encoder, clock)
{
    _logger = logger.CreateLogger(typeof(Saml2Handler));
    _samlService = samlService;
    _httpRedirectBinding = httpRedirectBinding;
    _httpArtifactBinding = httpArtifactBinding;
    _claimFactory = claimFactory;
}

/// <inheritdoc />
/// <summary>
/// Called on every request.
/// Handles authentication and signout response requests.
/// </summary>
/// <returns>True or false if request is handled</returns>
public async Task<bool> HandleRequestAsync()
{
    if (await HandleSignIn())
        return true;
    if (await HandleSignOut())
        return true;
```

```
protected override Task<AuthenticateResult> HandleAuthenticateAsync()
{
    return Task.FromResult(AuthenticateResult.Fail("Not supported"));
}

/// <inheritdoc />
/// <summary>
/// Handles service provider authentication requests.
/// Creates <AuthnRequest></AuthnRequest> and redirects to identity provider
/// </summary>
/// <param name="properties"></param>
/// <returns>Task.CompletedTask</returns>
protected override Task HandleChallengeAsync(AuthenticationProperties properties)
{
    _logger.LogDebug(
        $"Entering {nameof(HandleChallengeAsync)}", properties);

    var authnRequestId = CreateUniqueId();

    var deleteCookieOptions = Options.RequestIdCookie .Build(Context, Clock.UtcNow);
    Response.DeleteAllRequestIdCookies(Context.Request, deleteCookieOptions);

    var cookieOptions = Options.RequestIdCookie .Build(Context, Clock.UtcNow);
    Response.Cookies.Append(Options.RequestIdCookie.Name, Options.StringDataFormat.Protect(authnRequestId),
                              deleteCookieOptions);

    return Task.CompletedTask;
}
A SAML 2.0 Authentication Middleware for ASP.NET Core

cookieOptions);

var relayState =
⊥ Options.StateDataFormat.Protect(properties);
var requestUrl = _samlService.GetAuthnRequest(
authnRequestId, relayState,
$"{Request.GetBaseUrl()}/
{Options.AssertionConsumerServiceUrl}");

_logger.LogDebug(
  $"Method={nameof(HandleChallengeAsync)}. 
  Redirecting to saml identity provider for SSO. 
  Url={requestUrl}");

Context.Response.Redirect(requestUrl, true);
return Task.CompletedTask;

}/// <summary>
/// Handles sigout response requests from identity 
provider
/// </summary>
/// <returns>True/false</returns>
private async Task<bool> HandleSignOut()
{
  if (!Request.Path.Value.EndsWith(
    Options.SingleLogoutServiceUrl,
    StringComparison.OrdinalIgnoreCase))

    return false;

_logger.LogDebug($"Entering
  {nameof(HandleSignOut)}");

if (!_httpRedirectBinding.IsValid(Context.Request))

    return false;

var uri = new Uri(Context.Request.GetEncodedUrl());
//idp initiated logout. BUG:Context.User and cookies
⊥ are not populated
if (_httpRedirectBinding

var logoutReponse = _samlService.GetLogoutReponse(uri);
    return false;

var relayState = _httpRedirectBinding.GetCompressedRelayState(Context.Request);
var url = _samlService.GetLogoutResponseUrl(logoutReponse, relayState);

await Context.SignOutAsync(Options.SignOutScheme, new AuthenticationProperties());
Context.Response.Redirect(url, true);
return true;

// sp initiated logout
var response = _httpRedirectBinding.GetResponse(Context.Request);

var initialLogoutRequestId = GetRequestId();
if (!_samlService.IsLogoutResponseValid(uri, initialLogoutRequestId))
    return false;

await Context.SignOutAsync(Options.SignOutScheme, authenticationProperties);
```csharp
var cookieOptions = Options.RequestIdCookie
    .Build(Context,
            Clock.UtcNow);

Context.Response
    .DeleteAllRequestIdCookies(Context.Request,
                               cookieOptions);

var redirectUrl =
    GetRedirectUrl(authenticationProperties);

_logger.LogDebug($"Method={nameof(HandleSignOut)}.Received and handled sp initiated logout response. Redirecting to {redirectUrl}";

Context.Response.Redirect(redirectUrl, true);
return true;
}

/// <summary>
/// Handles authentication response requests from idp HTTP Redirect binding
/// </summary>
/// <returns>True/false</returns>
private async Task<bool> HandleSignIn()
{
    if (!Request.Path.Value.EndsWith(
        Options.AssertionConsumerServiceUrl,
        StringComparison.OrdinalIgnoreCase))
        return false;

_logger.LogDebug($"Entering {nameof(HandleSignIn)}");

if (!_httpRedirectBinding.IsValid(Context.Request))
    return false;

var initialAuthnRequestId = GetRequestId();
```

105
```csharp
var result = _httpRedirectBinding.GetResponse(Context.Request);
var base64EncodedSamlResponse = result.Response;
var assertion = _samlService.HandleHttpRedirectResponse(base64EncodedSamlResponse, initialAuthnRequestId);

var authenticationProperties = Options.StateDataFormat.Unprotect(result.RelayState) ?? new AuthenticationProperties();
await SignIn(assertion, authenticationProperties);

var cookieOptions = Options.RequestIdCookie .Build(Context, Clock.UtcNow);
Response.DeleteAllRequestIdCookies(Context.Request, cookieOptions);

var redirectUrl = GetRedirectUrl(authenticationProperties);

_logger.LogDebug($"Method={nameof(HandleSignIn)}. Received and handled SSO redirect response. Redirecting to {redirectUrl}"");
Context.Response.Redirect(redirectUrl, true);
return true;

/// <summary>
/// Handles authentication request responses from idp
/// HTTP Artifact binding
/// </summary>
/// <returns>True/false</returns>
private async Task<bool> HandleHttpArtifact() {
    if (!Request.Path.Value.EndsWith("/ authenticators"))
```
Options.AssertionConsumerServiceUrl,
StringComparison.OrdinalIgnoreCase))
    return false;

_logger.LogDebug($"Entering
← {nameof(HandleHttpArtifact)}");

if (!_httpArtifactBinding.IsValid(Context.Request))
    return false;

//TODO validate inResponseTo
var initialAuthnRequestId = GetRequestId();

var assertion = _samlService
    .HandleHttpArtifactResponse(Context.Request);

var relayState = _httpArtifactBinding
    .GetRelayState(Context.Request);
var authenticationProperties =
    Options.StateDataFormat
    .Unprotect(relayState) ?? new
    ← AuthenticationProperties();

await SignIn(assertion, authenticationProperties);

var cookieOptions = Options.RequestIdCookie
    .Build(Context,
    ← Clock.UtcNow);
Response.DeleteAllRequestIdCookies(Context.Request,
    ← cookieOptions);

var redirectUrl =
    ← GetRedirectUrl(authenticationProperties);

_logger.LogDebug(
    $"Method={nameof(HandleHttpArtifact)}.Received and handled SSO artifact response.
    Redirecting to {redirectUrl}";
Context.Response.Redirect(redirectUrl, true);
return true;

/// <summary>
/// Gets identity information from assertion and signs in user
/// </summary>
/// <param name="assertion">Saml2Assertion</param>
/// <param name="authenticationProperties">AuthenticationProperties</param>
/// <returns></returns>
private async Task SignIn(
    Saml2Assertion assertion,
    AuthenticationProperties authenticationProperties)
{
    var claims = _claimFactory.Create(assertion);
    var identity = new ClaimsIdentity(claims, Scheme.Name);
    var principal = new ClaimsPrincipal(identity);

    await Context.SignInAsync(
        Options.SignInScheme,
        principal,
        authenticationProperties);
}

/// <summary>
/// Creates a unique cryptorandom id
/// </summary>
/// <param name="length"></param>
/// <returns></returns>
private static string CreateUniqueId(int length = 32)
{
    var bytes = new byte[length];
    using (var randomNumberGenerator = RandomNumberGenerator.Create())
    {
        randomNumberGenerator.GetBytes(bytes);
        var hex = new StringBuilder(bytes.Length * 2);
        foreach (var b in bytes)
            hex.Append((char)(b >> 4));
```csharp
    hex.AppendFormat("{0:x2}", b);

    return hex.ToString();

    /// <summary>
    /// Gets the requestId from cookie
    /// </summary>
    /// <returns>RequestId</returns>
    private string GetRequestId()
    {
        var requestIdCookie = Request.GetRequestIdCookie();
        if (string.IsNullOrEmpty(requestIdCookie))
            throw new ArgumentNullException(
                nameof(requestIdCookie));

        return Options.StringDataFormat
            .Unprotect(requestIdCookie);
    }

    /// <summary>
    /// Gets the redirect url from authentication properties
    /// </summary>
    /// <param name="authenticationProperties">AuthenticationProperties</param>
    /// <returns>RedirectUrl</returns>
    private string GetRedirectUrl( AuthenticationProperties authenticationProperties)
    {
        return authenticationProperties.RedirectUri
            .IsNotNullOrEmpty()
            ? authenticationProperties.RedirectUri
            : Options.DefaultRedirectUrl;
    }
```
C Saml2ServiceCollectionExtensions

```csharp
using System;
using System.IO;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Options;
using Saml2.Authentication.Core.Extensions;
using Saml2.Authentication.Core.Factories;
using Saml2.Authentication.Core.Options;
using Saml2.Authentication.Core.Providers;
using Saml2.Authentication.Core.Services;
using Saml2.Authentication.Core.Validation;

namespace Microsoft.Extensions.DependencyInjection
{
    public static class Saml2ServiceCollectionExtensions
    {
        public static IServiceCollection AddSaml(
            this IServiceCollection services)
        {
            if (services == null)
                throw new ArgumentNullException(nameof(services));

            services.AddRequiredServices();
            return services;
        }

        private static void AddRequiredServices(
            this IServiceCollection services)
        {
            services.AddOptions();
        }
    }
}
```
services.TryAddSingleton(resolver =>
    resolver.GetRequiredService<IOptions<Saml2Configuration>>().Value);

services.TryAddSingleton<ISaml2Validator, Saml2Validator>();
services.TryAddSingleton<ISaml2ClaimFactory, Saml2ClaimFactory>();
services.TryAddSingleton<ISamlProvider, SamlProvider>();
services.TryAddSingleton<ISaml2MessageFactory, Saml2MessageFactory>();
services.TryAddSingleton<ISignatureProviderFactory, SignatureProviderFactory>();
services.TryAddSingleton<IHttpRedirectBinding, HttpRedirectBinding>();
services.TryAddSingleton<IHttpArtifactBinding, HttpArtifactBinding>();
services.TryAddSingleton<ISamlService, SamlService>();
}

/// <summary>
/// Add signing certificates by thumbprint
/// </summary>
/// <param name="services"></param>
/// <param name="serviceProviderCertificateThumbprint">ServiceProvider's certificate thumbprint</param>
/// <param name="identityProviderCertificateThumbprint">IdentityProvider's certificate thumbprint</param>
/// <returns></returns>
public static IServiceCollection AddSigningCertificates(
    this IServiceCollection services,
    string serviceProviderCertificateThumbprint,
    string identityProviderCertificateThumbprint)
{
    if (services == null)
        throw new ArgumentNullException(nameof(services));
return AddSigningCertificates(services,
    GetSigningCertificates(
        X509FindType.FindByThumbprint,
        serviceProviderCertificateThumbprint
            .TrimSpecialCharacters(),
        identityProviderCertificateThumbprint
            .TrimSpecialCharacters()));
}

public static IServiceCollection AddSigningCertificates(
    this IServiceCollection services,
    X509FindType findType,
    string serviceProviderCertificateName,
    string identityProviderCertificateName)
{
    if (services == null)
        throw new ArgumentNullException(nameof(services));

    return AddSigningCertificates(services,
        GetSigningCertificates(
            findType,
            serviceProviderCertificateName,
            serviceProviderCertificateName));
}

/// <summary>
/// Add signing certificates from file
/// </summary>
/// <param name="services"></param>
/// <param name="serviceProviderCertificateFileName">ServiceProvider's certificate filename</param>
/// <param name="identityProviderCertificateFileName">IdentityProvider's certificate filename</param>
/// <returns></returns>
public static IServiceCollection AddSigningCertificatesFromFile(
    this IServiceCollection services,
    string serviceProviderCertificateFileName,
    string identityProviderCertificateFileName)
if (services == null)
    throw new ArgumentException(nameof(services));

if (string.IsNullOrEmpty(serviceProviderCertificateFileName))
    throw new ArgumentNullException(nameof(serviceProviderCertificateFileName));

if (string.IsNullOrEmpty(identityProviderCertificateFileName))
    throw new ArgumentNullException(nameof(identityProviderCertificateFileName));

var serviceProviderCertificateFullFileName =
    !Path.IsPathRooted(serviceProviderCertificateFileName)
    : serviceProviderCertificateFileName;

var serviceProviderCertificate =
    new X509Certificate2(
        serviceProviderCertificateFullFileName);

var identityProviderCertificateFullFileName =
    !Path.IsPathRooted(identityProviderCertificateFileName)
    : identityProviderCertificateFileName;

var identityProviderCertificate =
    new X509Certificate2(
        identityProviderCertificateFullFileName);

return AddSigningCertificates(
    services,
    new SigningCertificate(
        identityProviderCertificate,
private static X509Certificate2 GetCertificate(
    string findValue,
    X509FindType findType,
    StoreName storeName = StoreName.My,
    StoreLocation storeLocation = StoreLocation.LocalMachine,
    bool validOnly = false)
{
    var store = new X509Store(storeName, storeLocation);
    try
    {
        store.Open(OpenFlags.ReadOnly);
        var found = store.Certificates.Find(
            findType,
            findValue,
            validOnly);

        if (found.Count == 0)
        {
            var searchDescriptor = SearchDescriptor(
                findValue,
                findType,
                storeName,
                storeLocation,
                validOnly);

            var msg = $"A configured certificate could not be found in the certificate store.{searchDescriptor}";
            throw new Exception(msg);
        }
        if (found.Count > 1)
        {
            var searchDescriptor = SearchDescriptor(
                findValue,
                findType,}
storeName,
storeLocation,
validOnly);

var msg =
$"Found more than one certificate in the
→ certificate store.
Make sure you don’t have duplicate
→ certificates installed.
{searchDescriptor}";

throw new Exception(msg);
}
return found[0];
}
finally
{
    store.Close();
}
}

private static string SearchDescriptor(
    string findValue,
    X509FindType findType,
    StoreName storeName,
    StoreLocation storeLocation,
    bool validOnly)
{
    var message =
$"The certificate was searched for in
→ {storeLocation}/{storeName},
{findType}={findValue}, validOnly={validOnly}";

    if (findType == X509FindType.FindByThumbprint
        && findValue?.Length > 0 && findValue[0] == 0x200E)
    
        message =
"The configuration for the certificate searches by
→ thumbprint but has an invalid character in the
→ thumbprint string." + message;
private static SigningCertificate GetSigningCertificates(
    X509FindType findType,
    string serviceProviderCertificateName,
    string identityProviderCertificateName)
{
    if (string.IsNullOrEmpty(
        serviceProviderCertificateName))
        throw new ArgumentNullException(
            nameof(serviceProviderCertificateName));

    if (string.IsNullOrEmpty(
        identityProviderCertificateName))
        throw new ArgumentNullException(
            nameof(identityProviderCertificateName));

    var serviceProviderCertificate =
        GetCertificate(
            serviceProviderCertificateName, 
            findType);

    var identityProviderCertificate =
        GetCertificate(
            identityProviderCertificateName, 
            findType);

    var certificates = new SigningCertificate(
        identityProviderCertificate,
        serviceProviderCertificate);

    return certificates;
}

private static void
    CheckServiceProviderCertificatePrivateKey(
        SigningCertificate signingCertificates)
{
    if (signingCertificates == null)
        throw new ArgumentNullException(
            nameof(signingCertificates));
if (!signingCertificates.ServiceProvider.HasPrivateKey)
    throw new InvalidOperationException(
        "Certificate does not have a private key.");
}

private static IServiceCollection AddSigningCertificates(
    IServiceCollection services,
    SigningCertificate signingCertificates)
{
    CheckServiceProviderCertificatePrivateKey(
        signingCertificates);
    services.AddSingleton<ICertificateProvider>(
        new CertificateProvider(
            signingCertificates));
    return services;
}
}
D  Saml2Extensions

```csharp
using System;
using Microsoft.AspNetCore.Authentication;
using Microsoft.Extensions.DependencyInjection.Extensions;
using Microsoft.Extensions.Options;
using Saml2.Authentication.Core.Options;

namespace Microsoft.Extensions.DependencyInjection
{
    public static class Saml2Extensions
    {
        public static AuthenticationBuilder AddSaml(
            this AuthenticationBuilder builder)
            => builder.AddSaml(
                Saml2Defaults.AuthenticationScheme, _ => {
                    ↓
                });

        public static AuthenticationBuilder AddSaml(
            this AuthenticationBuilder builder,
            Action<Saml2Options> configureOptions)
            => builder.AddSaml(
                Saml2Defaults.AuthenticationScheme,
                configureOptions);

        public static AuthenticationBuilder AddSaml(
            this AuthenticationBuilder builder,
            string authenticationScheme,
            Action<Saml2Options> configureOptions)
            => builder.AddSaml(
                authenticationScheme,
                Saml2Defaults
                    .AuthenticationSchemeDisplayName,
                configureOptions);
    }
}
```
```csharp
public static AuthenticationBuilder AddSaml(
    this AuthenticationBuilder builder,
    string authenticationScheme,
    string displayName,
    Action<Saml2Options> configureOptions)
{
    builder.Services.TryAddEnumerable(
        ServiceDescriptor.Singleton<
            IPostConfigureOptions<Saml2Options>,
            Saml2PostConfigureOptions>(
        ) );

    return builder.AddScheme<Saml2Options, Saml2Handler>(
        authenticationScheme, displayName,
        configureOptions);
}
```
E  Saml2PostConfigureOptions

```csharp
using System.Text;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.DataProtection;
using Microsoft.Extensions.Options;

namespace Saml2.Authentication.Core.Options
{
    public class Saml2PostConfigureOptions :
        IPostConfigureOptions<Saml2Options>
    {
        private readonly IDataProtectionProvider
            _dataProtectionProvider;

        public Saml2PostConfigureOptions(
            IDataProtectionProvider dataProtectionProvider)
        {
            _dataProtectionProvider = dataProtectionProvider;
        }

        public void PostConfigure(string name, Saml2Options
            options)
        {
            options.DataProtectionProvider =
                options.DataProtectionProvider
                ?? _dataProtectionProvider;

            if (string.IsNullOrEmpty(options.SignOutScheme))
                options.SignOutScheme = options.SignInScheme;

            if (options.StateDataFormat == null)
            {
                var dataProtector =
                    options.DataProtectionProvider
```
.CreateProtector(typeof(Saml2Handler).FullName);
    options.StateDataFormat =
        new PropertiesDataFormat(dataProtector);
}

if (options.StringDataFormat == null)
{
    var dataProtector =
        options.DataProtectionProvider
            .CreateProtector(
                typeof(Saml2Handler).FullName,
                typeof(string).FullName);

    options.StringDataFormat = new
        SecureDataFormat<string >(
            new StringSerializer(),
            dataProtector);
}

internal class StringSerializer : IDataSerializer<string>
{
    public string Deserialize(byte[] data)
    {
        return Encoding.UTF8.GetString(data);
    }

    public byte[] Serialize(string model)
    {
        return Encoding.UTF8.GetBytes(model);
    }
}
namespace Saml2.Authentication.Core.Options
{
    public class Saml2Options : AuthenticationSchemeOptions
    {
        private CookieBuilder _requestIdCookie;

        public Saml2Options()
        {
            AssertionConsumerServiceUrl = "Saml2/AssertionConsumerService";
            SingleLogoutServiceUrl = "Saml2/SingleLogoutService";
            DefaultRedirectUrl = "/";
            SignInScheme = Saml2Defaults.SignInScheme;
            AuthenticationScheme = "Saml2/AuthenticationScheme";
            RequestIdCookieLifetime = TimeSpan.FromMinutes(10);

            _requestIdCookie = new RequestIdCookieBuilder(this)
            {
                Name = $"{Saml2Defaults.RequestIdCookiePrefix}.{Guid.NewGuid():N}",
                HttpOnly = true,
                SameSite = SameSiteMode.None,
                SecurePolicy = CookieSecurePolicy.SameAsRequest
            };
        }
    }
}
public string AssertionConsumerServiceUrl { get; set; }

public string SingleLogoutServiceUrl { get; set; }

public string DefaultRedirectUrl { get; set; }

public string SignInScheme { get; set; }

public string SignOutScheme { get; set; }

public string AuthenticationScheme { get; set; }

public ISecureDataFormat<AuthenticationProperties> StateDataFormat {
    get; set; }

public IDataProtectionProvider DataProtectionProvider {
    get; set; }

public TimeSpan RequestIdCookieLifetime { get; set; }

public CookieBuilder RequestIdCookie {
    get => _requestIdCookie;
    set => _requestIdCookie = value
        ?? throw new AuthenticationException(nameof(value));
}

public ISecureDataFormat<string> StringDataFormat { get; set; }

internal class RequestIdCookieBuilder :
    RequestPathBaseCookieBuilder {
    private readonly Saml2Options _options;
public RequestIdCookieBuilder(Saml2Options options)
{
    _options = options;
}

public override CookieOptions Build(
    HttpContext context,
    DateTimeOffset expiresFrom)
{
    var cookieOptions = base.Build(context, expiresFrom);

    if (!Expiration.HasValue ||
        !cookieOptions.Expires.HasValue)
        cookieOptions.Expires =
            expiresFrom.Add(
                _options.RequestIdCookieLifetime);

    return cookieOptions;
}
namespace Saml2.Authentication.Core.Options
{
    public class Saml2Configuration
    {
        public bool ForceAuth { get; set; }

        public bool IsPassive { get; set; }

        public bool SignAuthnRequest { get; set; }

        public string AuthnContextComparisonType { get; set; }

        public string[] AuthnContextComparisonItems { get; set; }

        public bool? AllowCreate { get; set; }

        public bool OmitAssertionSignatureCheck { get; set; }

        public IdentityProviderConfiguration IdentityProviderConfiguration { get; set; }

        public ServiceProviderConfiguration ServiceProviderConfiguration { get; set; }
    }
}
```csharp
using System.ComponentModel;

namespace Saml2.Authentication.Core.Options
{
    public class IdentityProviderConfiguration
    {
        public string EntityId { get; set; }

        public string Name { get; set; }

        public string IssuerFormat { get; set; }

        [DefaultValue("urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified")]
        public string NameIdPolicyFormat { get; set; } = "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified";

        [DefaultValue("http://www.w3.org/2001/04/xmlenc#sha256")]
        public string DigestAlgorithm { get; set; } = "SHA256";

        [DefaultValue("http://www.w3.org/2001/04/xmldsig-more#rsa-sha256")]
        public string HashingAlgorithm { get; set; } = "SHA256";

        [DefaultValue("urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect")]
        public string ProtocolBinding { get; set; } = "urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect";

        public string SingleSignOnService { get; set; }

        public string SingleSignOutService { get; set; }
    }
}
```
public string ArtifactResolveService { get; set; }

}
I ServiceProviderConfiguration

```csharp
namespace Saml2.Authentication.Core.Options {
    public class ServiceProviderConfiguration {
        public string EntityId { get; set; }
    }
}
```
using System;
using System.Text;
using System.Xml;
using dk.nita.sam120;
using Microsoft.AspNetCore.Http;
using Saml2.Authentication.Core.Factories;
using Saml2.Authentication.Core.Options;
using Saml2.Authentication.Core.Providers;
using Saml2.Authentication.Core.Validation;

using Saml2LogoutResponse = 

namespace Saml2.Authentication.Core.Services
{
    internal class SamlService : ISamlService
    {
        private readonly ICertificateProvider _certificateProvider;
        private readonly IHttpArtifactBinding _httpArtifactBinding;
        private readonly IHttpRedirectBinding _httpRedirectBinding;
        private readonly IdentityProviderConfiguration _identityProviderConfiguration;
        private readonly Saml2Configuration _saml2Configuration;
        private readonly ISaml2MessageFactory _saml2MessageFactory;
        private readonly ISaml2Validator _saml2Validator;
        private readonly ISamlProvider _samlProvider;
        private readonly ServiceProviderConfiguration _serviceProviderConfiguration;
public SamlService(
    IHttpRedirectBinding httpRedirectBinding,
    IHttpArtifactBinding httpArtifactBinding,
    ISaml2MessageFactory saml2MessageFactory,
    ICertificateProvider certificateProvider,
    ISamlProvider samlProvider,
    ISaml2Validator saml2Validator,
    Saml2Configuration saml2Configuration)
{
    _httpRedirectBinding = httpRedirectBinding;
    _httpArtifactBinding = httpArtifactBinding;
    _saml2MessageFactory = saml2MessageFactory;
    _certificateProvider = certificateProvider;
    _samlProvider = samlProvider;
    _saml2Validator = saml2Validator;
    _saml2Configuration = saml2Configuration;
    _identityProviderConfiguration =
        saml2Configuration.IdentityProviderConfiguration;
    _serviceProviderConfiguration =
        saml2Configuration.ServiceProviderConfiguration;
}

public string GetAuthnRequest(
    string authnRequestId,
    string relayState,
    string assertionConsumerServiceUrl)
{
    var signingCertificate =
        _certificateProvider.GetCertificate();

    var saml20AuthnRequest =
        _saml2MessageFactory
            .CreateAuthnRequest(
                authnRequestId,
                assertionConsumerServiceUrl);

    // check protocol binding if supporting more than HTTP-REDIRECT
    return _httpRedirectBinding.BuildAuthnRequestUrl(
        saml20AuthnRequest,
        signingCertificate.ServiceProvider.PrivateKey,
public string GetLogoutRequest(string logoutRequestId, string sessionIndex, string subject, string relayState)
{
    var signingCertificate = _certificateProvider.GetCertificate();

    var logoutRequest = _saml2MessageFactory.CreateLogoutRequest(logoutRequestId, sessionIndex, subject);

    return _httpRedirectBinding.BuildLogoutRequestUrl(logoutRequest, signingCertificate.ServiceProvider.PrivateKey, _identityProviderConfiguration.HashingAlgorithm, relayState);
}

public bool IsLogoutResponseValid(Uri uri, string originalRequestId)
{
    var signingCertificate = _certificateProvider.GetCertificate();

    var logoutMessage = _httpRedirectBinding.GetLogoutResponseMessage(uri, signingCertificate.IdentityProvider.PublicKey.Key);

    var logoutRequest = _samlProvider.GetLogoutResponse(logoutMessage);
_saml2Validator.CheckReplayAttack(
    logoutRequest.InResponseTo,
    originalRequestId);

    == Saml2Constants.StatusCodes.Success;
}

public Saml2LogoutResponse GetLogoutResponse(Uri uri)
{
    var signingCertificate =
        _certificateProvider.GetCertificate();

    var logoutResponse =
        _httpRedirectBinding.GetLogoutReponse(
            uri,
            signingCertificate.IdentityProvider.PublicKey.Key);

    if (!_saml2Validator.ValidateLogoutRequestIssuer(
            logoutResponse.OriginalLogoutRequest.Issuer.Value,
            _identityProviderConfiguration.EntityId))
    {
        logoutResponse.StatusCode =
            Saml2Constants.StatusCodes.RequestDenied;
    }

    return logoutResponse;
}

public string GetLogoutResponseUrl(
    Saml2LogoutResponse logoutResponse,
    string relayState)
{
    var signingCertificate =
        _certificateProvider.GetCertificate();

    var response =
        _saml2MessageFactory.CreateLogoutResponse(

    return response;
logoutResponse.StatusCode,
logoutResponse.OriginalLogoutRequest.ID);

return _httpRedirectBinding.BuildLogoutResponseUrl(
    response,
    signingCertificate.ServiceProvider.PrivateKey,
    _identityProviderConfiguration.HashingAlgorithm,
    relayState);
}

public Saml2Assertion HandleHttpRedirectResponse(
    string base64EncodedSamlResponse,
    string originalSamlRequestId)
{
    var samlResponseDocument =
        _samlProvider.GetDecodedSamlResponse(
            base64EncodedSamlResponse,
            Encoding.UTF8);
    var samlResponseElement =
        samlResponseDocument.DocumentElement;

    _saml2Validator.CheckReplayAttack(
        samlResponseElement,
        originalSamlRequestId);

    return
        !_saml2Validator.CheckStatus(samlResponseDocument)
            ? null
            : GetValidatedAssertion(samlResponseElement);
}

public Saml2Assertion
    HandleHttpArtifactResponse(HttpRequest request)
{
    var signingCertificate =
        _certificateProvider.GetCertificate();

    var artifact =
        _httpArtifactBinding.GetArtifact(request);
```csharp
var stream =
    _httpArtifactBinding.ResolveArtifact(artifact,
    _identityProviderConfiguration
    .ArtifactResolveService,
    _serviceProviderConfiguration.EntityId,
    signingCertificate.ServiceProvider);

var artifactResponseElement =
    _samlProvider
    .GetArtifactResponse(stream);

return
    GetValidatedAssertion(artifactResponseElement);
}

private Saml2Assertion GetValidatedAssertion(
    XElement samlResponseElement)
{
    var signingCertificate =
        _certificateProvider.GetCertificate();
    var assertionElement =
        _samlProvider.GetAssertion(
            samlResponseElement,
            signingCertificate
            .ServiceProvider.PrivateKey);

    return _saml2Validator.GetValidatedAssertion(
        assertionElement,
        signingCertificate
        .IdentityProvider.PublicKey.Key,
        _serviceProviderConfiguration.EntityId,
        _saml2Configuration.OmitAssertionSignatureCheck);
}
```
using System;
using System.IO;
using System.Text;
using System.Xml;
using dk.nita.saml20;
using dk.nita.saml20_Utils;
using Saml2.Authentication.Core_BINDINGS;

namespace Saml2.Authentication.Core_PROVIDERS{
    internal class SamlProvider : ISamlProvider
    {
        public XmlDocument GetDecodedSamlResponse(
            string base64SamlResponse,
            Encoding encoding)
        {
            var doc = new XmlDocument
            {
                XmlResolver = null,
                PreserveWhitespace = true
            };
            var samlResponse = encoding.GetString(
                Convert
                .FromBase64String(
                    base64SamlResponse));
            doc.LoadXml(samlResponse);
            return doc;
        }

        public XmlElement GetAssertion(

    }
XmlElement xmlElement,
AsymmetricAlgorithm privateKey)
{
  if (IsEncrypted(xmlElement))
    return GetDecryptedAssertion(xmlElement,
        privateKey);

  var assertionList = xmlElement
      .GetElementsByTagName(
        Assertion.ELEMENT_NAME,
        Saml2Constants.ASSERTION);

  var assertion = (XmlElement) assertionList[0];
  if (assertion == null)
    throw new Saml2Exception("Missing assertion");
  return assertion;
}

public LogoutResponse GetLogoutResponse(string
        logoutResponseMessage)
{
  var doc = new XmlDocument
  {
    XmlResolver = null,
    PreserveWhitespace = true
  };
  doc.LoadXml(logoutResponseMessage);

  var logoutResponse =
      (XmlElement)doc.GetElementsByTagName(
        LogoutResponse.ELEMENT_NAME,
        Saml2Constants.PROTOCOL)[0];
  return Serialization
    .DeserializeFromXmlString<LogoutResponse>(
        logoutResponse.OuterXml);
}

public XmlElement GetArtifactResponse(Stream stream)
{
A SAML 2.0 Authentication Middleware for ASP.NET Core

```csharp
var parser = new HttpArtifactBindingParser(stream);
if (!parser.IsArtifactResponse())
    return null;

var status = parser.ArtifactResponse.Status;
if (status.StatusCode.Value !=
    Saml2Constants.StatusCodes.Success)
    throw new Exception(
        $"Illegal status: {status.StatusCode} for ArtifactResponse");

return parser.ArtifactResponse.Any.LocalName
    != Response.ELEMENT_NAME ? null :
    parser.ArtifactResponse.Any;

public Status GetLogoutResponseStatus(string
    logoutResponseMessage)
{
    var doc = new XmlDocument
    {
        XmlResolver = null,
        PreserveWhitespace = true
    };
    doc.LoadXml(logoutResponseMessage);

    var statElem = (XmlElement)doc.GetElementsByTagName(
        Status.ELEMENT_NAME,
        Saml2Constants.PROTOCOL)[0];

    return Serialization
        .DeserializeFromXmlString<Status>(
            statElem OUTER Xml);
}

public XmlElement GetDecriptedAssertion(
    XmlElement xmlElement, AsymmetricAlgorithm
    privateKey)
{
    var encryptedList =
```
xmlElement.GetElementsByTagName(
    EncryptedAssertion.ELEMENT_NAME,
    Saml2Constants.ASSERTION);

var assertion = (XmlElement) encryptedList[0];
if (assertion == null)
    throw new Saml2Exception("Missing assertion");

var encryptedAssertion
    = new Saml2EncryptedAssertion((RSA) privateKey);
encryptedAssertion.LoadXml(assertion);
encryptedAssertion.Decrypt();

    return encryptedAssertion.Assertion.DocumentElement;
}

private static bool IsEncrypted(XmlElement element)
{
    var encryptedList = element.GetElementsByTagName(
        EncryptedAssertion.ELEMENT_NAME,
        Saml2Constants.ASSERTION);
    return encryptedList.Count == 1;
}
namespace Saml2.Authentication.Core.Factories
{
    internal class Saml2MessageFactory : ISaml2MessageFactory
    {
        private readonly IdentityProviderConfiguration _identityProviderConfiguration;
        private readonly Saml2Configuration _saml2Configuration;
        private readonly ServiceProviderConfiguration _serviceProviderConfiguration;

        public Saml2MessageFactory(Saml2Configuration saml2Configuration)
        {
            _saml2Configuration = saml2Configuration;
            _serviceProviderConfiguration = saml2Configuration.ServiceProviderConfiguration;
            _identityProviderConfiguration = saml2Configuration.IdentityProviderConfiguration;
        }

        public Saml2AuthnRequest CreateAuthnRequest(string authnRequestId, string assertionConsumerServiceUrl)
        {
            var request = new Saml2AuthnRequest
        }
```csharp
    {
        ID = authnRequestId,
        Issuer = _serviceProviderConfiguration.EntityId,
        ForceAuthn = _saml2Configuration.ForceAuth,
        IsPassive = _saml2Configuration.IsPassive,
        Destination =
            _identityProviderConfiguration
                .SingleSignOnService,
        IssuerFormat =
            _identityProviderConfiguration.IssuerFormat,
        IssueInstant = DateTime.UtcNow,
        ProtocolBinding =
            _identityProviderConfiguration.ProtocolBinding
    };

    request.Request.AssertionConsumerServiceURL =
        assertionConsumerServiceUrl;

    var audienceRestrictions = new
        List<ConditionAbstract>(1);
    var audienceRestriction =
        new AudienceRestriction
            {Audience = new List<string>(1)
                -> { _serviceProviderConfiguration.EntityId}};
    audienceRestrictions.Add(audienceRestriction);
    request.Request.Conditions =
        new Conditions {Items =
            -> audienceRestrictions};

    if (_saml2Configuration.AllowCreate.HasValue &&
        _identityProviderConfiguration.NameIdPolicyFormat
            .IsNotNullOrEmpty())
    request.Request.NameIDPolicy = new NameIDPolicy
        {
            AllowCreate =
                _saml2Configuration-AllowCreate,
            Format = _identityProviderConfiguration
                .NameIdPolicyFormat
        };
```
A SAML 2.0 Authentication Middleware for ASP.NET Core

```csharp
if (_saml2Configuration.AuthnContextComparisonType
    .IsNotNullOrEmpty())
    request.RequestRequestedAuthnContext = new RequestedAuthnContext
    {
        Comparison = Enum.Parse<AuthnContextComparisonType>
            (_saml2Configuration
                .AuthnContextComparisonType),
        ComparisonSpecified = true,
        Items = _saml2Configuration
            .AuthnContextComparisonItems
    };
    return request;
}

public Saml2LogoutRequest CreateLogoutRequest(
    string logoutRequestId,
    string sessionIndex,
    string subject)
{
    var request = new Saml2LogoutRequest
    {
        Issuer = _serviceProviderConfiguration.EntityId,
        Destination = _identityProviderConfiguration
            .SingleSignOutService,
        Reason = Saml2Constants.Reasons.User,
        SubjectToLogOut = new NameID()
    };
    request.RequestID = logoutRequestId;

    if (sessionIndex.IsNotNullOrEmpty())
        request.SessionIndex = sessionIndex;

    if (subject.IsNotNullOrEmpty())
        request.SubjectToLogOut.Value = subject;

    return request;
```
public Saml2LogoutResponse CreateLogoutResponse(
    string statusCode,
    string inResponseTo)
{
    return new Saml2LogoutResponse
    {
        StatusCode = statusCode,
        Issuer = _serviceProviderConfiguration.EntityId,
        InResponseTo = inResponseTo,
        Destination =
            _identityProviderConfiguration
                .SingleSignOutService
    };
}
M  Saml2ClaimFactory

```csharp
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using Saml2.Authentication.Core.Extensions;

namespace Saml2.Authentication.Core.Factories
{
    internal class Saml2ClaimFactory : ISaml2ClaimFactory
    {
        public IList<Claim> Create(Saml2Assertion assertion)
        {
            var claims = new List<Claim>();
            if (!string.IsNullOrEmpty(assertion.Subject.Value))
            {
                claims.Add(new Claim(
                    Saml2ClaimTypes.Subject,
                    assertion.Subject.Value));
                claims.Add(new Claim(
                    Saml2ClaimTypes.Name,
                    assertion.Subject.Value));
                claims.Add(new Claim(
                    Saml2ClaimTypes.NameIdentifier,
                    assertion.Subject.Value));
            }

            claims.Add(assertion.SessionIndex.IsNotNullOrEmpty() ?
                new Claim(
                    Saml2ClaimTypes.SessionIndex,
                    assertion.SessionIndex) :
                new Claim(Saml2ClaimTypes.SessionIndex,
                    assertion.Id));
        }
    }
}```
A SAML 2.0 Authentication Middleware for ASP.NET Core

```csharp
claims.AddRange(assertion.Attributes.Select(attribute =>
    new Claim(attribute.Name,
        attribute.AttributeValue[0].ToString()));

return claims;
```

148

namespace Saml2.Authentication.Core.Providers
{
    internal class CertificateProvider : ICertificateProvider
    {
        private readonly SigningCertificate _signingCertificate;

        public CertificateProvider(SigningCertificate signingCertificate)
        {
            _signingCertificate = signingCertificate;
        }

        public SigningCertificate GetCertificate()
        {
            return _signingCertificate;
        }
    }

    public class SigningCertificate
    {
        public SigningCertificate(
            X509Certificate2 identityProvider,
            X509Certificate2 serviceProvider)
        {
            IdentityProvider = identityProvider;
            ServiceProvider = serviceProvider;
        }

        public X509Certificate2 IdentityProvider { get; }
        public X509Certificate2 ServiceProvider { get; }
    }
}
```csharp
33 }
34 }
```
namespace Saml2.Authentication.Core.Bindings
{
    /// <inheritdoc cref="HttpSoapBinding" />
    /// <summary>
    /// Implementation of the artifact over HTTP SOAP
    /// binding.
    /// </summary>
    internal class HttpArtifactBinding : HttpSoapBinding, IHttpArtifactBinding
    {
        public bool IsValid(HttpRequest request)
        {
            return request?.Method == HttpMethods.Get &&
                   !string.IsNullOrEmpty(request?.Query["SAMLart"]);
        }

        public string GetArtifact(HttpRequest request)
        {
            return request?.Query["SAMLart"];  
        }

        public string GetRelayState(HttpRequest request)
        {
            var encodedRelayState =
                request?.Query["RelayState"].ToString();
return encodedRelayState.DeflateDecompress();
}

/// <inheritdoc />
/// <summary>
/// Resolves an artifact.
/// </summary>
/// <returns>A stream containing the artifact response from the IdP</returns>
public Stream ResolveArtifact(
    string artifact,
    string artifactResolveEndpoint,
    string serviceProviderId,
    X509Certificate2 cert)
{
    if (artifactResolveEndpoint == null)
        throw new InvalidOperationException(
            "Received artifact from unknown IDP.");

    var resolve = new Saml2ArtifactResolve
    {
        Issuer = serviceProviderId,
        Artifact = artifact
    };

    var doc = resolve.GetXml();
    if (doc.FirstChild is XmlDeclaration)
        doc.RemoveChild(doc.FirstChild);

    XmlSignatureUtils.SignDocument(doc, resolve.ID, cert);

    var artifactResolveString = doc.OuterXml;
    return GetResponse(artifactResolveEndpoint, artifactResolveString);
}
namespace Saml2.Authentication.Core.Bindings
{
    /// <summary>
    /// Handles the creation of redirect locations when using the HTTP redirect binding, which is outlined in [SAMLBind] section 3.4.
    /// </summary>
    internal class HttpRedirectBinding : IHttpRedirectBinding
    {
        private const string SamlResponseQueryKey = "SamlResponse";
        private const string SamlRequestQueryKey = "SAMLRequest";
        private const string SamlRelayStateQueryKey = "RelayState";
        private readonly ISignatureProviderFactory _signatureProviderFactory;
        public HttpRedirectBinding(ISignatureProviderFactory signatureProviderFactory)
```csharp
{  
    _signatureProviderFactory = signatureProviderFactory;
}

public bool IsValid(HttpRequest request)  
{
    if (request == null)
    return false;

    if (request.Method == HttpMethods.Get)
    return
    request.Query.ContainsKey(SamlRequestQueryKey)
    ||
    request.Query.ContainsKey(SamlResponseQueryKey);

    if (request.Method != HttpMethods.Post)
    return false;

    var form = request.Form;
    return form != null && form.ContainsKey(SamlResponseQueryKey);
}

public bool IsLogoutRequest(HttpRequest request)  
{
    if (request == null)
    return false;

    if (request.Method == HttpMethods.Get)
    return request.Query.ContainsKey(SamlRequestQueryKey);

    if (request.Method != HttpMethods.Post)
    return false;

    var form = request.Form;
    return form != null &&
    form.ContainsKey(SamlRequestQueryKey);
}
```
```csharp
public Saml2Response GetResponse(HttpRequest request)
{
    if (request.Method == HttpMethods.Get)
        return new Saml2Response
        {
            Response = request.Query[SamlResponseQueryKey],
            RelayState = request.Query[SamlRelayStateQueryKey].ToString()?.DeflateDecompress()
        };

    if (request.Method != HttpMethods.Post)
        return null;

    var form = request.Form;
    if (form == null)
        return null;

    return new Saml2Response
    {
        Response = form[SamlResponseQueryKey],
        RelayState = form[SamlRelayStateQueryKey].ToString() ?? .DeflateDecompress()
    };
}

public string GetCompressedRelayState(HttpRequest request)
{
    if (request.Method ==HttpMethods.Get)
        return request.Query[SamlRelayStateQueryKey].ToString();

    if (request.Method != HttpMethods.Post)
        return null;

    var form = request.Form;
```
return form?[SamlRelayStateQueryKey].ToString();
}

public string BuildAuthnRequestUrl(
    Saml2AuthnRequest saml2AuthnRequest,
    AsymmetricAlgorithm signingKey,
    string hashingAlgorithm,
    string relayState)
{
    var request = saml2AuthnRequest.GetXml().OuterXml;
    return BuildRequestUrl(
        signingKey,
        hashingAlgorithm,
        relayState,
        request,
        saml2AuthnRequest.Destination);
}

public string BuildLogoutRequestUrl(
    Saml2LogoutRequest saml2LogoutRequest,
    AsymmetricAlgorithm signingKey,
    string hashingAlgorithm,
    string relayState)
{
    var request = saml2LogoutRequest.GetXml().OuterXml;
    return BuildRequestUrl(
        signingKey,
        hashingAlgorithm,
        relayState,
        request,
        saml2LogoutRequest.Destination);
}

public string BuildLogoutResponseUrl(
    dk.nita.saml20.Saml2LogoutResponse logoutResponse,
    AsymmetricAlgorithm signingKey,
    string hashingAlgorithm,
    string relayState)
{
var response = logoutResponse.GetXml().OuterXml;
return BuildRequestUrl(
    signingKey,
    hashingAlgorithm,
    relayState,
    response,
    logoutResponse.Destination);
}

public string GetLogoutResponseMessage(Uri uri, 
    AsymmetricAlgorithm key)
{
    var parser = new HttpRedirectBindingParser(uri);
    if (key == null)
        throw new ArgumentNullException(nameof(key));

    if (!parser.IsSigned)
        throw new InvalidOperationException(
            "Query is not signed, so there is no 
            signature to verify."
        );

    // Validates the signature using the public part of 
    // the asymmetric key given as parameter.
    var signatureProvider =
        _signatureProviderFactory
            .CreateFromAlgorithmUri(
                key.GetType(),
                parser.SignatureAlgorithm);
    if (!signatureProvider.VerifySignature(
        key,
        Encoding.UTF8.GetBytes(
            parser.SignedQuery),
        parser.DecodeSignature()))
        throw new InvalidOperationException(
            "Logout request signature verification failed");

    return parser.Message;
}
public Saml2LogoutResponse GetLogoutResponse(
    Uri uri,
    AsymmetricAlgorithm key)
{
    var response = new Saml2LogoutResponse();
    var parser = new HttpRedirectBindingParser(uri);
    if (key == null)
        throw new ArgumentNullException(nameof(key));

    response.OriginalLogoutRequest = parser.LogoutRequest;

    if (!parser.IsSigned)

    // Validates the signature using the public part of the asymmetric key given as parameter.
    var signatureProvider =
        _signatureProviderFactory
            .CreateFromAlgorithmUri(  // signatureProviderFactory
            key.GetType(), parser.SignatureAlgorithm);

    if (!signatureProvider.VerifySignature(  // signatureProvider.VerifySignature
        key,
            parser.SignedQuery),
            parser.DecodeSignature()))
            Saml2Constants.StatusCodes.Success;

    return response;
}

/// <summary>
/// If an asymmetric key has been specified, sign the request.
/// </summary>
private void AddSignature(
    StringBuilder result,
    AsymmetricAlgorithm signingKey,
    ShaHashingAlgorithm hashingAlgorithm)
{
    if (signingKey == null)
        return;

    result.Append(
        string.Format("&{0}=",
            HttpRedirectBindingConstants.SigAlg));

    var signingProvider =
        _signatureProviderFactory
            .CreateFromAlgorithmName(
                signingKey.GetType(), hashingAlgorithm);

    var urlEncoded =
        signingProvider.SignatureUri.UrlEncode();
    result.Append(urlEncoded.UpperCaseUrlEncode());

    // Calculate the signature of the URL as described in
    // [SAMLBind] section 3.4.4.1.
    var signature = signingProvider.SignData(
        signingKey,
        Encoding.UTF8.GetBytes(
            result.ToString()));

    result.AppendFormat(
        "&{0}=", HttpRedirectBindingConstants.Signature);

    result.Append(
        HttpUtility.UrlEncode(
            Convert.ToBase64String(signature)));
}

private string BuildRequestUrl(
    AsymmetricAlgorithm signingKey,
    string hashingAlgorithm,
```csharp
string relayState,
string request,
string destination
{
    var shaHashingAlgorithm = signatureProviderFactory
        .ValidateShaHashingAlgorithm(shaHashingAlgorithm);

    // Check if the key is of a supported type.
    // [SAMLBind] sect. 3.4.4.1 specifies this.
    if (!(signingKey is RSA || signingKey is DSA ||
        signingKey == null))
        throw new ArgumentException(
            "Signing key must be an instance of either RSA or DSA.");

    var result = new StringBuilder();
    result.AddMessageParameter(request, null);
    result.AddRelayState(request, relayState);
    AddSignature(result, signingKey, shaHashingAlgorithm);

    return $"{destination}?{result}";
}
```
namespace Saml2.Authentication.Core.Extensions
{
    public static class Saml2StringBuilderExtensions
    {
        /// <summary>
        /// If the RelayState property has been set, this method adds it to the query string.
        /// </summary>
        /// <param name="result"></param>
        /// <param name="relayState"></param>
        /// <param name="request"></param>
        public static void AddRelayState(
            this StringBuilder result,
            string request,
            string relayState)
        {
            if (relayState == null)
                return;

            result.Append("&RelayState=");
            // Encode the relay state if we're building a request. Otherwise, append unmodified.
            result.Append(request != null
                ? relayState.DeflateEncode().UrlEncode() : relayState);
        }

        /// <summary>
        /// Depending on which one is specified, this method adds the SAMLRequest or SAMLResponse parameter to the URL query.
        /// </summary>
        /// <param name="result"></param>
        /// <param name="request"></param>
        /// <param name="relayState"></param>
        public static void AddSamlParameters(
            this StringBuilder result,
            string request,
            string relayState)
        {
public static void AddMessageParameter(
    this StringBuilder result,
    string request,
    string response)
{
    if (!(response == null || request == null))
        throw new Exception(
            "Request or Response property MUST be set.");

    string value;
    if (request != null)
    {
        result.AppendFormat(
            "{0}=", HttpRedirectBindingConstants.SamlRequest);
        value = request;
    }
    else
    {
        result.AppendFormat(
            "{0}=", HttpRedirectBindingConstants.SamlResponse);
        value = response;
    }

    var encoded = value.DeflateEncode();
    var urlEncoded = encoded.UrlEncode();
    result.Append(urlEncoded.UpperCaseUrlEncode());
}

public static string TrimSpecialCharacters(this string str)
{
    var sb = new StringBuilder();
    foreach (var c in str)
    {
        if (c >= '0' && c <= '9'
            || c >= 'A' && c <= 'Z'
            || c >= 'a' && c <= 'z'
            || c == '.')
        {
            sb.Append(c);
        }
    }
    return sb.ToString();
}
|| c == ' _ ')
    sb.Append(c);
    
    return sb.ToString();

}
namespace Saml2.Authentication.Core.Extensions
{
    public static class Saml2StringExtensions
    {
        public static bool IsNotNullOrEmpty(this string value)
        {
            return !string.IsNullOrEmpty(value);
        }

        public static bool IsNullOrEmpty(this string value)
        {
            return string.IsNullOrEmpty(value);
        }

        public static string UrlEncode(this string value)
        {
            return HttpUtility.UrlEncode(value);
        }

        public static string UrlDecode(this string value)
        {
            return HttpUtility.UrlDecode(value);
        }

        /// <summary>
        /// Uses DEFLATE compression to compress the input value. Returns the result as a Base64 encoded string.
        /// </summary>
        public static string Compress(this string value)
        {
            using (var memoryStream = new MemoryStream())
            {
                using (var compressedStream = new GZipStream(memoryStream, CompressionMode.Compress, true))
                {
                    using (var writer = new StreamWriter(compressedStream))
                    {
                        writer.Write(value);
                    }
                }

                return Convert.ToBase64String(memoryStream.ToArray());
            }
        }
    }
}

165
```csharp
public static string DeflateEncode(this string value)
{
    var memoryStream = new MemoryStream();
    using (var writer = new StreamWriter(
        new DeflateStream(memoryStream, CompressionMode.Compress, true),
        new UTF8Encoding(false)))
    {
        writer.Write(value);
        writer.Close();

        return Convert.ToBase64String(
            memoryStream.GetBuffer(), 0, (int)
            memoryStream.Length,
            Base64FormattingOptions.None);
    }
}

/// <summary>
/// Take a Base64-encoded string, decompress the
/// result using the DEFLATE algorithm and return the
/// resulting
/// string.
/// </summary>
public static string DeflateDecompress(this string value)
{
    var encoded = Convert.FromBase64String(value);
    var memoryStream = new MemoryStream(encoded);

    var result = new StringBuilder();
    using (var stream = new DeflateStream(
        memoryStream, CompressionMode.Decompress))
    {
        var testStream = new StreamReader(
            new BufferedStream(stream), Encoding.UTF8);
        // It seems we need to "peek" on the StreamReader
to get it started. If we don't do this, the
        first call to
```
// ReadToEnd() will return string.empty.
    var peek = testStream.Peek();
    result.Append(testStream.ReadToEnd());

    stream.Close();
}

return result.ToString();

    }
SAML 2.0 Authentication Middleware for ASP.NET Core

Identity provider metadata

```xml
<md:EntityDescriptor
  xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  entityID="https://saml2login.my.salesforce.com"
  validUntil="2028-05-05T09:46:30.439Z">
  <md:IDPSSODescriptor
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <md:KeyDescriptor use="signing">
      <ds:KeyInfo>
        <ds:X509Data>
          <ds:X509Certificate>
            MIIErDCCA5SgAwIBAgIOAWMp4yXaAAAAAcFp9YwDQYJKoZIhvcNAQELBQAawgZAx
            KDAmbNgNVBAMMH1NlbGZTaWduZWRDZXJ0XzA0TWF5MjAxOF8wNjQyMjAxOCAwEAAaO
            IAQAwf0wHQYDVQR0BBYEFPiQUh1nbrZhKxSjg5z2LiYiGMVqMa8GA1UdEwEB
          </ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </md:KeyDescriptor>
  </md:IDPSSODescriptor>
</md:EntityDescriptor>
```
T  Base64 encoded SAMLResponse
Decoded SAMLResponse

```xml
<?xml version="1.0" encoding="UTF-8"?>
<samlp:Response
   xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
   Destination="https://localhost:44344/Saml2/AssertionConsumerService"
   ID="_a2f0a6ec822e51674785ac43a63c5eab1525535780073"
   InResponseTo="b3dc5923fff09491d9f66ec6000c"
   <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
      Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">
      https://saml2login.my.salesforce.com</saml:Issuer>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference
            URI="#_a2f0a6ec822e51674785ac43a63c5eab1525535780073">
            <ds:Transforms>
               <ds:Transform
                  Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </ds:Transforms>
            <ds:Transform
               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            <ec:InclusiveNamespaces
               xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"
               PrefixList="ds saml samlp xs xsi"/>
         </ds:Reference>
      </ds:SignatureMethod>
   </ds:SignedInfo>
</samlp:Response>
```
A SAML 2.0 Authentication Middleware for ASP.NET Core

<ds:DigestValue>
rMTl5dD0RsJGVaXbhWyoYp49HDc=
</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>heKcp+99s0dYWohiJXu1WYZ3Blh4NNGf1Rx/hHuM/kOmY6vsjeaENO3m64PwBDhqVVxfzkJMf6Eh
e71E2ALX/cVaDT416HLzkvcf2WEO9NNOX5hPV/YT4AL6RYRL/
96AusIjFSSletJ8XMGrmu4s0Wnpmg3P3rTBXNj4WE/+KTMm0jGp
cGhYgLaMc61h1hSCTyfXLBsz8RMLLyQIG1Fajarj07PWlSCiov06q
x69okwCwl1N1n1LyPwsJuS0tOdiPzyPT3zi6FUzD/TnBks6ShT
2LY1ZLGLqFoskhOD/+PBUeq7fby62tcbcbJsM110VXXuawgmgii
/sobTEN11JFC==</ds:SignatureValue>
</ds:KeyInfo>
</ds:X509Certificate>

MITER/DCCA5sGwAbIBAgIOAWMp4yXaaAAAAAcFp9YWdQYJKoZIhvcNQELBQAwgZAx
KoA7mgVBAMMH11bGzTaWduZWRDZXJ0xZIAXO5Fw5MAxOF8wNjQyMjIxGDAWBgNV
BAoMDzAwRDRfMDA0MDAYQ0NzcjEXMBUGwW2Z2vN8mb3JjZ5j20xFeJAU
BgNVBAAcMDVhNhiBGcmFvY21zY28xLzAcAJBGVBAgMAkNBQwCgYDVQQE0xNVU0Ec
HhcNMTgwNTA0MDY0MjIyWhcNMTgwNTA0MDAwMDAwWjCBkDEoMCYgAUEwU2FS
Z1NpZ251ZENcIcnRMDXNYxkYDE4XzA2NDIyMjEYMBYGA1ECwqMDEMXiw
MDA0MDAYQ0NzcjEXMBUGwW2Z2vN8mb3JjZ5j20xFeJAU
BGNVBAAcMDVhNhiBGcmFvY21zY28xLzAcAJBGVBAgMAkNBQwCgYDVQQE0xNVU0Ec
HhcNMTgwNTA0MDY0MjIyWhcNMTgwNTA0MDAwMDAwWjCBkDEoMCYgAUEwU2FS
Z1NpZ251ZENcIcnRMDXNYxkYDE4XzA2NDIyMjEYMBYGA1ECwqMDEMXiw
MDA0MDAYQ0NzcjEXMBUGwW2Z2vN8mb3JjZ5j20xFeJAU
BGNVBAAcMDVhNhiBGcmFvY21zY28xLzAcAJBGVBAgMAkNBQwCgYDVQQE0xNVU0Ec
HhcNMTgwNTA0MDY0MjIyWhcNMTgwNTA0MDAwMDAwWjCBkDEoMCYgAUEwU2FS
Z1NpZ251ZENcIcnRMDXNYxkYDE4XzA2NDIyMjEYMBYGA1ECwqMDEMXiw
MDA0MDAYQ0NzcjEXMBUGwW2Z2vN8mb3JjZ5j20xFeJAU
BGNVBAAcMDVhNhiBGcmFvY21zY28xLzAcAJBGVBAgMAkNBQwCgYDVQQE0xNVU0Ec
HhcNMTgwNTA0MDY0MjIyWhcNMTgwNTA0MDAwMDAwWjCBkDEoMCYgAUEwU2FS
Z1NpZ251ZENcIcnRMDXNYxkYDE4XzA2NDIyMjEYMBYGA1ECwqMDEMXiw
0Juc0f1SfNjTAIDTgonPNf2PoNUmgLgXcan6S38bZBCRWfdC4f2Gohfi0uScD0+4pMF9yv/zm21dh/j0cFe2hknbCu2zXeWcnFgE31kE5ZsSj</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</ds:Signature>
<samlp:Status>
</samlp:Status>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <ds:Reference URI="#_f580b191d80f081b766398a5738360aa1525535780073">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <ec:InclusiveNamespaces xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="ds saml xs xsi"/>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>YKJC6DmbJUociUDISdhW9R08=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
  </ds:Signature>
</saml:Assertion>
A SAML 2.0 Authentication Middleware for ASP.NET Core

<ds:SignatureValue>
H1y01GGSys3Vdpj71Nn0P+ZPn90wN/
vhPA4wZse5ItWdXF+V4mMkR8k+RXBHYw3wWzfuY9Z
JdyX7nwZwI9jTMWEvRREkBSQ4S1vCFa89pNMZs7qi0fLfj
Rfz9B90LEfDeNabv3a67ouVsC8Ss4KABkPNz12NydcSbq
Sv2AKEUKPueGzRBKHKoDV7LVwycz/z/uNkRE1QPZu926
spRy4U4SBdj1bS1bhB9e7XePE7IIsQF9ZC7ASC7SVoG
ibRsRLk8w1Zzn+y1fK9U9K2YeN79Crh5XJZ4geo7EtHK
v5R4Ae3+0uB/u/sCQAALspNmY/5koy20fwSvArpwww=
</ds:SignatureValue>

<ds:KeyInfo>
<ds:X509Data>
<ds:X509Certificate>
MIIErDCCA5SgAwIBAgIOAWMp4yxAAAACFp9YWdQYJK0ZIhvcNAQELBQAwgZAx
KDAmbgNVBAAMBMMH1bG1zaXZlZ29yaWVydGhyZSw0Yj4wczIwYjAV
BAeMDaAwMDAwMDAyQ3YzcjEXMBUGA1UECgwOU2FsZXNmb3JjZS5jb20xFjAW
BGVBAsMDVBNhabBGcmFuY2luZ2FteWNoZWNrZ3JlYXRpb25z
ZLlhbXxhZGRyb2t1aGlvYnJhbmRvZ3JlYXRpb25z
</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>

176
A SAML 2.0 Authentication Middleware for ASP.NET Core

```xml
<ds:X509Certificate>
<ds:X509Data>
<ds:Signature>
<saml:KeyInfo>
<saml:Signature>
<saml:Subject>
<saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified">
jkm0xxxxxxxxx.com</saml:NameID>
<saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
<saml:SubjectConfirmationData InResponseTo="b3dc5923fff09491d9f66ec6000c" NotOnOrAfter="2018-05-05T16:01:20.073Z" Recipient="https://localhost:44344/Saml2/AssertionConsumerService"/>
</saml:SubjectConfirmationData>
<saml:AudienceRestriction>
</saml:AudienceRestriction>
</saml:Conditions>
<saml:AuthnContext>
</saml:AuthnContext>
</saml:AuthnStatement>
<saml:AttributeStatement>
<saml:Attribute Name="userId" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
0051r0000089ZM2</saml:AttributeValue>
</saml:Attribute>
</saml:AttributeStatement>
</saml:NameID>
</saml:SubjectConfirmationData>
</saml:Conditions>
</saml:AuthnStatement>
</saml:Subject>
</ds:Signature>
</ds:X509Data>
</ds:X509Certificate>
```

177
<saml:Attribute>
  <saml:Attribute Name="username"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
    <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="xs:anyType">jkmu@xxxxxxxx.com</saml:AttributeValue>
  </saml:Attribute>
  <saml:Attribute Name="email"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
    <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="xs:anyType">jkmu@xxxxxxxx.com</saml:AttributeValue>
  </saml:Attribute>
  <saml:Attribute Name="is_portal_user"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
    <saml:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:type="xs:anyType">false</saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
</saml:Assertion>
</samlp:Response>
V Class diagram