Универзитет у Београду
Математички факултет

Мастер рад

SSO аутентификациони систем и подршка за SSO у оквиру .NET Core радног оквира

Аутор:
Милица Ђурић

Ментор:
доц. др Милан Банковић

Чланови комисије:

проф. др Саша Малков
доц. др Јелена Граовац

6. септембар 2021.

Универзитет у Београду

Сажетак

Математички факултет
Катедра за Рачунарство и информатику

SSO аутентификациони систем и подршка за SSO у оквиру .NET Core радног оквира

Милица Ђурић

У овом раду дат је приказ система аутентификације *SSO* (енгл. *Single Sign On*) који омогућава кориснику приступ различитим апликацијама коришћењем само једног корисничког имена и лозинке. Током времена креирани су различити протоколи који омогућавају овај тип аутентификације, а у овом раду дат је преглед и поређење три најчешће коришћена протокола. То су *SAML*, *OAuth* и *OpenID Connect*. SAML је протокол заснован на XML-у, и иако је најстарији и најкомплекснији, и даље је широко коришћен. Протокол OAuth је, са друге стране, веома једноставан и пре свега је намењен ауторизацији и ограничавању приступа различитим подацима, али је дуги низ година коришћен и за аутентификацију. Како би се стандардизовао и начин за постизање аутентификације код протокола OAuth, креирана је његова надоградња, протокол OpenID Connect. Да би се стекао шири увид у однос ова три протокола, њихово поређење је дато и кроз имплементацију у оквиру једног од најпознатијих и најчешће коришћених радних оквира .Net Core. Овај радни оквир има уграђену подршку за протоколе OAuth и OpenID Connect, што употребу ових протокола чини веома једноставном. Са друге стране, радни оквир .Net Core нема уграђену подршку нити званичну библиотеку за протокол SAML, те је уз овај рад имплементирана и једна таква библиотека која омогућава слање захтева и обраду одговора приликом аутентификације коришћењем протокола SAML.

Садржај

Глава 1

Увод

Упоредо са убрзаним и перманентним развојем веб технологија, расте и потреба корисника за различитим веб апликацијама које ће користити било за свакодневно обављање посла, било за употпуњавање слободног времена. Велики број таквих апликација захтева прављење налога ради персонализовања корисничког искуства. При сваком креирању новог налога, корисник мора да унесе корисничко име и лозинку коју ће користити убудуће за доказивање свог идентитета. Што је већи број апликација којима корисник свакодневно приступа, већи је и број лозинки које он треба да памти и уноси. Наравно, корисник може поставити исту лозинку на различитим апликацијама. Овим приступом корисник себе излаже ризику да ће злонамерни софтвер који успе да се домогне једне његове лозинке, моћи да приступи и другим апликацијама користећи исту ту лозинку и тиме добити приступ многим информацијама. Уколико се корисник одлучи да на свим апликацијама користи различиту лозинку, он може да прибегне коришћењу лозинки које су лаке за памћење, али су недовољно јаке да га заштите од напада. Овим приступом корисник олакшава злонамерном софтверу да се домогне његове лозинке и приступи многим приватним информацијама. Заштита података је посебно важна уколико се ради о апликацијама које се користе у пословне сврхе.

Како би се кориснику обезбедио једноставан и сигуран начин приступа апликацијама, уведен је аутентификациони систем звани *SSO* (енгл. *Single Sign-On*). Овај систем омогућава корисницима приступ различитим апликацијама коришћењем само једног корисничког имена и лозинке. Потребно је да се корисник само једном аутентификује како би користио све апликације које користе овај аутентификациони систем.

Овакав приступ би се могао упоредити са неким примерима из живота. На пример, уколико особа жели да присуствује неком фестивалу филмова за који се купује једна карта која омогућава приступ свим филмовима, он неће морати да показује карту при сваком улазу у салу у којој се пушта одређени филм. Довољно ће бити показивање карте само једном, при улазу на сам фестивал.

SSO омогућава централизовано управљање налозима чиме се комплексност аутентификације и њене сигурности пребацује на једно место. Централизовано управљање налозима олакшава решавање проблема који се могу јавити нпр. при заборављању лозинке или при неовлашћеном упаду на налог. Овакви системи

форсирају коришћење јаких лозинки којих се злонамерни софтвер тешко може домоћи.

У овом раду дат је приказ аутентификационог система SSO и његове подршке у оквиру радног оквира .NET Core. У глави 2 уведен је сам појам аутентификације, наведене су карактеристике SSO-а и различите врсте протокола који се користе за његову имплементацију. У главама 3, 4, 5 описан је начин рада протокола OAuth2, OpenID Connect и SAML, њихове карактеристике, предности и мане. У глави 6 описана је подршка за SSO у оквиру радног оквира .NET Core и дати су примери коришћења уграђених библиотека. Како овај радни оквир нема уграђену подршку за протокол SAML, уз овај рад креирана је и библиотека која имплементира овај протокол и у глави 7 описан је начин рада креиране библиотеке. Глава 8 представља закључак овог рада и у њој се сумирају предности и мане оваквог аутентификационог система и дати су предлози за побољшање библиотеке која је направљена као додатак овом раду.

Глава 2

Аутентификација

Безбедно чување личних података корисника и персонализовање корисничког искуства важни су аспекти модерних веб технологија. Велики развој веба довео је до раста броја апликација и података који се налазе на њему. Постало је важно заштити податке од неовлашћеног приступа у многим апликацијама као што су сервиси за електронско пословање, апликације банака или електронска комуникација путем имејла. Безбедност личних података као и угођај коришћења веб апликације се повећава уколико су садржај страница и могућности веб апликације прилагођени конкретном кориснику. Један од најзначајнијих корака ка сигурнијем и угоднијем корисничком искуству је креирање налога на веб апликацији којем ће моћи да приступи само корисник који је креирао тај налог. На тај начин, кориснику се омогућава и чување приватних информација које су доступне само њему или корисницима којима он дозволи приступ.

Процес верфикације идентитета корисника приликом пријављивања на систем назива се *аутентификација*, док је онемогућавање приступа неовлашћеним корисницима *ауторизација* [2].

2.1 Типови аутентификације

Приликом процеса пријављивања корисника на систем, врши се идентификација корисника захтевањем информација о кориснику које ће систем користити у процесу верификације идентитета.

Све типове аутентификације можемо поделити у четири основне групе према информацијама које корисник доставља приликом процеса идентификације. [2]:

1. аутентификација на основу тога *шта корисник зна*

   Ово је група са најчешће коришћеним типовима аутентификације. У ову групу спадају лозинке, PIN лозинке[1] и сигурносна питања.

2. аутентификација на основу тога *шта корисник има*

   У ову групу спада аутентификација коришћењем личног идентификационог документа (попут личне карте и пасоша) или сигурносног адаптера (попут случајног генератора бројева).

[1] Скраћеница PIN означава лични идентификациони број (енгл. *personal identification number*).

3. аутентификација на основу тога *ко је корисник*

   Ова група је група биометријске аутентификације. У њу спадају аутентификација коришћењем отиска прста, гласа, препознавања лица или скенирања зенице.

4. аутентификација на основу тога *где се налази корисник*

   У ову групу спадају аутентификације које користе локацију корисника. Може се користити корисникова IP адреса ili GPS геолокација.

За потребе овог рада посебно је интересантна аутентификација коришћењем лозинке.

2.2 Аутентификација коришћењем лозинке

Аутентификација коришћењем лозинке представља начин пријављивања на систем у коме корисник као доказ свог идентитета пружа лозинку коју је поставио у тренутку креирања налога. Ово је један од најчешће коришћених типова аутентификације.

У оваквом типу аутентификације, поред саме лозинке, корисника јединствено одређује и његово корисничко име или имејл адреса које је изабрао у тренутку креирања налога.

Како би се заштитили од неовлашћеног коришћења налога, корисници би требало да користе лозинке које се дефинишу као *јаке*. Јака лозинка је лозинка коју тешко открива и човек и машина. Овакве лозинке су обично комбинација великих и малих слова, бројева и симбола. Истраживање Дигитал Гардијана (енгл. *Digital Guardian*) које је извршено 2020. године и у којем је учествовало 1000 људи показује да 55.8% испитаника користи јаке лозинке попут "Ilovef00tball!", 37.8% испитаника користи полујаке лозинке попут "Football1", а 6.5% испитаника користи лозинке које су веома једноставне (укључују честе речи или фразе) попут "football" [10].

Велики број веб апликација форсира креирање јаких лозинки. Овакве веб апликације не допуштају креирање налога све док се не изабере лозинка која је довољно јака према њиховим стандардима. Насупрот овоме, постоје и апликације које допуштају креирање било каквих лозинки. На оваквим апликацијама, иако корисници имају свест о томе да је јако битно користити јаку лозинку, често прибегну коришћењу једноставних лозинки које је лако запамтити, али и лако открити. Десет најгорих и најчешће коришћених лозинки за 2018. и 2019. годину према подацима о 5 милиона лозинки које су процуреле на интернету представљене су табелом 2.1.

Поред тога што лозинка треба да буде јака како бисмо били сигурни да је наш налог безбедан, она не би требало ни да се понавља. Коришћење исте лозинке на више налога олакшава злонамерном софтверу приступ већем броју

Табела 2.1: Десет најгорих и најчешће коришћених лозинки за 2018. и 2019. годину [15][16]

Ранг	Лозинка 2018. године	Лозинка 2019. године
1	123456	123456
2	password	123456789
3	123456789	qwerty
4	12345678	password
5	12345	1234567
6	111111	12345678
7	1234567	12345
8	sunshine	iloveyou
9	qwerty	111111
10	iloveyou	123123

информација. Колико је важно не понављати лозинке може се показати на следећем примеру. Злонамеран софтвер може бити једноставна веб апликација која форсира креирање налога како би приказала неке информације које корисника занимају. Приликом креирања налога, овај злонамерни софтвер чува унете лозинке на свом систему такве какве јесу, без хеш вредности. Све лозинке које су прошле кроз систем су изложене и злонамерни софтвер може пробати да приступи другим налозима користећи изложене лозинке. На тај начин, сви корисници који понављају своје лозинке могу бити оштећени.

Компанија Гугл (енгл. *Google*) у сарадњи са компанијом Харис Пол (енгл. *Harris Poll*) је у 2019. години извршила испитивање о схватању и понашању људи кад је у питању онлајн сигурност. У ово испитивање било је укључено 3000 људи из Сједињених Америчких Држава и показано је да многи корисници веб апликација користе исту лозинку на многим својим налозима. Резултати испитивања приказани су на слици 2.1.

Слика 2.1: Резултати истраживања понављања лозинке [12]

Овакви резултати нису зачуђујући с обзиром на то да корисници на дневном нивоу користе велики број апликација. Према истраживању компаније Дашлејн (енгл. *Dashlane*) које је извршено 2015. године и у којем је учествовало 20000

људи, просечан интернет корисник има 90 налога везаних за једну имејл адресу [10]. Памћење јаких лозинки за сваки налог постаје теже како број апликација расте.

Аутентификација коришћењем лозинке се може поделити у два типа по начину на који систем чува лозинке:

1. *локална аутентификација* - лозинка се чува у оквиру система
2. *аутентификација коришћењем добављача* - лозинка се чува на страни добављача

2.2.1 Локална аутентификација

Приликом креирања налога, велики број веб апликација чува у својој бази података изабрано корисничко име и лозинку[2]. Процес аутентификације се у овом случају своди на неколико корака:

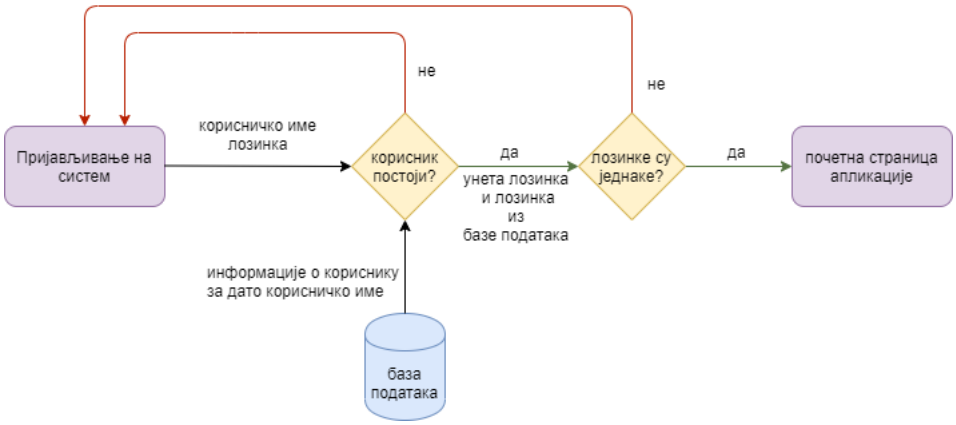
1. корисник отвара аутентификациону страну апликације на којој је приказана форма за пријављивање
2. *идентификација* - корисник прилаже своје корисничко име и лозинку као доказ идентитета
3. *верификација идентитета* - за дато корисничко име, апликација упоређује приложену лозинку и лозинку из базе података
4. у зависности од резултата верификације:
   (а) уколико је верификација успешно прошла, креира се сесија и корисник се пребацује на почетну страну апликације
   (б) уколико је верификација није успешно прошла, корисник остаје на страници за пријављивање са приказаном грешком аутентификације

Овакав процес аутентификације може се назвати *локална аутентификација* с обзиром да се лозинке чувају интерно у оквиру апликације. Процес локалне аутентификације приказан је на слици 2.2.

2.2.2 Аутентификација коришћењем добављача

Веб апликација која има могућност пријављивања на систем не мора да чува локално корисничка имена и лозинке. Уместо ње, то може да ради неки други сервис назван *аутентификациони добављач* (енгл. *autentifiaction provider*).

[2]Из сигурносних разлога, не чува се чиста лозинка него *досољена хеш вредност лозинке* (енгл. *password hash salting*).



Слика 2.2: Процес локалне аутентификације

Задатак аутентификационог добављача је да врши верификацију идентитета корисника. На овај начин, сва логика аутентификације пребацује се на страну аутентификационог добављача. Добављач може бити део целокупног система у оквиру кога се апликација извршава, али може бити и сервер који није део система, а којем апликација верује. Процес се своди на неколико корака:

1. корисник отвара аутентификациону страну апликације где је могуће одабрати једног од понуђених аутентификационих добављача путем којих корисник може да се пријави на систем

2. *редирекција ка добављачу* - у зависности од тога којег аутентификационог добављача је корисник изабрао, врши се редирекција корисника на страницу за пријављивање датог добављача

3. *идентификација* - корисник прилаже своје корисничко име и лозинку као доказ идентитета

4. *верификација идентитета* - за дато корисничко име, добављач упоређује приложену лозинку и лозинку из базе података и креира сесију уколико су лозинке једнаке

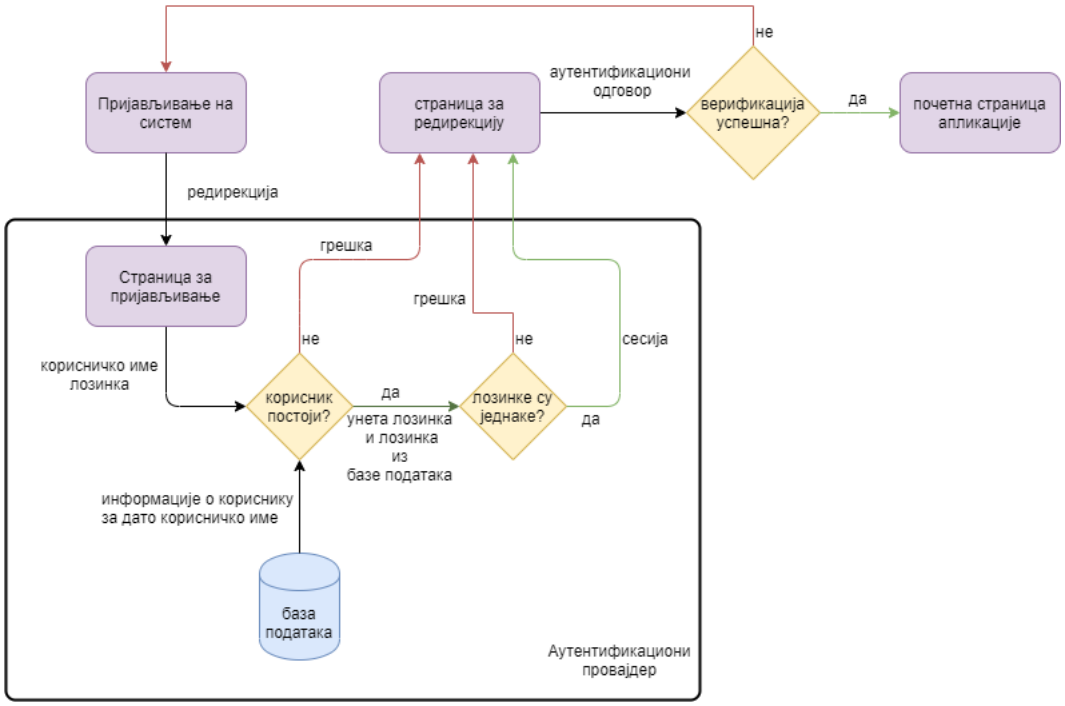
5. *редирекција назад* - добављач врши редирекцију корисника на апликацију која је послала захтев за аутентификацију и шаље креирану сесију уколико је верификација успешно прошла, иначе шаље грешку аутентификације

6. у зависности од резултата редирекције (тј. верификације на страни добављача):

   (а) уколико је резултат редирекције сесија, корисник се пребацује на почетну страну апликације

   (б) уколико је резултат редирекције грешка аутентификације, корисник се враћа на страницу за пријављивање са приказаном грешком аутентификације

Процес аутентификације коришћењем добављача приказан је на слици 2.3.



Слика 2.3: Процес аутентификације коришћењем аутентификационог добављача

Лако је уочљиво да коришћење оваквог начина аутентификације отвара могућност коришћења истог аутентификационог добављача и његове базе од стране више апликација које не морају бити повезане ни са чим другим, осим са самим добављачем.

2.3 SSO аутентификација

Аутентификација коришћењем добављача може омогућити приступ истим налозима од стране више апликација. Овакав аутентификациони приступ у коме корисник може да приступи различитим апликацијама коришћењем само једног корисничког имена и лозинке назива се *Single Sign On*, скраћено *SSO*.

SSO је аутентификациони приступ који пре свега пружа боље корисничко искуство. Потребно је да корисник само једном откуца своје корисничко име и лозинку како би био пријављен на више од једне апликације. Овакав приступ штеди корисниково време што говоре статистички подаци из Медитех (енгл. *Meditech*) здравственог система [2]:

- две секунде се штеди за једно пријављивање
- просечан број пријављивања на недељном нивоу је 70000

- 140000 секунди недељно уштеђено, што значи 2333 минута, тј. 39 сати
- 2022 сати уштеђено на годишњем нивоу, што значи 84 дана

У случају великих предузећа, коришћење SSO-а значајно олакшава свакодневицу запосленог лица с обзиром да у оваквим предузећима постоји велики број апликација које запослени користи на дневном нивоу.

2.3.1 Аутентификација коришћењем друштвених мрежа

Најпознатији тип SSO-а јесте *аутентификација коришћењем друштвених мрежа* (енгл. *Social SSO*). Многе апликације пружају могућност приступа систему коришћењем неког од налога на друштвеним мрежама попут Фејсбука, Твитера или Гугла. У овом случају друштвене мреже представљају аутентификационог добављача.

Процес аутентификације коришћењем друштвених мрежа се може поделити у неколико корака:

1. кориснику се на страни за пријављивање прикажу све друштвене мреже које може да користи како би се пријавио на систем
2. када корисник изабере друштвену мрежу чији налог ће користити да добије приступ жељеној апликацији, он бива преусмерен на страну за пријављивање изабране друштвене мреже (уколико већ није пријављен)
3. након уноса корисничког имена и лозинке, кориснику се обично избацује форма у коме је приказано којим подацима може екстерна апликација приступити и непоходно је да корисник прихвати тражен приступ подацима како би аутентификација на почетној апликацији била успешна
4. након прихватања услова, корисник бива преусмерен на почетну апликацију где се подаци о његовом налогу преузимају од друштвене мреже путем које је извршио аутентификацију

2.3.2 SSO протоколи

Да би се постигао аутентификациони приступ SSO било је потребно поставити јасна правила која зна да разуме и чита и страна која захтева аутентификацију и страна која врши аутентификацију и даје одговор. Скуп правила којима се постиже аутентификациони приступ SSO може се назвати *SSO протоколом*.

Током година развијено је неколико врста SSO протокола. Неки од њих су:

- LDAP
- Kerberos
- Radius

- SAML
- OAuth
- OpenId Connect

Најчешће употребљавани SSO протоколи су SAML, OAuth и OpenId Connect и они ће у следећим поглављима бити детаљно описани.

Глава 3

Протокол SAML

SAML (Security Assertion Markup Language) је SSO протокол у коме се слање захтева и одговора заснива на размени XML докумената. Главни концепт овог протокола јесу *тврђења* (енгл. *assertions*) којима се приказују информације о кориснику које могу бити дељене. Овим протоколом су дефинисана правила која се морају поштовати приликом размене података – тврђења између два сервиса како би размена осетљивих информација о кориснику била извршена на сигуран начин. Иако дефинише строги скуп правила, овај протокол је и флексибилан и прихвата и размену додатних нестандардних информација.

Творац овог протокола јесте заједница *OASIS (Organization for the Advancement of Structured Information Standards)*[1]. Прва верзија овог протокола SAML 1.0 настаје 2001. године, а постаје стандард крајем 2002. Следећа верзија SAML 1.1 постаје OASIS стандард 2003. године, док велике промене доноси трећа верзија стандарда SAML 2.0 која је настала 2005. године. SAML 2.0 доноси нове функционалности попут подршке за енкриптовање XML докумената, могућност иницирања аутентификације од стране сервиса који нема информације о корисничкој лозинки и принцип обједињене одјаве [9]. У даљем раду помињање протокола SAML ће се односити на стандард SAML 2.0.

3.1 Основни појмови

У овом поглављу уводимо неке од основних појмова који се користе у протоколу SAML. Овај протокол дефинише учеснике који учествују у размени XML докумената [13]:

- ***страна која тврди*** (енгл. *asserting party*) - односи се на систем који је задужен за креирање тврђења о кориснику и често се назива и *добављач идентитета* (енгл. *identity provider*)
- ***страна која се ослања*** (енгл. *relying party*) - односи се на систем који користи тврђења о кориснику које је добио од стране која тврди и често се назива и *пружалац услуга* (енгл. *service provider*)

[1]Званична веб страница ове организације је `https://www.oasis-open.org`.

И страна која тврди и страна која се ослања могу започети комуникацију посредством протокола SAML. У односу на то ко је започео комуникацију, дефинишу се појмови [13]:

- *SAML подносилац захтева* (енгл. *SAML requester*) - односи се на систем који започиње неки од дефинисаних процеса у оквиру протокола, односно систем који креира захтев
- *SAML одзивна страна* (енгл. *SAML responder*) - односи се на систем који прима захтев и даје одговор

Поред тврђења, SAML дефинише још три основне компоненте: *протоколе* (енгл. *protocols*), *везе* (енгл. *bindings*) и *профиле* (енгл. *profiles*) [13].

3.2 Тврђење

Страна која се ослања може користити информације које тврђење носи и омогућити приступ ресурсима кориснику на који се тврђење односи уколико је успостављен однос поверења између стране која тврди и стране која се ослања. Иако је успостављен односа поверења, страна која се ослања увек мора проверавати валидност добијеног тврђења с обзиром да злонамерни програми могу пресретнути комуникацију између стране која тврди и стране које се ослања.

Свако тврђење направљено од добављача идентитета може садржати информације које могу бити представљене једном реченицом. Пример: "Субјекат Џон До, са имејл адресом john.doe@example.com, је аутентификован 2. маја 2021. године коришћењем дефинисане лозинке.".

Тврђење се у оквиру XML документа представља елементом *Assertion*. Основни елементи тврђења биће дефинисани у наставку.

3.2.1 Субјекат

Како је SAML протокол који служи за аутентификацију корисника, подаци који се размењују коришћењем XML докумената морају да садрже информације о кориснику који се аутентификује. Ти подаци су део тврђења, а њена главна информација налази се у податку под називом *субјекат* (енгл. *subject*). Субјекат носи информацију о ентитету на који се тврђење односи и који се може аутентификовати. Не морају само крајњи корисници бити субјекат, то може бити и други ентитет попут рачунара. Страна која се ослања може користити информацију коју субјекат носи и потражити у свом систему поменутог субјекта како би му омогућила приступ ресурсима.

У оквиру овог елемента се не налази само сам идентификатор ентитета, већ и услови које је потребно верификовати како би се потврдила валидност самог субјекта.

Субјекат може бити приказан XML елементом *Subject* који изгледа овако:

```
<saml:Subject>
    <saml:NameID>_2jdic2949djskalv843</saml:NameID>
    <saml:SubjectConfirmation
     Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData
            NotBefore="2020-12-30T02:45:56Z"
            NotOnOrAfter="2021-01-01T08:53:48Z"
            Recipient="http://someSp.com/saml">
       </saml:SubjectConfirmationData>
    </saml:SubjectConfirmation>
</saml:Subject>
```

Елемент *NameID* у оквиру елемента *Subject* служи да представи идентификатор субјекта на који се тврђење односи. Елемент *SubjectConfirmation* носи информације које омогућавају потврду повезаности субјекта са страном која тврди. Ако је присутан више од један елемент *SubjectConfirmation*, довољно је да је један од њих задовољен како би субјекат био потврђен. Атрибут *Method* јесте URI који дефинише протокол или метод који треба користити како би се субјекат потврдио. Елемент *SubjectConfirmationData* у оквиру елемента *SubjectConfirmation* дефинише додатне информације које су потребне да се субјекат потврди или дефинише ограничења у оквиру којих је могуће извршити потврду субјекта. Примера ради, ако овај елемент садржи атрибут *NotBefore*, који има вредност неког датума, потврда субјекта се не сме десити пре овог дефинисаног датума, а ако садржи атрибут *NotOnOrAfter*, потврда субјекта се не сме десити у то време или после тог датума. Ако овај елемент садржи атрибут *Recipient*, то може значити да је приложено тврђење упућено само примаоцу дефинисаном овим атрибутом.

3.2.2 Искази

Поред самог субјекта, у оквиру SAML тврђења налазе се и информације о њему представљене коришћењем *исказа* (енгл. *statements*). Стандард дефинише три врсте исказа [1]:

- *аутентификациони искази* - говоре о начину и времену аутентификације субјекта
- *атрибутски искази* - ближе одређују субјекат, дају више информација о њему
- *ауторизациони искази* - дефинишу шта је субјекту дозвољено да ради у оквиру система, односно да ли му је дозвољен или одбијен приступ траженом ресурсу

Аутентификациони исказ

Страна која се ослања може користити аутентификациони исказ како би валидирала постојећу сесију која припада страни која издаје тврђење и креирала

сесију на својој страни. Валидација тврђења може да укључује и проверу да ли је метод коришћен при аутентификацији довољно сигуран.

Аутентификациони исказ може бити приказан XML елементом *AuthnStatement* који изгледа овако:

```
<saml:AuthnStatement
 AuthnInstant="2020-12-30T08:45:12"
 SessionNotOnOrAfter="2021-01-01T09:00:00">
    <saml:AuthnContext>
        <saml:AuthnContextClassRef>
              urn:oasis:names:tc:SAML:2.0:ac:classes:Password
        </saml:AuthnContextClassRef>
        <saml:AuthenticatingAuthority>
              https://someIdp.com
        </saml:AuthenticatingAuthority>
    </saml:AuthnContext>
</saml:AuthnStatement>
```

Атрибут *AuthnInstant* у оквиру елемента *AuthnStatement* представља време када се аутентификација догодила, док атрибут *SessionNotOnOrAfter* у оквиру истог елемента дефинише време после ког се субјекат сматра неаутентификованим, односно време после ког је сесија на страни добављача идентитета прекинута. У оквиру елемента *AuthnStatement* могуће је дефинисати елемент *AuthnContext* који дефинише контекст аутентификације субјекта и који може садржати неколико поделемената. Његов поделемент *AuthnContextClassRef* дефинише аутентификациони метод приказан коришћењем одређене референце на класу, а поделементи *AuthenticatingAuthority* приказују све системе који су учествовали у процесу аутентификације субјекта, изузев система који је креирао тврђење.

Атрибутски исказ

Страна која се ослања може користити атрибутски исказ како би повезала субјекта из тврђења са ентитетом у својој локалној бази. Уколико ентитет не постоји у локалној бази, страна која се ослања може користити послате атрибуте како би га креирала и дала му приступ својим ресурсима.

Атрибутски исказ може бити приказан XML елементом *AttributeStatement* који изгледа овако:

```
<saml:AttributeStatement>
     <saml:Attribute Name="mail">
          <saml:AttributeValue> test@example.com</saml:AttributeValue>
     </saml:Attribute>
</saml:AttributeStatement>
```

Елемент *Attribute* у оквиру елемента *AttributeStatement* дефинише атрибут који се односи на субјекта из тврђења и који ближе одређује самог субјекта.

Његов атрибут *Name* дефинише име атрибута који се односи на субјекат, а његов поделемент *AttributeValue* дефинише вредност атрибута који је дефинисан датим ограничавајућим елементом *Attribute*.

Ауторизациони исказ

Страна која се ослања може користити ауторизациони исказ како би валидирала приступ субјекта одређеним ресурсима. Ауторизациони исказ може бити приказан XML елементом *AuthzDecisionStatement* који изгледа овако:

```
<saml:AuthzDecisionStatement Resource="Printer" Decision="Deny">
    <saml:Action> GET </saml:Action>
</saml:AuthzDecisionStatement>
```

Атрибут *Resource* у оквиру елемента *AuthzDecisionStatement* представља ресурс за који се издаје дата ауторизациона дозвола, а атрибут *Decision* представља одлуку стране која издаје тврђење о дозволи приступа траженом ресурсу. У оквиру елемента *AuthzDecisionStatement* се дефинише и елемент *Action* који представља дозвољене или забрањене акције над траженим ресурсом.

3.2.3 Услови

Тврђење може садржати и *услове* (енгл. *conditions*) који дефинишу ограничења при којима се тврђење може користити. Уколико тврђење садржи и услове, страна која се ослања мора утврдити њихову испуњеност приликом процене валидности тврђења. Тврђење није валидно уколико било који услов дефинисан овим елементом није задовољен. Уколико су сви услови задовољени, тврђење се сматра валидном. Услови могу бити приказани XML елементом *Conditions* који изгледа овако:

```
<saml:Conditions
 NotBefore="2020-12-30T01:01:18Z"
 NotOnOrAfter="2021-01-01T08:35:12Z">
      <saml:AudienceRestriction>
            <saml:Audience> http://someSp.com </saml:Audience>
      </saml:AudienceRestriction>
</saml:Conditions>
```

Атрибут *NotBefore* у оквиру елемента *Conditions* дефинише време пре ког тврђење није валидно, док атрибут *NotOnOrAfter* дефинише време после ког тврђење није валидно. У оквиру елемента *Conditions* потребно је дефинисати и елемент *AudienceRestriction* који представља листу система за које је ово тврђење валидно. Уколико се страна која се ослања и која проверава валидност тврђења не налази на овој листи, она може сматрати ово тврђење невалидним.

3.3 Протокол

Стандард SAML дефинише *протоколе захтева и одговора* који се користе како би се извршила размена тврђења између аутентификационог добављача и пружаоца услуга. Ови протоколи дефинишу структуру и форму захтева и одговора које аутентификациони добављач и пружалац услуга морају да испоштују уколико за размену података користе SAML. Такође, они дефинишу ширину и могућности примене стандарда SAML.

Примери неких од протокола које стандард SAML дефинише су [13]:

- *Протокол аутентификационог захтева* (енгл. *Authentication Request Protocol*)

  Дефинише скуп правила према којима пружалац услуга захтева и добија тврђења која садрже аутентификационе и атрибутске исказе. Овај протокол се користи приликом започињања аутентификације на страни пружаоца услуга.

- *Протокол обједињеног одјављивања* (енгл. *Single Logout Protocol*)

  Дефинише начин уништавања свих активних сесија креираних од стране аутентификационог добављача. Разлог уништавања сесија може бити захтев самог корисника, истек сесије на страни аутентификационог добављача или захтев администратора.

- *Протокол упита и захтева тврђења* (енгл. *Assertion Query and Request Protocol*)

  Протокол захтева тврђења дефинише начин на који пружалац услуге може да захтева већ постојеће тврђење означено одређеним идентификатором. Протокол упита тврђења дефинише начин на који пружалац услуге може да захтева ново или већ постојеће тврђење коришћењем информације о субјекту.

- *Протокол мапирања именског идентификатора* (енгл. *Name Identifier Mapping Protocol*)

  Дефинише начин мапирања једног SAML именског идентификатора корисника у други. Пример може бити интеграција једног пружаоца услуга са другим. Потребно је да један пружалац услуга захтева од аутентификационог добављача именски идентификатор корисника који може да користи приликом приступа другом пружаоцу услуга.

3.3.1 Протокол аутентификационог захтева

Нама најзначајнији протокол јесте протокол аутентификационог захтева с обзиром да он представља начин путем којег се може иницирати аутентификација. Како би процес аутентификације започео, пружалац услуга шаље XML документ

са кореним XML елементом *AuthnRequest* добављачу идентитета. Захтев може садржати информације о субјекту чија се аутентификација иницира. Уколико је информација о субјекту изостављена, подразумева се да је корисник који је започео аутентификацију на страни пружаоца услуга сам субјекат. Пружалац услуга добија одговор у облику XML документа са кореним XML елементом *Response* и који садржи тврђење које се тиче траженог субјекта. Уколико је пружалац услуга добио одговор који означава успешну аутентификацију субјекта, он може да користи ово тврђење како би креирао сесију на својој страни.

Аутентификација субјекта не мора да се догоди приликом добијања аутентфикационог захтева. Може се десити да је субјекат већ аутентификован на страни добављача идентитета, па нема потребе затражити поновну аутентификацију, већ искористити информације о претходној и послати их у оквиру тврђења. XML елемент *AuthnRequest* којим се започиње аутентификација може изгледати овако:

```
<samlp:AuthnRequest
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="_189450638274920148438492"
Version="2.0"
IssueInstant="2020-03-12T11:34:23Z"
Destination="https://someIdp.com/saml/receive"
AssertionConsumerServiceURL="https://someSp.com/saml/receive"
ForceAuthn="true">
     <saml:Issuer>https://someSp.com</saml:Issuer>
     <samlp:RequestedAuthnContext Comparison="exact">
          <saml:AuthnContextClassRef>
               urn:oasis:names:tc:SAML:2.0:ac:classes:Password
          </saml:AuthnContextClassRef>
     </samlp:RequestedAuthnContext>
</samlp:AuthnRequest>
```

Елемент *AuthnRequest* може садржати неколико атрибута који служе да ближе опишу аутентификациони захтев. Атрибут *Version* дефинише верзију протокола SAML која се користи у захтеву и која се очекује у одговору. *IssueInstant* дефинише тачно време када је захтев креиран. *Destination* дефинише URL где пружалац услуга шаље захтев добављачу идентитета, док *AssertionConsumerServiceURL* дефинише URL где пружалац услуга очекује да добије одговор. Атрибут *ForceAuthn* дефинише да ли пружалац услуга захтева поновну аутентфикацију субјекта чак иако он већ има важећу сесију на страни добављача идентитета. Елемент *AuthnRequest* укључује и своје поделементе, па тако поделемент *Issuer* дефинише ко је пружалац услуга који шаље аутентификациони захтев, док поделемент *RequestedAuthnContext* дефинише који аутентификациони метод пружалац услуга захтева да се користи приликом аутентификације субјекта.

Када добављач идентитета прими аутентификациони захтев, потребно је да одговори на њега слањем XML документа чији је корени XML елемент *Response*

и који у себи има елемент *Status* који говори о томе да ли је аутентификација успешна или је дошло до грешке. Одговор мора да садржи барем једно тврђење и поменуте елементе тврђења *Subject*, *AuthnStatement*, *AudienceRestriction*. Пример *Response* елемента:

```
<samlp:Response
xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
ID="_sfspodp1494031ksds094"
Version="2.0"
IssueInstant="2020-06-12T11:11:32Z"
Destination="https://someSp.com/saml/receive"
InResponseTo="_189450638274920148438492">
     <saml:Issuer>https://someIdp.com/</saml:Issuer>
     <samlp:Status>
          <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
     </samlp:Status>
     <saml:Assertion
     ID="_l4k54l4kmc4co545k405lr4lk5c55"
     IssueInstant="2020-06-12T11:11:32Z">
          <saml:Subject>
              <saml:NameID>toma.tomic@test.com</saml:NameID>
              <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
                    <saml:SubjectConfirmationData
                     NotOnOrAfter="2021-03-21T05:13:12Z"
                     Recipient="https://someSp.com/saml/receive"/>
              </saml:SubjectConfirmation>
          </saml:Subject>
          <saml:Conditions
          NotBefore="2020-06-12T11:11:32Z"
          NotOnOrAfter="2021-03-21T05:13:12Z">
              <saml:AudienceRestriction>
                    <saml:Audience>https://someSp.com</saml:Audience>
              </saml:AudienceRestriction>
          </saml:Conditions>
          <saml:AuthnStatement
          AuthnInstant="2020-06-12T11:11:32Z"
          SessionNotOnOrAfter="2021-03-21T05:13:12Z"
          SessionIndex="_osvodifdjf148opoasjci2838592">
               <saml:AuthnContext>
                      <saml:AuthnContextClassRef>
                             urn:oasis:names:tc:SAML:2.0:ac:classes:Password
                      </saml:AuthnContextClassRef>
               </saml:AuthnContext>
          </saml:AuthnStatement>
          <saml:AttributeStatement>
               <saml:Attribute Name="firstName"">
                     <saml:AttributeValue>Toma</saml:AttributeValue>
               </saml:Attribute>
               <saml:Attribute Name="lastName"">
                     <saml:AttributeValue>Tomic</saml:AttributeValue>
               </saml:Attribute>
```

```
        </saml:AttributeStatement>
    </saml:Assertion>
</samlp:Response>
```

Добијени одговор може се представити реченицом: "Субјекат Тома Томић, са имејл адресом toma.tomic@test.com, је успешно аутентификован 12. јуна 2020. године коришћењем дефинисане лозинке од стране https://someIdp.com/ добављача идентитета и његова сесија траје до 21. марта 2021. године."

3.4 Веза

SAML подносилац захтева и SAML одзивна страна комуницирају помоћу размене XML докумената. Механизам помоћу којег се врши слање ових докумената, односно порука назива се *веза*. SAML не дефинише сопствене транспортне протоколе, већ се ослања на већ постојећи транспортни протокол HTTP.

Примери неких од веза које стандард SAML дефинише су [13]:

- *HTTP веза преусмеравања* (енгл. *HTTP Redirect Binding*)

  Ова веза дефинише размену порука коришћењем HTTP преусмеравања чији је статусни код HTTP одговора 302. Поруке се шаљу као део URL параметара. За преусмеравање порука користи се прегледач, па се ова веза често зове и *веза преусмеравања прегледача* (енгл. *browser redirect binding*). SAML подносилац захтева прво мора да компресује поруку коришћењем дефлационог компресовања, затим да је кодира помоћу base64 кодирања, и на крају, потребно је да уради URL кодирање. HTTP кеширање не треба да се користи приликом овакве размене података.

  Пример поруке послате коришћењем ове везе:

  ```
  https://someSamlResponderDomain.rs/saml?SAMLRequest=alQvbMnrPKsV...
  ```

- *HTTP POST веза* (енгл. *HTTP POST Binding*)

  Ова веза дефинише размену порука коришћењем HTML форме унутар које је порука кодирана помоћу base64 кодирања. Користи се форма која је самоподносећа, односно корисник не мора да кликне на дугме како би потврдио слање захтева обухваћено формом, већ то за њега уради веб прегледач. Цела форма је сакривена од ока корисника. За омогућавање ове везе користи се прегледач, па се ова веза често зове и *POST веза прегледача* (енгл. *browser POST binding*). HTTP кеширање не треба да се користи приликом овакве размене података.

  Пример форме која садржи поруку која се преноси коришћењем ове везе:

  ```
  <form method='post'
  action='https://someSamlResponderDomain.rs/saml'>
  ```

```
    <input type='hidden' name='SAMLRequest' value='cVt9nwkRPQaZ...'>
    <button type='submit'>POST</button>
</form>
```

3.5 Профил

Профили дефинишу како се тврђења, везе и протоколи захтева и одговора користе заједно у оквиру комуникације између SAML подносиоца захтева и SAML одзивне стране.

Примери неких од профила које стандард SAML дефинише су [13]:

- *SSO профил који укључује веб прегледач* (енгл. *Web Browser SSO Profile*)

  Дефинише како се комбинују тврђења и протокол аутентификационог захтева како би се постигла SSO аутентификација. Овај профил се може користити и уз HTTP везу преусмеравања и уз HTTP POST везу.

- *Профил откривања добављача идентитета* (енгл. *Identity Provider Discovery Profile*)

  Дефинише начин откривања добављача идентитета од стране пружаоца услуга. Пружалац услуга може открити само оне добављаче идентитета чије је веб странице корисник претходно посетио.

- *Профил обједињеног одјављивања* (енгл. *Single Logout Profile*)

  Дефинише како се протокол обједињеног одјављивања користи уз HTTP везу преусмеравања и уз HTTP POST везу.

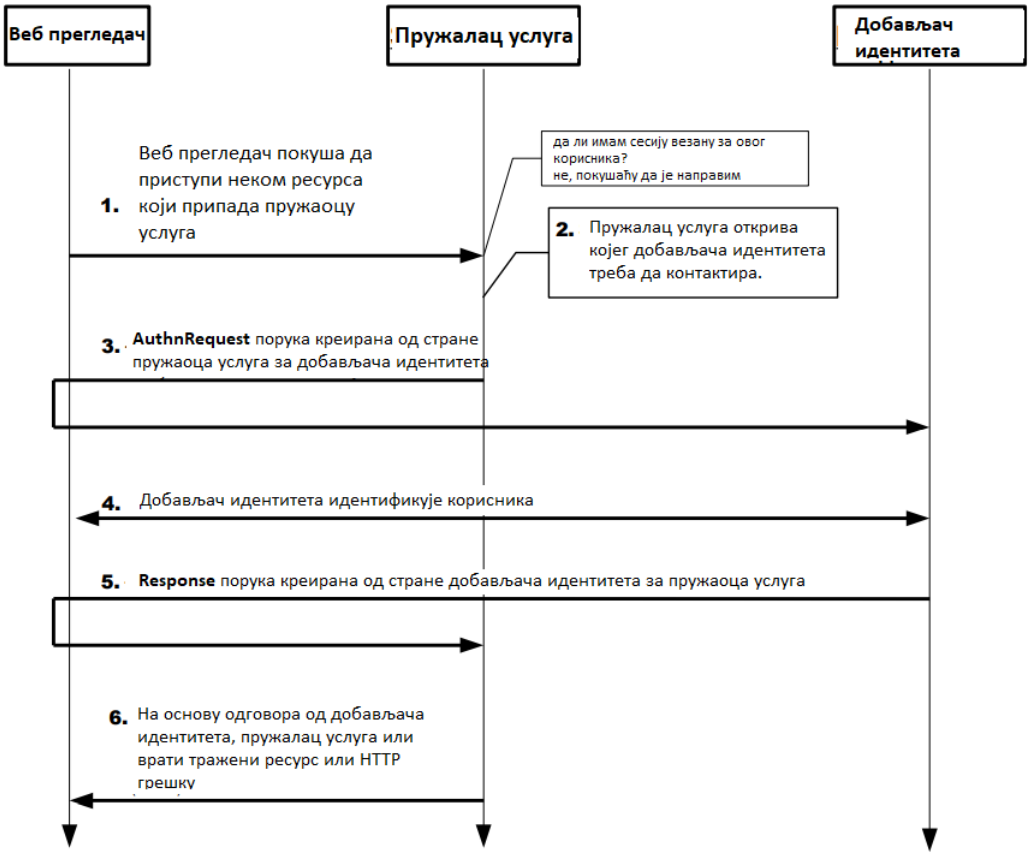
3.5.1 SSO профил који укључује веб прегледач

Нама најзначајнији профил јесте SSO профил који укључује веб прегледач с обзиром да он представља начин за постизање SSO аутентификације.

Процес обухваћен овим профилом може бити приказан графиком на слици 3.1. Састоји се из шест главних корака:

1. Корисник преко свог веб прегледача покушава да приступи неком заштићеном ресурсу на страни пружаоца услуга. За приступ заштићеном ресурсу користи се протокол HTTP. Корисник у овом тренутку није аутентификован и нема сесију на страни пружаоца услуга.

2. Пружалац услуга проверава да ли корисник има сесију. Пошто установи да сесија не постоји, он покушава да је направи. Овај корак се бави откривањем добављача идентита ка којем ће се послати захтев за аутентификацијом корисника. Могу се користити разне методе. Једна од њих је поменута метода коришћења профила откривања добављача идентитета. Други начин је слобода самих програмера, што може значити коришћење



Слика 3.1: SSO профил који укључује веб прегледач [6]

базе, колачића или неког трећег начина за утврђивање добављача идентитета.

3. Пружалац услуга креира аутентификациони захтев тако што прави XML документ који садржи корени елемент *AuthnRequest* упућен пронађеном добављачу идентитета. Пружалац услуга може да бира коју ће везу користити за транспорт захтева. То може бити HTTP веза преусмеравања или HTTP POST веза. Када је захтев направљен, он га шаље помоћу протокола HTTP као одговор на захтев направљен у кораку 1. Аутентификациони захтев се уз помоћ веб прегледача шаље добављачу идентитета.

4. Добављач идентитета прима аутентификациони захтев. Он валидира постојање пружаоца услуга тако што проверава да ли је послати пружалац услуга регистрован на страни добављача идентитета. Када утврди да је захтев валидан, добављач идентитета идентификује корисника и тражи му да се аутентификује уколико већ није или ако је атрибут *ForceAuthn* у оквиру аутентификационог захтева постављен на тачну вредност.

5. Корисник је аутентификован на страни добављача идентитета или се догодила грешка. Потребно је да се пошаље одговор пружаоцу услуга о статусу

аутентификације. Добављач идентитета шаље XML документ са кореним елементом *Response*. У оквиру овог одговора елемент *Status* се поставља на успешан или на грешку до које је дошло. Одговор се шаље помоћу протокола HTTP, али само коришћењем HTTP POST везе јер је одговор превише дугачак како би био део URL параметара у оквиру HTTP везе преусмеравања.

6. Пружалац услуга прима одговор од добављача идентитета. Валидира његову исправност. Уколико је одговор валидан, на основу његовог статуса пружалац услуга одређује да ли ће дати приступ траженом ресурсу из корака 1 или ће послати грешку као одговор. Ако је статус успешан, добављач идентитета може креирати сесију на својој страни користећи информације о кориснику из тврђења.

3.6 Сигурносни аспекти

Протокол SAML може бити изложен нападима злонамерних програма с обзиром да за реализацију својих профила користи веб прегледач. Одговори и захтеви који пролазе кроз веб прегледач могу бити пресретнути и коришћени у недозвољене сврхе.

У целом протоколу SAML тврђење је кључни податак који треба заштитити од злонамерног програма. Како би се избегли разни напади, SAML уводи правила којима штити тврђење. Неки од њих су:

- Пружалац услуга треба да обезбеди да је тврђење искоришћено само једном за креирање сесије. Ово је могуће постићи тако што памти сваки идентификатор захтева за аутентификацију и брише га оног тренутка када се добије одговор са атрибутом *InResponseTo* који садржи идентификатор захтева. На овај начин, пружалац услуга не креира сесију за одговоре чији атрибут *InResponseTo* не нађе у својој меморији. Тиме се спречава да злонамерни софтвер пресретне одговор и искористи га за приступ информацијама којима је приступ забрањен.

- Добављач идентитета може у оквиру одговора уврстити и елемент *OneTimeUse* који говори да пружалац услуга не сме чувати у својој меморији примљено тврђење, већ га сме искористити само једном. Тиме се спречава да злонамерни софтвер дође до тврђења преко базе пружаоца услуге. Овим се спречава такозвани напад понављања [5].

- Атрибути *NotBefore* и *NotOnOrAfter* у оквиру тврђења треба да буду што ближи једно другом. Тиме се постиже да украдена тврђења могу бити коришћене од стране злонамерног софтвера само у кратком временском периоду [5].

Како би се повећала сигурност у оквиру протокола SAML, уведене су могућности *потписивања* и *шифровања* захтева и одговора.

3.6.1 Потписивање

Потписивање SAML захтева даје доказ о подносиоцу захтева страни која даје одговор. Ово представља једну врсту аутентификације подносиоца захтева код стране које одговара. Важи и обрнуто, могуће је потписати и SAML одговор како би се страна која даје одговор аутентификовала код стране која је поднела захтев.

Потписивање је могуће уколико су обе стране размениле своје јавне кључеве. Страна која захтева или одговара потписује поруку користећи свој пар приватног и јавног кључа. Страна која прима поруку валидира потпис тако што израчуна очекивани потпис на основу јавног кључа стране која је послала поруку и примљене поруке. Уколико се очекивани и примљени потпис не слажу, добијена порука није валидна.

Такође, потписивање потврђује и тачност примљене поруке. Уколико злонамерни софтвер пресретне захтев или одговор и измени његов садржај како би приступио недозвољеним информацијама, потпис више неће бити исти и страна која прима поруку ће одбацити њену валидност јер ће упоредити очекивани потпис са приспелим потписом и они неће бити једнаки.

Потпис је могуће убацити као део XML документа. Представља се XML елементом *Signature*[2]. SAML захтева да у оквиру SSO профила који укључује веб прегледач буде потписан цео одговор или само тврђење. Ово значи да и елемент *Response* и елемент *Assertion* могу имати поделемент *Signature*. У оквиру елемента *Signature* налази се сам потпис одговора или тврђења, као и информације о томе којим јавним кључем и којим алгоритмом треба валидирати потпис. Пример елемента *Signature*:

```
<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/2001/04/xmldsig-more#rsa-sha256"/>
        <ds:Reference URI="#A-bb4a2147-0b32-4715-a5a4-0ce796566645">
            <ds:Transforms>
                <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>
                WhK7uBpsFX0ZMkydGGz6rn5FrmXBU4mtngBw3
```

[2]Све информације о потписивању XML документа могуће је наћи на страници https://www.w3.org/TR/xmldsig-core1/.

```
            </ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>eYY6Mq1j5Bu5c...</ds:SignatureValue>
    <ds:KeyInfo>
        <ds:X509Data>
            <ds:X509Certificate>MIIDGjCCAgKg...</ds:X509Certificate>
        </ds:X509Data>
    </ds:KeyInfo>
</ds:Signature>
```

Елемент *SignedInfo* садржи све информације о потписивању одговарајућег елемента. Пре сваког потписивања, потребно је извршити канонизацију елемента који треба да се потпише. Пример канонизације је наслеђивање свих простора имена уколико се потписује неки од поделемената поруке од стране својих наделемената. Алгоритам извршене канонизације записује се у оквиру елемента *CanonicalizationMethod* и могуће је имати више оваквих елемената. Страни која проверава потпис потребно је нагласити који алгоритам је коришћен за потписивање. Ову информацију је могуће наћи у оквиру елемента *SignatureMethod*. Елемент *Reference* садржи референцу на елемент који се потписује. У оквиру њега могу бити дефинисане разне трансформације, приказане елементом *Transforms*, које су се извршиле пре процеса *guīecīuje* (мапирања ниски произвољне дужине у ниске фиксне дужине). Алгоритам дигестије и дигестивна вредност могу се наћи у елементима *DigestMethod* и *DigestValue*. Добијена дигестивна вредност користи се приликом назначеног алгоритма потписивања. Сама вредност потписа дефинише се у оквиру елемента *SignatureValue*, а јавни кључ који је потребно користити за проверу потписа може се наћи у оквиру елемента *KeyInfo*.

Уколико се користи HTTP веза преусмеравања када се шаље аутентификациони захтев, не потписује се само XML документ већ је потребно потписати URL параметре који ће бити послати (то су захтев и алгоритам који се користи за потпис) и додати потпис као још један URL параметар.

3.6.2 Шифровање

Шифровање SAML одговора омогућава заштиту приватних података субјекта на који се одговор и тврђење односи. Када су информације шифроване, злонамерни софтвер који пресретне поруку неће бити у могућности да прочита заштићене податке. Њих може прочитати једино страна која треба да прими поруку, с обзиром да једино она поседује приватни кључ којим се може дешифровати порука.

Шифровање се у SAML-у може користити и за сакривање информација о кључу који се користи за један од процеса шифровања.

Шифровање је могуће уколико су обе стране размениле своје јавне кључеве. Добављач идентитета и пружалац услуга дефинишу своје јавне кључеве у оквиру својих метаподатака[3] које размењују у току процеса мануелне регистрације који се врши само једном када пружалац услуга одлучи да жели да користи услуге добављача идентитета. Процес аутентификације није могуће извршити пре регистрације. Потребно је да се пружалац услуга региструје на страни добављача идентитета и обрнуто. Након регистрације, стране у комуникацији могу користити шифровање, тако што страна која одговара креира једнократни симетрични кључ којим ће шифровати поруку, а затим га шифрује користећи јавни кључ стране која треба да прими поруку. Страна која прима поруку прво дешифрује симетрични кључ користећи свој приватни кључ, а затим њега користи да дешифрује примљену поруку. Из овога следи да се јавни кључеви користе само за размену симетричног кључа који се користи у симетричној криптографији. Један симетрични кључ се користи за шифровање и дешифровање само једне поруке.

Протокол SAML дозвољава шифровање целог тврђења, само атрибута или именског идентитета у оквиру елемента који говори о субјекту. За шифровање ових елемената се користе правила дефинисана као део стандарда XML Encryption[4] [1]. Шифровано тврђење се представља елементом *EncryptedAssertion*, шифрован атрибут се представља елементом *EncryptedAttribute*, а шифровани именски идентитет елементом *EncryptedID*. Сви они у себи носе шифроване податке и информације о кључу и алгоритму који је коришћен приликом шифровања. Пример једног шифрованог атрибута:

```
<EncryptedAttribute>
    <xenc:EncryptedData
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
    Type="http://www.w3.org/2001/04/xmlenc#Element">
        <xenc:EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
        <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
            <xenc:EncryptedKey>
                <xenc:EncryptionMethod
                Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
                <xenc:CipherData>
                    <xenc:CipherValue>MgpCdUHdShN...</xenc:CipherValue>
                </xenc:CipherData>
            </xenc:EncryptedKey>
        </dsig:KeyInfo>
        <xenc:CipherData>
            <xenc:CipherValue>QuGhJLTP5JkL5Un2...=</xenc:CipherValue>
        </xenc:CipherData>
```

[3] Више информација о метаподацима који се размењују могуће је пронаћи на страници https://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf.

[4] Све информације о шифровању XML докумената могуће је пронаћи на страници https://www.w3.org/TR/xmlenc-core1/.

```
    </xenc:EncryptedData>
</EncryptedAttribute>
```

Страна која потписује елемент је дужна да достави информацију о коришћеном алгоритму за шифровање у оквиру елемента *EncryptionMethod*. Елемент *CipherData* и његов поделемент *CipherValue* у оквиру елемента *EncryptedAttribute* дефинишу вредност шифрованог атрибута. За дешифровање овог атрибута потребно је користити симетрични кључ чија се шифрована вредност налази у елементу *CipherData* у оквиру елемента *EncryptedKey* који је део елемента *KeyInfo*. Алгоритам коришћен за шифровање симетричног кључа дефинисан је у елементу *EncryptionMethod* који је део елемента *EncryptedKey*.

3.7 Добре и лоше стране протокола SAML

С обзиром да је SAML протокол који се базира на размени XML докумената, то му омогућава његову реализацију на свим могућим платформама. Сва логика аутентификације је централизована и комплексност идентификације и аутентификације корисника се потпуно пребацује на страну добављача идентитета.

Једна од највећих мана протокола SAML јесте његова комплексност. Постоји превише правила која треба испоштовати и имплементација јесте временски захтевна. Зато су се временом развили једноставнији SSO протоколи о којима ће бити реч у наредним поглављима.

Глава 4

Протокол OAuth

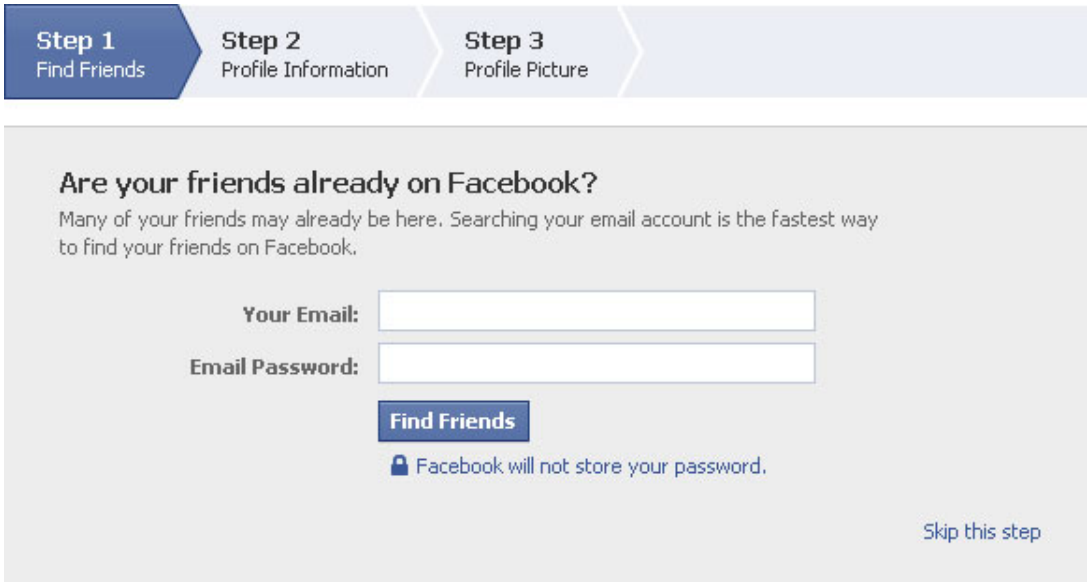
Комплексност протокола SAML, развој технологије и настанак нових проблема и захтева у процесу аутентификације које протокол SAML није могао да реши довео је до стварања новог аутентификационог протокола под називом OAuth.

Један од нових изазова са којим су програмери морали да се суоче и који је утицао на развој новог протокола јесте аутентификација у оквиру мобилних апликација. Развој мобилне технологије довео је до потребе за унапређивањем мобилног корисничког искуства. Проблем који се јавио у развоју мобилних апликација јесте одржавање дугорочне сесије чак и у тренуцима када је апликација искључена. Коришћење колачића се није показало као добра пракса на мобилним уређајима [1].

Други проблем који се јавио и чије је ефикасно и сигурно решење морало бити део неког новог протокола јесте *делегирање ауторизације*. Овај проблем би се могао описати као проблем приступа корисничким подацима који припадају апликацији А из неке друге апликације Б, с тиме да апликација Б нема увид у лозинку која је потребна како би се приступило апликацији А. Ово је проблем с чијим се решењем данас свакодневно срећемо, а то је опција аутентификације и приступа некој апликацији коришћењем Гугл или Фејсбук налога.

Раније су постојала само јако лоша решења овог проблема у којем је једна апликација тражила лозинку за приступ другој апликацији како би довукла потребне податке. Пример оваквог решења се налази на слици 4.1 где видимо да је Фејсбук тражио имејл и лозинку која се користи за приступ унетом имејлу како би пронашао листу нових могућих пријатеља који су део контаката на имејл налогу. Уколико се користи овај начин делегирања ауторизације, апликација којој смо открили лозинку има потпуни приступ нашим подацима и право на вршење акција у наше име. Једини начин да се опозове приступ другој апликацији јесте мењање лозинке. У примеру који је приказан на слици 4.1 Фејсбук добија могућност да приступи садржају свих приватних мејлова, па чак и могућност

[1] Коришћење колачића је лоше у овкиру мобилних уређаја јер је ограниченог деловања када се не користи веб прегледач. На мобилним уређајима могуће је користити апликације које су повезане са интернетом, али коришћење колачића је ограничено у оквиру једне апликације. Дељење колачића између стране која пружа услугу и стране која врши аутентификацију није могуће осим уколико се не користи веб прегледач.



Слика 4.1: Пример лошег решења проблема делегирања ауторизације [3]

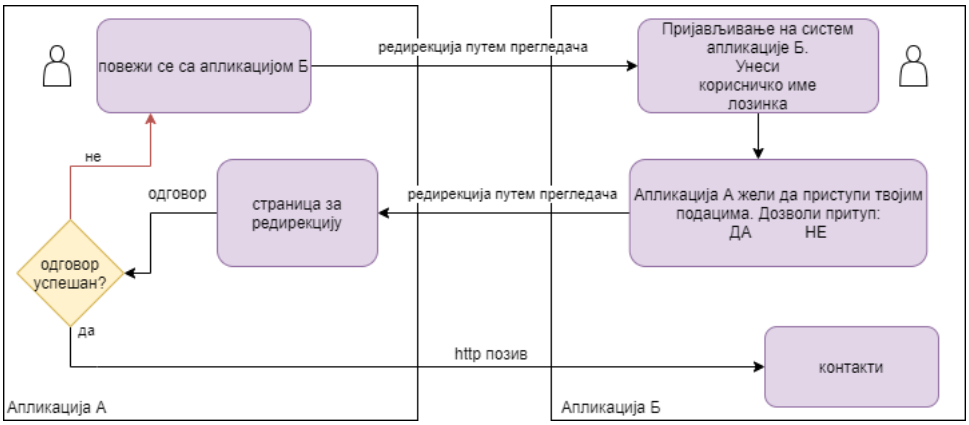
креирања и слања нових. Још једна мана овог приступа јесте немогућност централизованог излиставања свих апликација којима смо дали приступ. Једино сам корисник може да води своју евиденцију датих приступа, не и сама апликација.

OAuth је протокол који омогућава сигуран начин делегирања ауторизације без потребе за дељењем лозинке. Датира од 2006. године, а креиран је од стране Твитер и Гугл тима. Године 2010. постао је стандард под називом *RFC 5849*[2]. Ту не стаје његов развој, 2012. године излази друга верзија овог протокола OAuth 2.0 и постаје стандард под називом *RFC 6749*[3] [4]. Овај протокол је и данас популаран и широко коришћен. Главна предност овог протокола је пре свега сигурност, али и могућност ограничавања количине података која ће бити дељена и време у току којег ће информације бити дељене.

Апликација А, којој су потребне информације из апликације Б о кориснику, приказује дугме на које је потребно кликнути како би се процес покренуо. Када корисник кликне на дугме, апликација А преусмерава корисника коришћењем прегледача на апликацију Б и апликацији Б доставља информацију о томе који подаци о кориснику су јој потребни. У том тренутку, кориснику се приказује прозор за пристанак на дељење информација. Уколико корисник није пристао на дељење података, апликација Б шаље грешку апликацији А. Уколико је пристао, апликација Б шаље дозволу и од тог тренутка, апликација А може слати HTTP захтеве апликацији Б како би покупила тражене податке (нпр. листу контаката).

[2] Детаљи о стандарду *RFC 5849* се могу наћи на адреси https://tools.ietf.org/html/rfc5849.
[3] Детаљи о стандарду *RFC 6749* се могу наћи на адреси https://tools.ietf.org/html/rfc6749.

Груб приказ процеса који дефинише протокол OAuth 2.0 приказан је на слици 4.2.



Слика 4.2: Груб приказ протокола OAuth 2.0

Проблем делегирања ауторизације приказан на слици 4.1 може се решити коришћењем протокола OAuth 2.0. Фејсбук би приказао дугме које би преусмерило корисника на Гугл страницу која би тражила дозволу приступа подацима о контактима. Када корисник пристане, Фејсбук шаље захтев Гуглу за добијање свих контаката које припадају тренутном кориснику.

4.1 Основни појмови

Ради бољег разумевања овог протокола, потребно је увести основне појмове који се користе при његовом дефинисању. Сви појмови биће објашњени коришћењем проблема приказаног на слици 4.1.

1. *Власник ресурса* (енгл. *Resource owner*)

   Термин који дефинише корисника који је власник података које нека апликација жели да дохвати. Он је особа која мора да дозволи приступ траженим подацима. У поменутом проблему, власник ресурса јесте корисник чији Фејсбук налог тражи приступ контаката са Гугла.

2. *Клијент* (енгл. *Client*)

   Термин који дефинише апликацију која жели да дохвати податке чији је власник неки корисник, односно власник ресурса. У поменутом проблему, клијент јесте Фејсбук апликација.

3. *Ауторизациони послужилац* (енгл. *Authorization server*)

   Термин који дефинише систем који се користи за ауторизацију приступа подацима. У поменутом проблему, ауторизациони послужилац јесте Гугл.

4. *Послужилац ресурса* (енгл. *Resource server*)

Термин који дефинише систем или API[4] који чува податке којима клијент жели да приступи. У поменутом проблему, послужилац ресурса јесте Гуглов API за дохватање контаката.

5. *Ауторизациона дозвола* (енгл. *Authorization grant*)

   Термин који дефинише дозволу коју је власник ресурса дао за приступ подацима. Ова дозвола представља доказ да је власник ресурса кликнуо на ДА приликом тражења дозволе приступа. У поменутом проблему, дозвола за ауторизацију је одговор који ауторизациони послужилац шаље клијенту након што је власник ресурса дао приступ подацима.

6. *Опсег* (енгл. *Scope*)

   Термин који представља део захтева за повезивање преко којег се клијенту ограничава приступ само одређеним подацима. Један од проблема лошег решења делегирања ауторизације приказаног на слици 4.1 јесте неограничавање приступа информацијама клијентској апликацији А од стране друге апликације Б. Као што је већ речено, Фејсбук не добија само тражене контакте, већ и приступ свим приватним мејловима корисника и могућност слања нових. Протокол OAuth решава и ову врсту проблема тиме што ограничава клијентску апликацију пружајући јој приступ само одређеним информацијима за које је власник ресурса дао своју дозволу. Ауторизациони сервер дефинише своју листу опсега које он разуме и које очекује да добије при клијентском захтеву за повезивање. Пример једног опсега може бити читање контакта или слање мејла. Клијент приликом слања захтева за повезивање шаље жељену листу опсега која представља подскуп опсега дефинисаних од стране ауторизационог сервера. Добијени скуп опсега ауторизациони сервер користи како би приказао власнику ресурса страницу на којој су излистани опсези којима клијент захтева приступ и на којој власник ресурса даје своју дозволу за приступ њима.

7. *URI за преусмеравање* (енгл. *Redirect URI*)

   Термин који дефинише URI (чији је власник клијент) на коме ауторизациони послужилац треба да достави ауторизациону дозволу. У поменутом проблему, URI за преусмеравање јесте неки Фејсбуков URL који ће обрадити ауторизациону дозволу.

8. *Приступни токен* (енгл. *Access token*)

   Термин који дефинише кључ који клијент користи како би приступио послужиоцу ресурса и дохватио потребне податке за које је власник ресурса дао ауторизациону дозволу. Овај приступни токен ауторизациони послужилац може да шаље у исто време кад и ауторизациону дозволу, а може

[4] API је скраћеница енглеског термина Application Programming Interface, а представља скуп метода које омогућавају комуникацију између два софтверска продукта.

га слати и тек на захтев самог клијента уз коришћење ауторизационе дозволе. У поменутом проблему, приступни токен јесте кључ који Гугл даје Фејсбуку који ће му омогућити приступ API-ју за дохватање контаката.

4.2 Ауторизациона дозвола

Ток протокола OAuth се разликује према томе која ауторизациона дозвола је захтевана од стране клијента. Клијент приликом слања захтева за повезивање са ауторизационим послужиоцем шаље који тип одговора, односно коју ауторизациону дозволу очекује. Врсте ауторизационих дозвола које клијент може да захтева су [8]:

1. *Ауторзациони код* (енгл. *Authorization code*)
2. *Имплицитна дозвола* (енгл. *Implicit grant*)
3. *Лозинка власника ресурса* (енгл. *Resource owner password credentials*)
4. *Клијентски креденцијали* (енгл. *Client credentials*)

4.2.1 Ауторизациони код

Ауторизациони код представља ниску карактера коју ауторизациони послужилац шаље клијенту као одговор на захтев за повезивање.

Овај тип ауторизационе дозволе користе углавном веб апликације које имају свог послужиоца и којег ћемо у даљем тексту називати клијентски послужилац. Овакве апликације имају могућност коришћења редирекције путем прегледача и њихов извршни код сакривен је од јавности (користи се послужилац како би се дохватили подаци). Ове две карактеристике омогућавају сигурно дељење информација између клијента и ауторизационог послужиоца. Сви делови протокола који могу бити јавни се извршавају путем прегледача, док они који морају остати ван домашаја јавности се извршавају коришћењем послужилаца и HTTP позива.

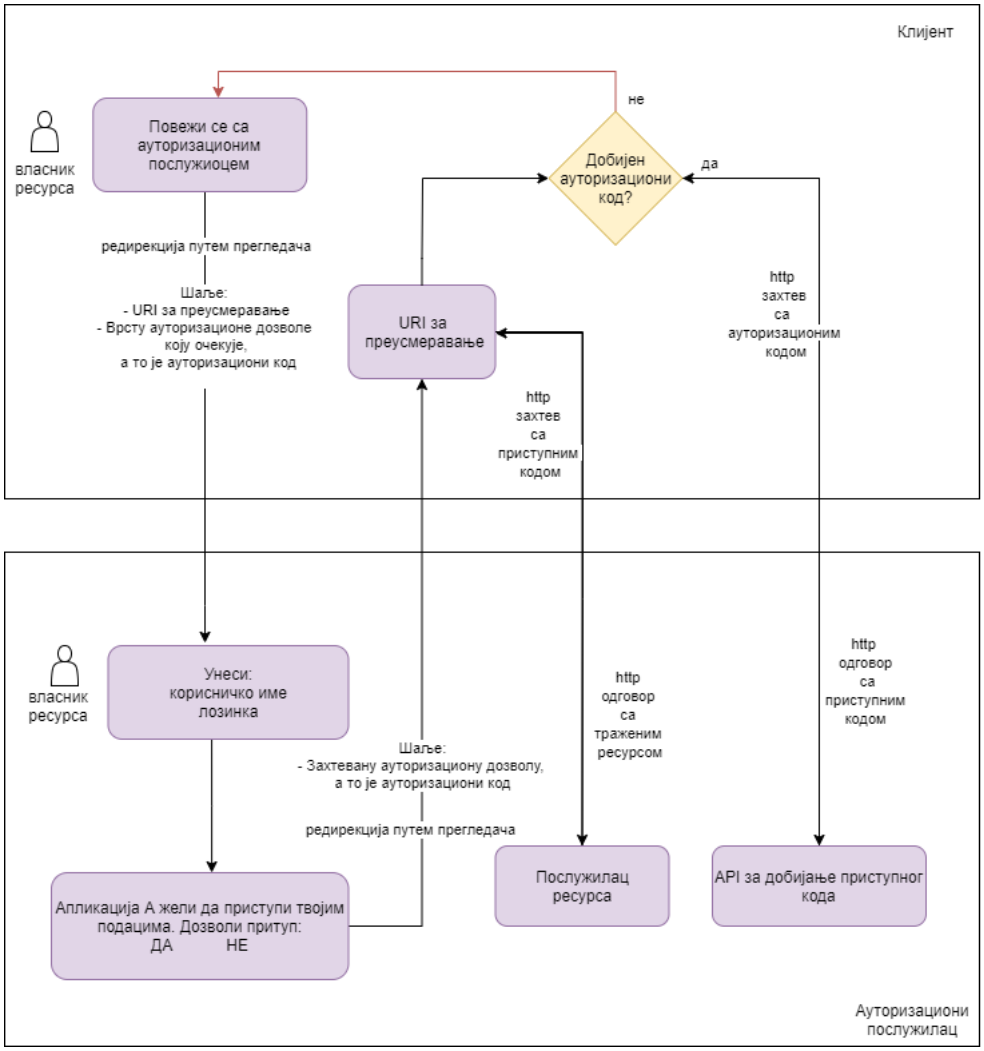
Добијање ауторизационог кода врши се коришћењем редирекције путем прегледача. Ово значи да је ауторизациони код доступан јавности. Из овог разлога, траженом ресурсу није могуће приступити коришћењем овог кода јер би злонамерни софтвер могао пресрећи одговор ауторизационог послужиоца и користити добијени ауторизациони код за неовлашћен приступ подацима.

Да би клијент приступио траженом ресурсу, потребно је да размени ауторизациони код за приступни токен. Ова размена врши се путем HTTP комуникације између два послужиоца (клијентског и ауторизационог) и одговор није доступан јавности јер нико не може да га пресретне. То би значило да знање о приступном токену има само клијентски послужилац и траженим ресурсима може приступити само он. Приликом размене ауторизационог кода за приступни токен потребно је да се клијентски послужилац аутентификује код ауторизационог

послужиоца коришћењем клијентског идентификатора и клијентске тајне коју знају само клијент и ауторизациони послужилац. Клијентска тајна се добија у процесу мануелне регистрације клијента на страни ауторизационог послужиоца која се врши само једном када клијент одлучи да жели да подржи услуге одређеног ауторизационог послужиоца. Процес аутентификације није могуће извршити пре регистрације.

Уколико неки злонамерни програм пресретне ауторизациони код и покуша помоћу њега да добије приступни токен, то не би било могуће јер злонамерном софтверу није позната клијентска тајна која се користи при аутентификацији на ауторизационом послужиоцу.

Процес протокола OAuth 2.0 са ауторизационим кодом као захтеваном дозволом приказан је на слици 4.3.



Слика 4.3: Приказ протокола OAuth 2.0 са ауторизационим кодом као дозволом

У циљу илустрације коришћења ауторизационог кода, наводимо пример употребе ауторизационог кода као ауторизационе дозволе за добијање приступа заштићеном Јутјуб API-ју. Цео поступак се састоји из неколико корака:

1. Шаље се GET захтев у коме се тражи ауторизациони код од Гугл API-ја за приступ апликацији Јутјуб:

```
GET /o/oauth2/auth
    ?client_id=315466250127-kljsl9p0nu22gk16vj0npoo12rrgedmg
        .apps.googleusercontent.com
    &redirect_uri=https%3A%2F%2Flocalhost%3A44373%2Fsignin-google
    &scope=https://www.googleapis.com/auth/youtube
    &response_type=code HTTP/1.1
Host: accounts.google.com
```

2. Послати захтев добија одговор на адресу https://localhost:44373/signin-google која је наведена као *redirect_ uri* параметар. Пример одговора:

```
https://localhost:44373/signin-google
    ?code=4/0AX4XfWiz-wM7OE6EXnkOe47HP9hqDSnUQ1Yn7-2pjBkGu0fDqE55u82
    &scope=https://www.googleapis.com/auth/youtube
```

3. Размењује се добијени ауторизациони код за приступни токен. Пример POST захтева који то ради:

```
POST /o/oauth2/token HTTP/1.1
Host: accounts.google.com
Content-Type: application/x-www-form-urlencoded

code=4/0AX4XfWiz-wM7OE6EXnkOe47HP9hqDSnUQ1Yn7-2pjBkGu0fDqE55u82
?client_id=315466250127-kljsl9p0nu22gk16vj0npoo12rrgedmg
            .apps.googleusercontent.com
&client_secret=someSecret
&redirect_uri=https://localhost:44373/signin-google
&grant_type=authorization_code
```

4. Послати POST захтев добија приступни токен у одговору:

```
{
    "access_token": "ya29.a0ARrdaM87M-r0VHfNdNYq_Jb0T5FpmW-pVgXN7yW
        ErtimcyIgTSbh_UI57dbkD3sDbUqQB_KaChomaCnCUzvfOy_q-4ONrDR8vn
        PB68k2oajVSVc8048zf7Mj9cIsVGbd7JUBOKR5CbFUdSBFPl1_mEE9gH41",
    "expires_in": 3599,
    "scope": "https://www.googleapis.com/auth/youtube",
    "token_type": "Bearer"
}
```

5. Добијени приступни токен се смешта у Authorization заглавље захтева и приступа се заштићеном Јутјуб API-ју:

```
GET /youtube/v3/channels?part=id&mine=true HTTP/1.1
Host: www.googleapis.com
Authorization: Bearer ya29.a0ARrdaM87M-r0VHfNdNYq_Jb0T5FpmW-pVgXN7yW
    ErtimcyIgTSbh_UI57dbkD3sDbUqQB_KaChomaCnCUzvfOy_q-4ONrDR8vnPB68k
    2oajVSVc8048zf7Mj9cIsVGbd7JUBOKR5CbFUdSBFPl1_mEE9gH41
```

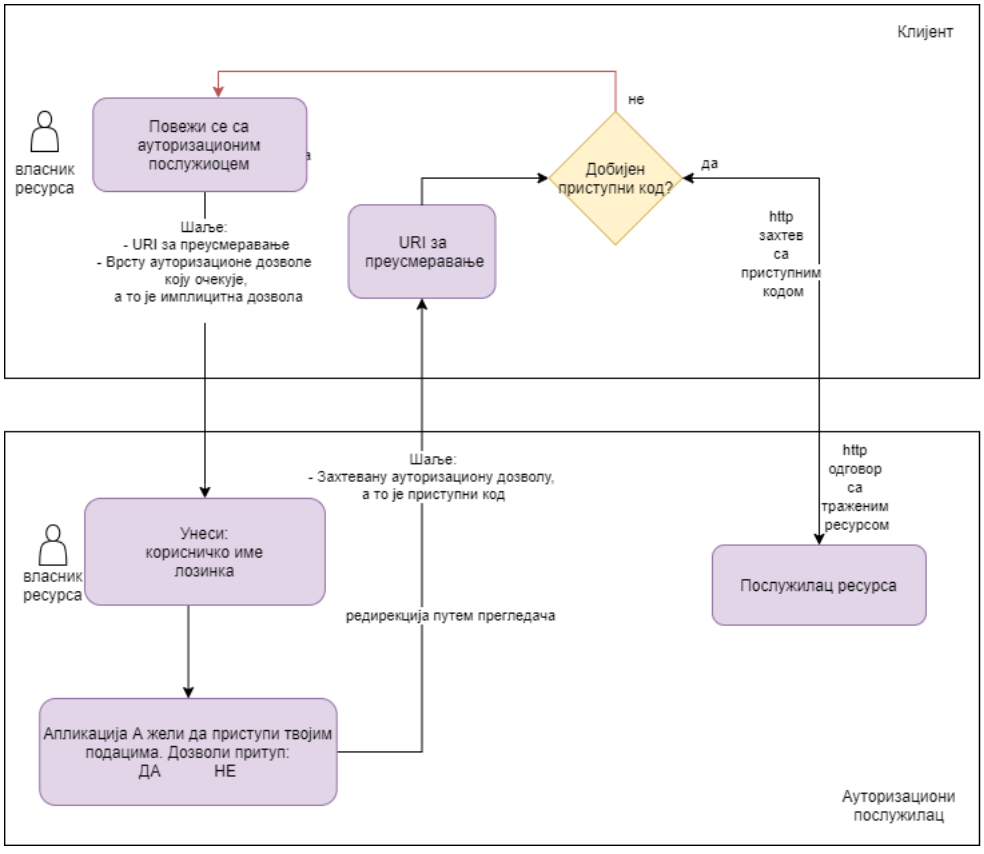
4.2.2 Имплицитна дозвола

Овај тип ауторизационе дозволе користе апликације које немају свог послужиоца. То су апликације које се извршавају тотално на страни прегледача и које немају могућност сигурног чувања клијентске тајне. Називају се *једностране апликације* (енгл. *Single-page applications*).

Једина разлика у односу на протокол са ауторизационим кодом као дозволом је у томе што нема размене ауторизационог кода и приступног токена. Клијент одмах као одговор приликом редирекције добија приступни токен. Коришћењем њега шаље захтев послужиоцу ресурса.

Коришћење овог типа ауторизационе дозволе има сигурносних мана. Приступни токен и лични подаци корисника су изложенији потенцијалним нападима.

Процес протокола OAuth 2.0 са имплицитном дозволом као захтеваном дозволом приказан је на слици 4.4.



Слика 4.4: Приказ протокола OAuth 2.0 са имплицитном дозволом као траженом ауторизационом дозволом

Као илустрацију, наводимо пример употребе имплицитне дозволе као ауторизационе дозволе за добијање приступа заштићеном Јутјуб API-ју. Цео поступак се састоји из неколико корака:

1. Шаље се GET захтев у коме се тражи ауторизациони код од Гугл API-ја за приступ апликацији Јутјуб:

```
GET /o/oauth2/auth
    ?client_id=315466250127-kljsl9p0nu22gk16vj0npoo12rrgedmg
        .apps.googleusercontent.com
    &redirect_uri=https%3A%2F%2Flocalhost%3A44373%2Fsignin-google
    &scope=https://www.googleapis.com/auth/youtube
    &response_type=token HTTP/1.1
Host: accounts.google.com
```

2. Послати захтев добија одговор на адресу https://localhost:44373/signin-google која је наведена као *redirect_ uri* параметар. Пример одговора:

```
https://localhost:44373/signin-google
    ?access_token=ya29.a0ARrdaM_kiZKVbqqvwYans20rMXFuXqDeydSWMlBzlm
        6rMW69PCj9-i0BIpgEK5VDwa109Mi-MtG3dQYi6NsaiqQqLlMNee3lE2x_5
        4IyQ38HgtxcY3ATcI_dzXcbYuVD2wjYvjrPZYuZ4d0K9eqNah-Lr7iG
    &token_type=Bearer
    &expires_in=3599
    &scope=https://www.googleapis.com/auth/youtube
```

3. Добијени приступни токен се смешта у Authorization заглавље захтева и приступа се заштићеном Јутјуб API-ју:

```
GET /youtube/v3/channels?part=id&mine=true HTTP/1.1
Host: www.googleapis.com
Authorization: Bearer ya29.a0ARrdaM_kiZKVbqqvwYans20rMXFuXqDeydSWMl
    Bzlm6rMW69PCj9-i0BIpgEK5VDwa109Mi-MtG3dQYi6NsaiqQqLlMNee3lE2x_5
    4IyQ38HgtxcY3ATcI_dzXcbYuVD2wjYvjrPZYuZ4d0K9eqNah-Lr7iG
```

4.2.3 Лозинка власника ресурса

Овај тип ауторизационе дозволе користе апликације којима власник ресурса апсолутно верује јер је потребно да подели своју лозинку са клијентом. Користе га обично мобилне апликације које су део система ауторизационог послужиоца, па је висок ниво веровања клијенту увек присутан.

Потребно је прво да клијент затражи од власника ресурса корисничко име и лозинку за приступ ауторизационом послужиоцу. Клијент приликом слања захтева за повезивање шаље добијено корисничко име и лозинку власника ресурса. Захтев се шаље преко HTTP позива, што значи да се редирекција не користи. Као одговор одмах се добија приступни токен. Приликом слања овог захтева потребно је да се клијент аутентификује код ауторизационог послужиоца коришћењем клијентског идентификатора и клијентске тајне коју знају само клијент и ауторизациони послужилац.

Овај тип ауторизационе дозволе омогућава клијентској апликацији да избегне процес провере и чувања лозинке корисника вршењем размене лозинке за дуготрајни приступни токен.

Као илустрацију, наводимо пример употребе лозинке власника ресурса као ауторизационе дозволе за добављање приступног токена у апликацији Okta. Цео поступак се састоји из неколико корака:

1. Шаље се POST захтев у коме се тражи приступни токен:

```
POST /oauth2/default/v1/token HTTP/1.1
Host: dev-7375101.okta.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic base64Encode(0oa4z2u20nG1u01dZ5d6:someClientSecret)

grant_type=password
&username=milica.djuric
&password=somePassword
&scope=someCustomScope
```

2. Послати POST захтев добија приступни токен у одговору:

```
{
    "token_type": "Bearer",
    "expires_in": 3600,
    "access_token": "eyJraWQiOiJiNjJ3NEg1RzUzcEFZUXNjamhGc1Z3WTBpenN
        KWEJsQkM3cWpYVV81R3JVIiwiYWxnIjoiUlMyNTYifQ.eyJ2ZXIiOjEsImp0
        aSI6IkFULkNrZFJLdFQ5cDNXQ3J3TDN2R2tzN0N1ejhINEJreE1DYVlRRnZu
        dlhEZEEiLCJpc3MiOiJodHRwczovL2Rldi03Mzc1MTAxLm9rdGEuY29tL29h
        dXRoMi9kZWZhdWx0IiwiYXVkIjoiYXBpOi8vZGVmYXVsdCIsImlhdCI6MTYy
        ODM4MTM1MiwiZXhwIjoxNjI4Mzg0OTUyLCJjaWQiOiIwb2E0ejJ1MjBuRzF1
        MDFkWjVkNiIsInNjcCI6WyJzb21lQ3VzdG9tU2NvcGUiXSwic3ViIjoiMG9h
        NHoydTIwbkcxdTAxZFo1ZDYifQ.ZUQhVQQvBW787LniDiN1FrEL7YqfHkvoH
        XcfYk2c3ur8IPI7cD87DYGSyRA8dYLf4X2Yyq4Vtfq0upA76oP52Xkr-xPcx
        1iBvXcaTohE9OHr8zl5WNjcTU9EGpSQ75k3nzomjyY3YGsMH3peqojndBlvD
        kxetvpC-dDGehmcgz2j7_2lq1yhtQ7Qqa_9qFNKT9LW4rwmnElxbkXyL6piC
        ONGqs4CcoqAs1mTAUHVDC-qPbF26WlsOLtCqMXw4U6YfDXITZhvwhP3MUAJq
        U7StQ2tkF_IyyXOtXldlwashZtyBZrA8TPpf9N6Tn_BV5-QUb-4DC4FUtJrZ
        BcN7DhmlQ",
    "scope": "someCustomScope"
}
```

4.2.4 Клијентски креденцијали

Овај тип ауторизационе дозволе користи се приликом аутентификације машине, а не корисника. Користе се код неких системских позива на које крајњи корисник нема утицаја. Не захтева постојање веб прегледача, захтев за добијање токена се шаље директно преко протокола HTTP.

Клијент шаље клијентски идентификатор и клијентску тајну ауторизационом послужиоцу како би се аутентификовао. Ауторизациони послужилац валидира примљене податке и шаље приступни токен уколико су подаци валидни. Клијент са примљеним приступним токеном може приступити потребним ресурсима.

Као илустрацију, наводимо пример употребе клијентских креденцијала као ауторизационе дозволе за добављање приступног токена у апликацији Okta. Цео поступак се састоји из неколико корака:

1. Шаље се POST захтев у коме се тражи приступни токен:

```
POST /oauth2/default/v1/token HTTP/1.1
Host: dev-7375101.okta.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic base64Encode(0oa4z2u20nG1u01dZ5d6:someClientSecret)

grant_type=client_credentials
&scope=someCustomScope
```

2. Послати POST захтев добија приступни токен у одговору:

```
{
    "token_type": "Bearer",
    "expires_in": 3600,
    "access_token": "eyJraWQiOiJiNjJ3NEg1RzUzcEFZUXNjamhGc1Z3WTBpenN
        KWEJsQkM3cWpYVV81R3JVIiwiYWxnIjoiUlMyNTYifQ.eyJ2ZXIiOjEsImp0
        aSI6IkFULkNrZFJLdFQ5cDNXQ3J3TDN2R2tzN0N1ejhINEJreE1DYVlRRnZu
        dlhEZEEiLCJpc3MiOiJodHRwczovL2Rldi03Mzc1MTAxLm9rdGEuY29tL29h
        dXRoMi9kZWZhdWx0IiwiYXVkIjoiYXBpOi8vZGVmYXVsdCIsImlhdCI6MTYy
        ODM4MTM1MiwiZXhwIjoxNjI4MzgOOTUyLCJjaWQiOiIwb2EOejJ1MjBuRzF1
        MDFkWjVkNiIsInNjcCI6WyJzb21lQ3VzdG9tU2NvcGUiXSwic3ViIjoiMG9h
        NHoydTIwbkcxdTAxZFo1ZDYifQ.ZUQhVQQvBW787LniDiN1FrEL7YqfHkvoH
        XcfYk2c3ur8IPI7cD87DYGSyRA8dYLf4X2Yyq4VtfqOupA76oP52Xkr-xPcx
        1iBvXcaTohE9OHr8zl5WNjcTU9EGpSQ75k3nzomjyY3YGsMH3peqojndBlvD
        kxetvpC-dDGehmcgz2j7_2lq1yhtQ7Qqa_9qFNKT9LW4rwmnElxbkXyL6piC
        ONGqs4CcoqAs1mTAUHVDC-qPbF26WlsOLtCqMXw4U6YfDXITZhvwhP3MUAJq
        U7StQ2tkF_IyyXOtXldlwashZtyBZrA8TPpf9N6Tn_BV5-QUb-4DC4FUtJrZ
        BcN7DhmlQ",
    "scope": "someCustomScope"
}
```

4.3 Добре и лоше стране протокола OAuth

Јако важна предност протокола OAuth јесте једноставност његове имплементације. Све што може да се постигне компликованим протоколом SAML, може се постићи и једноставним протоколом OAuth који користи већ широко познати трансфер података HTTP и формат JSON. Такође, OAuth је дизајниран тако да подржава и мобилне и десктоп платформе, за разлику од протокола SAML. Још једна јако важна предност протокола OAuth је што ограничава приступ само оним информацијама за које је корисник дао своју дозволу.

Једна од мана овог приступа јесте то што протокол OAuth никада није дизајниран да се користи у сврхе аутентификације, већ у сврхе ауторизације, па ни нема стандардизован начин за прибављање информација о кориснику. Друга мана је што безбедносно критични подаци могу бити послати коришћењем прегледача у зависности од изабране ауторизационе дозволе. Још једна мана овог приступа је што апликације које немају свог послужиоца користе Javascript и немају сигуран начин чувања приступног токена (localStorage и sessionStorage могу бити доступни злонамерном софтверу.)

Глава 5

Протокол OpenId Connect

Када је протокол OAuth направљен, дизајниран је тако да омогући ефикасно и једноставно давање дозвола једном систему за приступ информацијама другог система, уз ограничење да се информације односе на истог корисника који даје дозволу за приступ. Ово значи да његова сврха није омогућавање аутентификације корисника на више система, већ ауторизације. Како би један протокол био намењен за аутентификацију корисника, потребно је да дефинише стандардизован начин за добављање информација о кориснику који покушава да се аутентификује. Ово је неопходно, с обзиром да приликом аутентификације систем мора знати ком кориснику треба да направи сесију. За прављење сесије систем може користити нпр. корисничко име, идентификатор или имејл корисника. Иако протокол OAuth нема стандардизован начин за добављање информација о кориснику, годинама се користио у ту сврху. Имплементација није могла бити јединствена за све системе, већ је зависила од начина на који је систем који врши аутентификацију дефинисао добављање информација о кориснику. То је велика мана оваквог приступа с обзиром да уколико се један систем одлучи да подржи аутентификацију путем више различитих система, било би потребно да има више имплементација довлачења података о кориснику у зависности од тога који систем за аутентификацију корисник изабере. Како би се избегло овакво ограничење у имплементацији, развијен је нови протокол под називом *протокол OpenId Connect*.

Протокол OpenId Connect (скраћено OIDC) је заправо само додатан слој над протоколом OAuth 2.0 који омогућава аутентификацију корисника на више система. Настао је 2014. године од стране организације *OpenId Foundation*[1]. Гугл је 2015. године препоручио коришћење овог протокола за делегирану аутентификацију и напуштање његовог претходника *OpenID 2.0*, протокола који је настао 2007. године [11]. У протоколу OpenID 2.0 потребна је екстензија како би се могућности протокола OAuth подржале, док су у OIDC протоколу те могућности интегрисане. Поред тога, овај протокол дефинише и подршку за шифровања и потписивања података за разлику од протокола OpenID 2.0. OIDC представља једноставан начин да системи који не желе да се баве комплексношћу аутентификационе логике предају одговорност неком другом систему који улаже напоре

[1]Званична веб страница организације *OpenId Foundation* је `https://openid.net/foundation/`.

за што бољом и безбеднијом аутентификацијом корисника.

С обзиром да је протокол OIDC само надоградња на протокол OAuth, све имплементације које су већ на погрешан начин користиле OAuth за аутентификацију корисника, могле су једноставно бити преправљене да користе протокол OIDC. Новине које су унете у протокол OIDC у односу на OAuth су:

- нови тип токена назван *ID токен*

  Ово је токен који носи информације о кориснику и његово креирање и довлачење се догађа у току аутентификационог процеса.

- стандардизован начин добављања информација о кориснику

  Уколико ID токен не садржи довољно информација о кориснику, систем може потражити још информација користећи API ауторизационог послужиоца. Сви ауторизациони послужиоци који подржавају OIDC протокол морају имплементирати овај API и подржавати и методе HTTP GET и методе HTTP POST. За приступ информацијама путем овог API-ја потребно је користити приступни токен добијен као одговор ауторизационог захтева. Одговор мора садржати стандардом дефинисана поља. Такве поља су нпр. *name* (пуно име, средње име и презиме корисника), *given_name* (име корисника), *family_name* (презиме корисника), *middle_name* (средње име корисника), *picture* (URL који служи за довлачење слике корисника), *nickname* (надимак корисника), *email* (имејл корисника), *email_verified* (податак да ли је корисник извршио потврду налога путем имејла), *gender* (пол корисника), *birthdate* (датум рођења корисника), *zoneinfo* (временска зона корисника), *phone_number* (број телефона корисника), *address* (адреса корисника), *updated_at* (време када су информације о кориснику последњи пут ажуриране) [14].

5.1 ID токен

Најважнија новина коју је протокол OpenId Connect увео у односу на OAuth јесте ID токен. Ово је токен који се креира и прима у току аутентификационог процеса у тренутку када се ауторизациони код замењује за приступни токен. Он у себи носи основне информације о кориснику и његовој аутентификацији. Ове информације могу бити довољне клијенту за његове потребе креирања сесије и коришћења и приказивања информација о кориснику. Уколико информације садржане у ID токену нису довољне, додатне податке је могуће довући преко стандардизованог API-ја.

Како у себи носи информације о кориснику, ID токен не може да буде обична ниска карактера као што то може бити приступни токен. Поред тога, пренос информација које он носи мора бити сигуран. Зато је одлучено да ID токен има форму токена *JWT*. Пример ID токена који има ову форму приказан је на слици 5.1. JSON Web Token, скраћено JWT, је компактан и сигуран начин путем којег

eyJqa3UiOiJodHRwczpcL1wvc2MtcWEtMDAwMS5hY2NvdW50czQwMC5vbmR
lbWFuZC5jb21cL29hdXRoMlwvY2VydHMiLCJraWQiOiJUSGwyRF9sbS1WMz
hiRk96TVVBb1Q1VVB3YW8iLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiJjYWRk
NjUyYi00YmJlLTRiZGQtOGZkMS1mMmVlN2QzMmNiYTIiLCJzdWIiOiJjYWR
kNjUyYi00YmJlLTRiZGQtOGZkMS1mMmVlN2QzMmNiYTIiLCJpc3MiOiJzYy
1xYS0wMDAxLmFjY291bnRzNDAwLm9uZGVtYW5kLmNvbSIsImV4cCI6MTYxO
Dk5ODc4MywiaWF0IjoxNjE4OTk1MTgzLCJqdGkiOiI2ZjBmMzMzZS00Y2Y1
LTQ0NWMtODc5NC0zYWZmZjc5ZTYwNmUifQ.D9SWWD2tmvjBXFo4dZXOvQtk
h4D8fXXsliojUTWCLCOtqICYJHLHM61pViitLiIxRk35lqFhYqoUNucHX_E
0MYGs4QtKbppYTQXate7GRVOwb5mL5MBijS89YrqSPZ3kHsceBv7A7IKr3o
ztvHBE5FICAS4yeL7vpxZyZAkjepGCnFYGMalPkfHKE3Olb4VTIEO55C8Jt
Y3oD0PbuFl5OgMHiVxHOMstb8_tw9pOO7qLOJuuEONSnmC1eONDEleZ0Aek
O5HUdtyWQJlGztR_3G1NjgZ8ORORC146oWG9kov0IZ5ICgmL5rdsu60DNE
cR2LIbztPLgoZ5oUpE4ZcZQ

Слика 5.1: Пример једног ID токена (црвеном бојом означено је заглавље, љубичастом тело, а плавом потпис)

се информације представљене као објекат JSON репрезентују у облику ниске карактера која се размењује између два ентитета [7]. Информације приказане у овом формату могу бити и потписане и шифроване из сигурносних разлога. JWT се састоји из неколико делова представљених base64url кодираним нискама карактера које су раздвојене тачкама. Први део је *заглавље* у коме су записане информације о алгоритму и кључевима коришћеним за потписивање или шифровање, други део је *тело* токена у коме се налазе информације које је потребно пренети и опциони трећи део је *потпис* кодираног заглавља, тачке и кодираног тела токена.

5.1.1 Тело ID токена

Основни скуп обавезних и опционих информација које тело ID токена може да садржи и које протокол OIDC дефинише су [14]:

- *iss*

  Обавезна информација која говори ко је ауторизациони послужилац који је креирао овај ID токен.

- *sub*

  Обавезна информација која говори ко је субјекат на који се ID токен односи. Вредност ове информације јесте ниска карактера која представља јединствени идентификатор корисника на страни ауторизационог послужиоца.

- *aud*

  Обавезна информација која говори ко сме да користи овај ID токен. Мора садржати један или више клијентских идентификатора. Клијент мора одбацити овај токен у процесу валидације уколико се његов клијентски идентификатор не налази у овој листи.

- *exp*

  Обавезна информација која говори када овај ID токен престаје да буде валидан. Вредност ове информације је број милисекунди протеклих од 1. јануара 1970. године до датума када овај токен истиче. Клијент мора одбацити овај токен у процесу валидације уколико је токен истекао.

- *iat*

  Обавезна информација која говори када је овај ID токен креиран. Вредност ове информације је број милисекунди протеклих од 1. јануара 1970. године до датума када је овај токен креиран.

- *nonce*

  Опциона информација која представља непромењену ниску карактера послату из аутентификационог захтева и која служи да повеже аутентификациони захтев и одговор и да онемугући тзв. нападе понављања у коме се злонамерни програм домогне већ искоришћеног ID токена и да путем њега покуша да приступи информацијама за које нема дозволу.

- *azp*

  Опциона информација која говори за кога тачно је овај ID токен креиран. Вредност ове информације је клијентски идентификатор клијента који је иницирао креирање овог токена. Ова вредност не мора бити део *aud* низа.

Када се токен са слике 5.1 декодира, добија се следеће тело:

```
{
    "aud": "cadd652b-4bbe-4bdd-8fd1-f2ee7d32cba2",
    "sub": "cadd652b-4bbe-4bdd-8fd1-f2ee7d32cba2",
    "iss": "sc-qa-0001.accounts400.ondemand.com",
    "exp": 1618998783,
    "iat": 1618995183,
    "jti": "6f0f333e-4cf5-445c-8794-3afff79e606e"
}
```

ID токен може садржати и додатне информације које нису део самог протокола OIDC. Такво је нпр. поље *jti* које означава јединствени идентификатор овог JWT-а. Потребно је да клијент зна да чита такво поље, иначе га може игнорисати. Поред тога, протокол OIDC дефинише и стандардан скуп додатних информација о кориснику које се могу захтевати као део ID токена. Овај скуп

информација дефинисан је истим пољима која се користе у API-ју за добављање информација о кориснику.

5.1.2 Заглавље и потпис ID токена

Како би пренос и коришћење информација о кориснику био сигуран, потребно је да се тело ID токена заштити. Заштита ових података врши се путем потписивања и шифровања. Информације о кључу и алгоритму коришћеним за потписивање или шифровање ID токена налазе се у његовом заглављу.

Протокол OIDC дефинише правило да ID токен мора бити потписан од стране ауторизационог послужиоца. Потписивање овог токена омогућава проверу на страни клијента да ли је токен заиста направљен од стране ауторизационог послужиоца и да ли су све информације у њему тачне. Потписивање се врши приватним, а провера потписа јавним кључем ауторизационог послужиоца. Клијент може унапред знати на којој локацији може пронаћи јавни кључ ауторизационог послужиоца коришћењем његових метаподатака који се налазе на тачно одређеној адреси. OIDC дефинише правило у коме сваки ауторизациони послужилац мора имати свој API на адреси *https://[base-server-url]/.well-known/openid-configuration* за довлачење метаподатака. Поред информација о адреси са које се могу довући токени и алгоритмима које овај ауторизациони послужилац подржава, у метаподацима се налази и информација о кључевима или референцама на кључеве. Како један ауторизациони послужилац може користити више парова јавни/приватни кључ, потребно је да у заглављу токена нагласи који кључ треба користити за проверу потписа. Идентификатор кључа који је потребно користити се у токену дефинише као *kid* поље. Када се токен са слике 5.1 декодира, добија се следеће заглавље:

```
{
    "jku": "https://sc-qa-0001.accounts400.ondemand.com/oauth2/certs",
    "kid": "TH12D_lm-V38bFOzMUAoT5UPwao",
    "alg": "RS256"
}
```

Поље *jku* заглавља представља адресу на којој се може наћи јавни кључ са идентификатором *kid*. Ова адреса није обавезна с обзиром да она мора бити и део метаподатака ауторизационог послужиоца.

ID токен може бити и потписан, па шифрован, што има за последицу угњежден токен JWT. Шифровање потписаног токена се врши јавним, а дешифровање приватним кључем клијента. Ауторизациони послужилац користи јавни кључ дефинисан од стране клијента приликом његове регистрације на страни ауторизационог послужиоца[2].

[2]Поред мануелне регистрације, OIDC протокол дефинише и правила за могућу динамичку регистрацију клијента. Више информација о динамичкој регистрацији могуће је пронаћи на адреси https://openid.net/specs/openid-connect-registration-1_0.html.

5.2 Аутентификација

Процес аутентификације у оквиру протокола OIDC се не разликује много од процеса делегирања ауторизације у оквиру протокола OAuth описаног у глави 4. Постоји три начина аутентификације која се разликују по типу одговора који клијент очекује, као и у протоколу OAuth. То су:

1. *Ток ауторизационог кода* (енгл. *authorization code flow*)
2. *Имплицитни ток* (енгл. *implicit flow*)
3. *Хибридни ток* (енгл. *hybrid flow*)

5.2.1 Ток ауторизационог кода

Као и код протокола OAuth, у овом току одговор аутентификационог захтева није приступни токен, већ ауторизациони код. Замена ауторизационог кода за приступни токен и ID токен врши се у следећем кораку, након добијања ауторизационог кода. Предност оваквог приступа је неоткривање приступног токена и ID токена веб прегледачу и свим злонамерним програмима који имају приступ њему.

Обавезни параметри аутентификционог захтева наслеђени од протокола OAuth су *client_id* (клијентски идентификатор), *redirect_url* (адреса за редирекцију, односно адреса на којој одговор треба да се достави), *response_type* (врста ауторизационе дозволе која мора имати вредност *code*), *scope* (опсег који мора садржати вредност *openid*). Поред ових информација, клијент у свом аутентификационом захтеву може послати још неке параметре дефинисане протоколом OIDC. Неки од њих су [14]:

- *nonce*

  Ниска карактера која служи да повеже аутентификациони захтев и одговор. Прослеђује се непромењена из аутентификационог захтева до ID токена.

- *prompt*

  Ниска карактера у којој су вредности раздвојене белином. Говори ауторизационом послужиоцу да ли да прикаже прозор за реаутентификацију и давање дозволе кориснику. Могуће вредности садржане у овој ниски карактера су: *none* (ауторизациони послужилац не сме да прикаже никакве прозоре, а уколико корисник није претходно аутентификован или није дао дозволу, производи грешку), *login* (ауторизациони послужилац би требало да затражи поновну аутентификацију од корисника уколико је већ аутентификован), *consent* (ауторизациони послужилац би требало да затражи дозволу од корисника пре било каквог слања одговора клијенту) и *select_account* (ауторизациони послужилац би требало да прикаже кориснику листу налога са којима може да се аутентификује).

- *max_ age*

  Максимално трајање сесије на страни ауторизационог послужиоца после којег он мора радити реаутентификацију корисника.

- *id_ token_ hint*

  ID токен претходно добијен од стране ауторизационог послужиоца. Прослеђује се као наговештај ауторизационом послужиоцу о тренутној или прошлој сесији на страни клијента. Уколико корисник из ID токена није аутентификован на страни ауторизационог послижиоца, производи се грешка.

Размена ауторизационог кода за приступни токен врши се исто као и у OAuth протоколу. Једина разлика је што се поред приступног токена у одговору добија и ID токен чије се информације могу користити за прављење сесије на страни клијента. Клијент је у обавези да валидира добијени ID токен пре него што направи сесију.

5.2.2 Имплицитни ток

У овом току одговор аутентификационог захтева је директно приступни токен и ID токен или само ID токен. Као и код протокола OAuth, користе га обично клијенти који немају свог послужиоца, већ су направљени коришћењем неког скриптног језика и извршавају се у оквиру веб прегледача. Аутентификациони захтев у овом току се разликује од тока ауторизационог кода само у вредностима два параметра. Параметар *response_ type* може имати само вредност *id_ token token* или само *id_ token*. Уколико има вредност *id_ token*, аутентификациони одговор ће садржати само ID токен, а ако има вредност *id_ token token*, одговор ће поред ID токена садржати и приступни токен. Параметар *nonce* је обавезан у имплицитном току.

5.2.3 Хибридни ток

У овом току, неки токени се добијају као одговор на аутентификациони захтев, а неки се могу потражити као и у току ауторизационог кода. Зове се хибридни ток јер комбинује ток ауторизационог кода и имплицитни ток. Користан је код клијената који користе само ID токен и желе да одмах провере вредност *nonce* параметра како би избегли више позива ка ауторизационом послужиоцу уколико овај параметар није валидан. Параметар *response_ type* аутентифкационог захтева може имати вредности *code id_ token*, *code token* и *code id_ token token*. ID токен послат као одговор аутентификационог захтева има смањен број информација о кориснику јер је његова вредност изложена веб прегледачу и могућим злонамерним програмима. Такође, приступни токен добијен као одговор аутентификационог захтева може имати веома кратко време трајања како би се избегла његова злоупотреба.

5.3 Добре и лоше стране протокола OIDC

Све што је наведено као предност протокола OAuth јесте и предност протокола OIDC с тиме да он нуди додатну флексибилност с обзиром да подржава и слање додатних информација о кориснику у оквиру ID токена.

Једна од мана OIDC-а, као и код свих SSO протокола, је што се са једним налогом може приступити многим апликацијама и информацијама које оне чувају. Уколико се злонамерни програм домогне налога на ауторизационом послужиоцу, све информације чуване за тај налог од стране свих регистрованих клијената на том ауторизационом послужиоцу су доступне злонамерном програму. Друга мана овог протокола је то што је за избегавање многих напада потребно детаљно пратити спецификацију приликом имплементације. На жалост, врло често се приликом имплементације изоставе неке од потребних провера, чиме апликација постаје рањива на поједине врсте напада.

Глава 6

Подршка SSO протокола у оквиру радног оквира .Net Core

Сви претходно описани SSO протоколи налазе своју примену у веб апликацијама. За креирање веб апликација користе се многи радни оквири који подржавају различите програмске језике. *Веб радни оквир* је окружење у коме је развој веб апликација поједностављен због могућности коришћења универзалног кода који долази са окружењем. Веб радни оквири имају уграђен компајлер и различите библиотеке које поједностављују и скраћују време имплементације. Такве библиотеке су нпр. оне које омогућавају једноставно конектовање и рад са базом података, читање докумената или креирање API метода. Један од широко коришћених и актуелних веб радних оквира јесте *.Net Core*. Он подржава програмски језик *C#* и његов код се може извршавати на свим платформама. Због његове велике популарности и употребљивости, радни оквир .Net Core је употребљен у овом раду да демонстрира примену и ниво комплексности поменутих SSO протокола у веб апликацијама.

6.1 .Net Core и идентитет корисника

Радни оквир .Net Core садржи уграђену подршку за рад са аутентификацијом корисника. Скуп метода које омогућавају ове функционалности назива се *.Net Core идентитет* (енгл. *.Net Core Identity*). Поред саме аутентификације, .Net Core идентитет подржава и функционалности попут чувања информација о кориснику, лозинки, дозвола, информација о потврђености идентитета преко имејла и закључавање корисника. Како би се користиле све ове функционалности, потребно је проћи кроз неколико корака. У већ креираном веб пројекту који користи базу SQL Server треба:

1. Променити да апликациони базни контекст наслеђује класу која пружа све што је потребно за рад са идентитом корисника у контексту базе уместо основне класе *DbContext* из простора имена *Microsoft.EntityFrameworkCore*.

Класа која пружа такав скуп функционалности назива се *IdentityDbContext* из простора имена *Microsoft.AspNetCore.Identity.EntityFrameworkCore*.

```
public class ApplicationDbContext : IdentityDbContext
{
    public ApplicationDbContext(
        DbContextOptions<ApplicationDbContext> opts)
        : base(opts)
    {
    }
}
```

2. Регистровати све сервисе које .Net Core идентитет користи. Регистровање се врши у оквиру фајла *Startup.cs*, у оквиру методе *ConfigureServices*. За регистровање се може користити предефинисана класа *IdentityUser* из простора имена *Microsoft.AspNetCore.Identity* које служи као модел за складиштење података о корисницима. Уколико су потребна још нека поља која чувамо као део корисниковог идентитета, можемо креирати нову класу која ће наслеђивати класу *IdentityUser* и користити је у оквиру методе *AddDefaultIdentity*. Ова метода и конфигурише аутентификацију да користи колачиће и додаје предефинисане странице које се појављују кориснику приликом пријављивања на систем, регистрације или промене лозинке.

```
// services је колекција типа IServiceCollection
services.AddDefaultIdentity<IdentityUser>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
}
```

3. Додати посреднички софтвер (енгл. *middleware*) за аутентификацију у ток процесирања захтева у оквиру методе *Configure* у фајлу *Startup.cs*.

```
// app је интерфејс типа IApplicationBuilder
 app.UseAuthentication();

 app.UseEndpoints(endpoints =>
 {
     endpoints.MapRazorPages();
 });
```

4. Додати миграцију која генерише све табеле у бази података које се користе у .Net Core идентитету. Табеле које се генеришу су *User* (табела која чува кориснике), *Role* (табела која чува улоге као што су администратор или запослени), *UserClaim* (табела која чува права која корисник поседује), *UserToken* (табела која чува аутентификационе токене), *UserLogin* (табела која чува аутентификације корисника), *RoleClaim* (табле која чува права која су загарантована свим корисницима са одређеном улогом), *UserRole* (табела која чува однос између корисника и улога). Ово је могуће урадити позивањем команди:

```
dotnet ef migrations add AddingIdentity
dotnet ef database update
```

Овако креирана апликација омогућава регистровање нових корисника и локалну аутентификацију. Поред ових основних корака који су приказани за омогућавање аутентификације, могуће је конфигурисати и додатна подешавања као што су својства колачића, јачине лозинки, закључавање корисника и многе друге.

6.2 .Net Core идентитет и аутентификација коришћењем познатих спољних аутентификационих добављача

Поред локалне аутентификације, .Net Core идентитет има уграђену подршку и за аутентификацију коришћењем познатих и често коришћених спољних аутентификационих добављача као што су Фејсбук, Гугл, Твитер, Мајкрософт. За имплементацију овакве аутентификације коришћен је протокол OAuth 2.0 [17]. Како би се користила ова уграђена функционалност, потребно је да се поред регистровања аутентификационог посредничког софтвера додају и сви спољни аутентификациони добављачи у оквиру методе *ConfigureServices* у фајлу *Startup.cs* на следећи начин:

```
services.AddAuthentication()
    .AddGoogle(googleOptions => { ... })
    .AddMicrosoftAccount(microsoftOptions => { ... })
    .AddTwitter(twitterOptions => { ... })
    .AddFacebook(facebookOptions => { ... });
```

Коришћене спољне аутентификационе методе могу се наћи у .NET Core уграђеним пакетима:

- *Microsoft.AspNetCore.Authentication.MicrosoftAccount*
- *Microsoft.AspNetCore.Authentication.Google*
- *Microsoft.AspNetCore.Authentication.Facebook*
- *Microsoft.AspNetCore.Authentication.Twitter.*

Да би се користила овако дефинисана аутентификација, потребно је додати код који преусмерава корисника на страну аутентификационог добављача и чита аутентификациони одговор. Преусмеравање корисника и читање аутентификационог одговора се имплементира као генерички код који се не разликује за више аутентификационих добављача. Да би се приказао цео процес рада са спољним аутентификационим добављачима у оквиру радног оквира .Net Core, приказаћемо пример конфигурисања аутентификације коришћењем аутентификационог добављача Гугл.

6.2.1 Аутентификација коришћењем аутентификационог добављача Гугл

Пре саме имплементације у оквиру радног оквира .NET Core, потребно је регистровати своју апликацију код аутентификационог добављача Гугл[1]. Свака регистрована апликација добија свој клијентски идентификатор и клијентску тајну које се користе у аутентификационом процесу OAuth 2.0. Добијени клијентски идентификатор и тајну треба убацити у дефинисане опције аутентификационог добављача Гугл у фајлу *Startup.cs* на следећи начин:

```
services.AddAuthentication()
    .AddGoogle(googleOptions => {
        googleOptions.ClientId = "1234987819200.apps.googleusercontent.com";
        googleOptions.ClientSecret = "f5bb16f7-ec44-4756-b776-987316cfe1a0";
    });
```

Следећи корак је приказ свих регистрованих аутентификационих добављача на страни за пријављивање. Тражену листу је могуће добити помоћу сервиса *SignInManager* регистрованог у простору имена *Microsoft.AspNetCore.Identity*. Потребно је да модел странице за пријављивање дефинише променљиву *ExternalLogins* која чува листу спољних аутентификационих добављача иницијализовану помоћу методе *GetExternalAuthenticationSchemesAsync* дефинисане у класи *SignInManager*. Пример иницијализовања листе приликом довлачења странице за пријављивање:

```
public async Task OnGetAsync()
{
    if (!string.IsNullOrEmpty(ErrorMessage))
    {
        ModelState.AddModelError(string.Empty, ErrorMessage);
    }

    ExternalLogins =
        (await _signInManager.GetExternalAuthenticationSchemesAsync())
        .ToList();
}
```

Пример кода који приказује листу крајњем кориснику и преусмерава га на изабраног аутентификационог добављача приликом његовог одабира:

```
<form id="external-account"
 asp-page="./ExternalLogin"
 method="post"
 class="form-horizontal">
    <div>
        <p>
            @foreach (var provider in Model.ExternalLogins)
```

[1]Више информација се може пронаћи на адреси https://developers.google.com/identity/protocols/oauth2/openid-connect#getcredentials.

```
            {
                <button type="submit"
                 class="btn btn-primary"
                 name="provider"
                 value="@provider.Name"
                 title="Log in using your @provider.DisplayName account">
                     @provider.DisplayName
                </button>
            }
        </p>
    </div>
</form>
```

Логика преусмеравања корисника на изабраног аутентификационог добављача се дефинише на страници *ExternalLogin*, као што је дефинисано атрибутом *asp-page* приказаног елемента *form*. У методи која обрађује ову форму, потребно је сервису *SignInManager* нагласити којег аутентификационог добављача треба да користи приликом аутентификације. Ово је могуће урадити помоћу методе *ConfigureExternalAuthenticationProperties* која као параметре прима име изабраног добављача и URI на који овај сервис треба да врати одговор аутентификације. Пример кода који ради претходно описану логику у оквиру модела *ExternalLogin*:

```
public IActionResult OnPost(string provider)
{
    string redirectUrl =
        Url.Page("./ExternalLogin", pageHandler: "Callback");

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

    return new ChallengeResult(provider, properties);
}
```

Враћањем одговора *ChallengeResult*, радни оквир .Net Core зна да је потребно извршити преусмеравање на аутентификакционог добављача, извршити аутентификацију и вратити одговор методи *OnGetCallbackAsync* у оквиру модела *ExternalLogin*. Приликом ове редирекције, извршава се цео процес OIDC у коме долази до размене ауторизационог кода за приступни токен и ID токен. Метода *OnGetCallbackAsync* би требало да прибави одговор аутентификационог добављача и да провери да ли је дошло до неке грешке приликом аутентификације. Уколико јесте, потребно је вратити корисника на страницу за пријављивање, а уколико нема грешке, направити сесију.

```
public async Task<IActionResult> OnGetCallbackAsync(string remoteError = null)
{
    if (remoteError != null)
    {
```

```
        ErrorMessage = $"Error from external provider: {remoteError}";
        return RedirectToPage("./Login");
    }

    ExternalLoginInfo info = await _signInManager.GetExternalLoginInfoAsync();
    if (info == null)
    {
        ErrorMessage = "Error loading external login information.";
        return RedirectToPage("./Login");
    }

    SignInResult result =
       await _signInManager
       .ExternalLoginSignInAsync(
          info.LoginProvider, info.ProviderKey, isPersistent: false
        );

    if (result.Succeeded)
    {
        return RedirectToPage("./Home");
    }
    else
    {
        string email = info.Principal.FindFirstValue(ClaimTypes.Email);
        if (email != null)
        {
            var user = await _userManager.FindByEmailAsync(email);

            await _userManager.AddLoginAsync(user, info);
            await _signInManager.SignInAsync(user, isPersistent: false);

            return LocalRedirect(returnUrl);
        }
    }

    return RedirectToPage("./CreateAccount");
}
```

Прибављање аутентификационог одговора могуће је извршити помоћу методе *GetExternalLoginInfoAsync* која припада сервису *SignInManager*. Одговор садржи прибављене токене у својој променљивој *AuthenticationTokens*, а парсирани подаци о кориснику довучени помоћу стандардизованог OIDC API-ја налазе у оквиру променљиве *Principal*. Прављење сесије могуће је извршити помоћу методе *ExternalLoginSignInAsync* сервиса *SignInManager*. Ова метода ће направити колачић само уколико за одговарајућег корисника постоји ред у табели *AspNetUserLogins* дефинисаној од стране .Net Core идентитета. То је табела која повезује корисника из табеле *AspNetUsers*[2] са одређеним аутентификационим добављачем. У нашем случају, потребно је да постоји ред у табели који повезује

[2]Табела *AspNetUsers* чува све регистроване кориснике и дефинисана је од стране .Net Core идентитета.

аутентификационог добављача Гугл са тренутним корисником. Уколико ред у овој табели не постоји, потребно је проверити да ли тренутни корисник постоји у табели *AspNetUsers*. Уколико постоји, потребно је само додати ред у табели *AspNetUserLogins* који ће се убудуће користити за корисниково пријављивање помоћу аутентификационог добављача Гугл. Ако ипак корисник није направио налог и не налази се у *AspNetUsers* потребно је преусмерити га на страницу за креирање налога.

Радни оквир .Net Core пружа и могућност коришћења предефинисаних страница за пријављивање и креирање корисника које се могу изменити по жељи. Сва претходно дефинисана логика приказавања листе спољних аутентификационих добављача, преусмеравања на изабраног добављача и обрада одговора једна је од функционалности које пружају предефинисане странице и њихови модели.

6.3 .Net Core идентитет и аутентификација коришћењем протокола OpenId Connect

Поред уграђене подршке за аутентификацију помоћу познатих и често коришћених спољних аутентификационих добављача, радни оквир .Net Core пружа могућност и дефинисања било ког спољног OIDC аутентификационог добављача. То је могуће урадити помоћу методе *AddOpenIdConnect* из простора имена *Microsoft.AspNetCore.Authentication.OpenIdConnect* у оквиру методе *ConfigureServices* у фајлу *Startup.cs* на следећи начин:

```
services
   .AddAuthentication(options =>
   {
       options.DefaultScheme =
           CookieAuthenticationDefaults.AuthenticationScheme;
       options.DefaultChallengeScheme =
           JwtBearerDefaults.AuthenticationScheme;
   })
   .AddOpenIdConnect(options =>
   {
       options.SignInScheme =
           CookieAuthenticationDefaults.AuthenticationScheme;
       options.Authority = "https://accounts.google.com";
       options.Configuration =
           new OpenIdConnectConfiguration();
       options.Configuration.AuthorizationEndpoint =
           "https://accounts.google.com/o/oauth2/v2/auth";
       options.Configuration.TokenEndpoint =
           "https://oauth2.googleapis.com/token";
       options.Configuration.UserInfoEndpoint =
           "https://openidconnect.googleapis.com/v1/userinfo";
       options.ClientId = "1234987819200.apps.googleusercontent.com";
       options.ClientSecret = "secret";
       options.SaveTokens = true;
```

```
        options.ResponseType = "code";
        options.GetClaimsFromUserInfoEndpoint = true;
        options.Scope.Add("openid");
        options.Scope.Add("profile");
    });
```

Да би аутентификација радила, потребно је дефинисати клијентски идентификатор и тајну, као и код познатих спољних аутентификационих добављача. Поред тога, потребно је и навести које URI-је користити за аутентификацију, довлачење токена и информација о кориснику. Поред ових опција, постоји и дефинисање опсега, типа одговора, да ли приликом аутентификације желимо довлачење информација о кориснику, или је довољан ID токен и још многе друге.

6.4 .Net Core идентитет и аутентификација коришћењем протокола SAML

Радни оквир .Net Core нема уграђену подршку за аутентификациони протокол SAML. Иако уграђена подршка не постоји, други појединци су направили пакете који се могу инсталирати и користити за SAML аутентификацију. Неки од познатих пакета су *ITfoxtec Identity SAML 2.0*[3] и *Sustainsys.Saml2*[4]. Оба пакета су отвореног кода и подржавају креирање и валидацију потписа, дешифровање аутентификационих порука као и обједињено одјављивање. Пакет *ITfoxtec Identity SAML 2.0* је мање коришћен од пакета *Sustainsys.Saml2*, али *Sustainsys.Saml2* има пуно отворених проблема пријављених од стране корисника пакета[5]. Неки од проблема су валидација потписа који садржи нове редове или грешке приликом аутентификације изазване од стране добављача идентитета. Пакет *ITfoxtec Identity SAML 2.0* нема уграђену подршку за дефинисање више од једног добављача идентитета, док пакет *Sustainsys.Saml2* подржава дефинисање више од једног добављача идентитета само уколико је дефинисано и више пружалаца услуга.

Као део овог рада имплементирана је библиотека која подржава креирање и валидацију потписа, дешифровање аутентификационих порука и регистрацију више добављача идентетета за једног пружаоца услуга. Детаљи имплементације ове библиотеке и начин коришћења биће приказани у наредној глави.

[3]Пакет *ITfoxtec Identity SAML 2.0* може се наћи на адреси https://www.nuget.org/packages/ITfoxtec.Identity.Saml2/.

[4]Пакет Sustainsys.Saml2 може се наћи на адреси https://www.nuget.org/packages/Sustainsys.Saml2/.

[5]Отворени проблеми пакета *Sustainsys.Saml2* се могу наћи на њиховој GitHub страници https://github.com/Sustainsys/Saml2/issues.

Глава 7

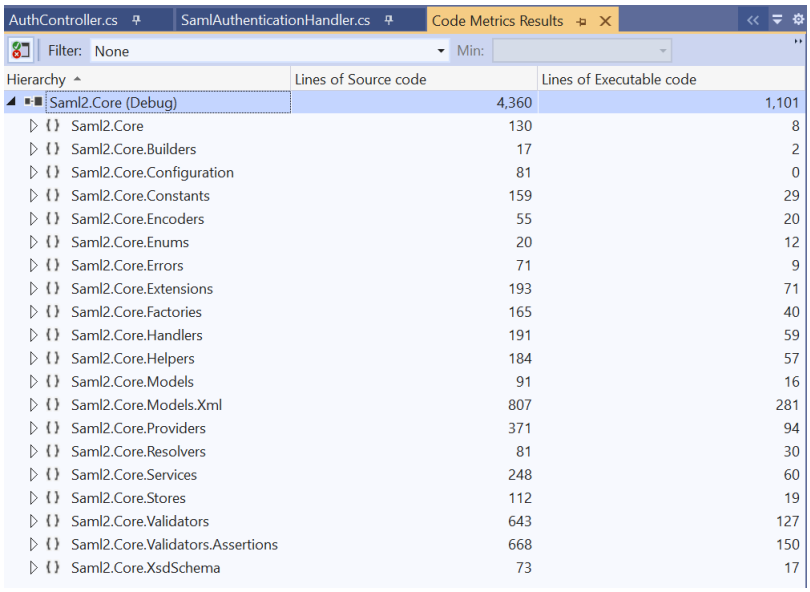
.Net Core SAML библиотека

Како радни оквир .Net Core нема уграђену подршку за аутентификациони протокол SAML, као део овог рада имплементирана је и библиотека која врши аутентификацију корисника путем протокола SAML за апликације написане у радном оквиру .Net Core. Библиотека је јавно доступна као *Nuget* пакет под називом *Core.Saml2.Authentication* и може се наћи на адреси `https://www.nuget.org/packages/Core.Saml2.Authentication`. Библитека се ослања на функционалности уграђеног .Net Core аутентификационог посредничког софтвера тако што наслеђује његову класу *AuthenticationHandler* у којој су дефинисане методе које служе за обраду аутентификационих захтева и прављења сесије. Да би библиотека била у могућности да функционише, потребно је унети податке о SAML пружаоцу услуга и регистровати аутентификационе добављаче. Функционалности које библиотека подржава су:

- Креирање аутентификационог захтева путем HTTP везе преусмеравања
- Примање аутентификационог одговора путем HTTP POST везе
- Потписивање аутентификационог захтева помоћу приватног кључа који представља један од унетих података о пружаоцу услуга
- Валидација и обрада аутентификационог одговора, што укључује и дешифровање заштићених елемената и проверу потписа целог одговора или тврђења
- Креирање сесије уз помоћ информација добијених у аутентификационом одговору и на начин који је дефинисан као подразумеван у оквиру подешавања аутентификационог посредничког софтвера

Библиотека има укупно 4360 линија кода. Анализа укупног броја линија кода и извршних линија подељених по простору имена у коме се налазе приказана је на слици 7.1.

Креирана библиотека омогућава аутентификацију коришћењем протокола SAML тако што се региструје спољни аутентификациони добављач на исти начин као што је описано у поглављу 6.3 где се користила уграђена подршка за протокол OpenId Connect. Ова функционалност је омогућена путем креиране екстензионе методе *AddSaml* над класом *AuthenticationBuilder*. Имплементација ове



Слика 7.1: Анализа укупног броја линија кода и извршних линија подељених по простору имена у коме се налазе у оквиру библиотеке *Core.Saml2.Authentication*

екстензионе методе приказана је на слици 7.2. Ова метода прима делегат са параметаром који је објекат класе *SamlAuthenticationOptions* и који у себи носи подешавања везана за аутентификационог добављача као што су јединствени идентификатор, URL на који је потребно послати аутентификациони захтев, путања на којој се може наћи јавни кључ који ће се користити приликом провере потписа, страница на коју треба преусмерити корисника након успешне аутентификације, параметар који показује да ли добављач захтева потпис аутентификационог захтева и мапирање имена атрибута добијених у аутентификационом одговору у атрибуте који су део креиране сесије. Ова метода региструје нову аутентификациону схему помоћу методе *AddScheme* која је део класе *AuthenticationBuilder*. Овако регистровани спољни аутентификациони добављач користи аутентификацију дефинисану у класи *SamlAuthenticationHandler* која је део креиране библиотеке и која наслеђује поменуту уграђену класу *AuthenticationHandler*. Приликом регистрације нове схеме, региструје се и класа која ће проверавати да ли су унета подешавања валидна, односно да ли се су дефинисани сви обавезни подаци. Пример регистровања једног SAML аутентификационог добављача који користи колачиће за креирање сесије приказан је на слици 7.3. Библиотека подржава регистровање више аутентификационих добављача, што се може постићи низањем метода *AddSaml*.

```
0 references | Milica Djuric, 348 days ago | 1 author, 1 change
public static class AuthenticationBuilderExtension
{
    0 references | 0 changes | 0 authors, 0 changes
    public static AuthenticationBuilder AddSaml(
        this AuthenticationBuilder builder,
        Action<SamlAuthenticationOptions> options
    )
    {
        return builder.AddSaml(AuthenticationSchemeConstant.Name, options);
    }

    3 references | 0 changes | 0 authors, 0 changes
    public static AuthenticationBuilder AddSaml(
        this AuthenticationBuilder builder,
        string authenticationSchemeName,
        Action<SamlAuthenticationOptions> options
    )
    {
        return builder.AddSaml(authenticationSchemeName, AuthenticationSchemeConstant.DisplayName, options);
    }

    1 reference | Milica Djuric, 348 days ago | 1 author, 1 change
    public static AuthenticationBuilder AddSaml(
        this AuthenticationBuilder builder,
        string authenticationScheme,
        string displayName,
        Action<SamlAuthenticationOptions> options
    )
    {
        builder.Services.TryAddEnumerable(
            ServiceDescriptor.Singleton<
                IValidateOptions<SamlAuthenticationOptions>,
                SamlAuthenticationOptionValidation>()
            );

        builder.Services.Configure(authenticationScheme, options);

        AuthenticationBuilder authenticationBuilder =
                builder.AddScheme<SamlAuthenticationOptions, SamlAuthenticationHandler>(
                    authenticationScheme,
                    displayName,
                    options
                );

        return authenticationBuilder;
    }
}
```

Слика 7.2: Имплементација екстензионе методе *AddSaml* у оквиру библиотеке *Core.Saml2.Authentication*

```
services
    .AddAuthentication(options =>
    {
        options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        options.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;
        options.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    })
    .AddCookie(options =>
    {
        options.LoginPath = "/auth/login";
    })
    .AddSaml("okta", options =>
    {
        options.IdentityProviderConfiguration = new IdentityProviderConfiguration()
        {
            EntityId = "http://www.okta.com/exk19fnc2urbp2lBa5d7",
            HttpRedirectSingleSignOnService =
                "https://dev-7375101.okta.com/app/dev-7375101_milicatest_1/exk19fnc2urbp2lBa5d7/sso/saml",
            PublicKeyPath = "./Certs/idpPublicKey.cert",
            AuthnReturnUrl = "/Home",
            WantAuthnRequestsSigned = true,
            UserAttributeMapping = new UserAttributeMapping()
            {
                Email = "Email",
                FirstName = "FirstName",
                LastName = "LastName"
            }
        };
    });
```

Слика 7.3: Пример регистровања спољног SAML аутентификационог добављача

Да би библиотека била спремна за исправно коришћење, потребно је у оквиру фајла *appsettings.json* дефинисати параметре везане за SAML пружаоца услуга. Параметри које је потребно дефинисати су јединствени идентификатор, URL на који је потребно да стигне аутентификациони одговор, параметар који показује да ли ће се аутентификациони захтев потписивати, којим алгоритмом ће се потписивање вршити и путања на којој се може наћи приватни кључ који ће се користити за потписивање. Пример дефинисања конфигурације SAML пружаоца услуга приказан је на слици 7.4. Библиотека подржава конфигурисање само једног пружаоца услуга.

```
"Saml": {
  "ServiceProviderConfiguration": {
    "EntityId": "https://localhost:44369",
    "AuthnResponseEndpoint": "/saml/login/receive",
    "AuthnRequestsSigned": true,
    "AuthnRequestSigningAlgorithm": "RSA-SHA256",
    "PrivateKeyFilePath": "./Certs/key.pem"
  }
}
```

Слика 7.4: Пример дефинисања конфигурације SAML пружаоца услуга

7.1 Креирање аутентификационог захтева

Креирање аутентификационог захтева дешава се у оквиру методе *HandleChallengeAsync* која је део класе *SamlAuthenticationHandler*. Ова метода биће позвана сваки пут када апликација која користи библиотеку као резултат неке API методе врати *ChallengeResult* који у себи садржи SAML аутентификационог добављача. *ChallengeResult* се враћа када апликација не зна који корисник покушава да приступи неком ресурсу. Библиотека користи подешавања аутентификационог добављача да види на коју адресу треба да пошаље аутентификациони захтев и да ли треба да га потпише. Крајњи резултат ове методе *HandleChallengeAsync* јесте преусмеравање корисника на страницу за аутентификацију која је део аутентификационог добављача. Имплементација методе *HandleChallengeAsync* приказана је на слици 7.5. Методи *HandleChallengeAsync* може бити прослеђен објекат *AuthenticationProperties* који у себи носи информацију на коју страницу се треба вратити након успешно извршене аутентификације. Добијена информација се прослеђује методи *CreateRedirectUrl* класе *AuthnRequestService* у којој се дешава читаво креирање аутентификационог захтева и URL-а који ће одвести корисника на страницу за пријављивање аутентификационог добављача. Имплементација креирања аутентификационог захтева приказана је на слици 7.6. Метода *CreateRedirectUrl* у оквиру класе *AuthnRequestService* креира аутентификациони захтев, извлачи из подешавања о пружаоцу услуга информације о томе да ли захтев треба да буде потписан и

```
0 references | Milica Djuric, 22 days ago | 2 authors, 4 changes
protected override async Task HandleChallengeAsync(AuthenticationProperties properties)
{
    properties ??= new AuthenticationProperties();

    IIdpConfigurationProvider idpConfigProvider =
        this.idpConfigurationProviderFactory.Create(Options);

    this.Logger.LogInformation(
        $"Started with creating authentication request for {idpConfigProvider.GetEntityId()} idp."
    );

    // should create auth request here
    if (idpConfigProvider.GetAuthnRequestBinding() == Enums.BindingType.HttpRedirect)
    {
        string redirectUrl = await this.authnRequestService.CreateRedirectUrl(properties);

        Context.Response.Redirect(redirectUrl);
    }

    return;
}
```

Слика 7.5: Имплементација методе *HandleChallengeAsync* у оквиру библиотеке *Core.Saml2.Authentication*

који алгоритам је потребно користити за то и добијену информацију прослеђује класи која креира URL за преусмеравање. Такође, овде се креира и објекат *RelayStateData* који се прослеђује аутентификационом добављачу и који мора бити враћен потпуно исти у аутентификационом одговору. На овај начин чувамо информацију о томе на коју страницу је потребно преусмерити корисника након успешне аутентификације. Ова метода ради још једну битну ствар, а то је чување креираног аутентификационог захтева како би се у валидацији одговора могао проверити атрибут *InResponseTo* и уверити се да је добијени одговор заиста одговор на захтев креиран од стране ове библиотеке.

Пример једног потписаног аутентификационог захтева креираног од стране библиотеке помоћу конфигурације приказане у овом поглављу је:

https://dev-7375101.okta.com/app/dev-7375101_milicatest_1/exk19fnc2urbp2lBa5d7/sso/saml
?SAMLRequest=fVLJbsIwEP2VyPcQJw5ZLIgE5VAkqiJIe%2BgFmWQCFomdehzE5zeEVqWHIs1lljfvzTJB0dQt
n3X2qDbw2QFa59LUCvmQmJLOKK4FSuRKNIDcFnw7e1nxYER5a7TVha6Js1xMyS5NIwCflS6L%2FcoNGVRumrLKT
RK296mfhoEfEecdDEqtpqTv0AMRO1gqtELZPkQD36WJS8c5pTxgvY1CRsMkiT6Is%2BjFSSXsgD5a2yL3vBLObs
zicU8w0icrRoVuPNG294ldI2tZCNvjd74Hl5OfVqoIOrNvg3ouxmXsIWrvOjFx1t9DzaUqpTo83sD%2BVoT8Oc%
2FX7vp1mxNnhgjmKvJJK%2BwaMFswZ1nA22b1K7vWhaiPGi0PQxalA3cfPEjlGShAnoFkw234sCFzd5THisQPO8
n%2B4Zp4d32zm%2Ff3A7Iv1b6yOQ%3D%3D&RelayState=lY%2FBCoNADET%2FJWdp70IpUnoQehDBD9iusS7GX
cnGVhD%2FvVEp9Opt8pKZJDOUKCP7iglSOGejtOdHeDl%2FM0RPY7srToLsDRUc3q5GvoRODCSwjqIXZ4244LU7
IIvDCOkMuWC%2FiRNj7RitHAzfpn5QvRteEigMmx7Vs6ZrncdCtYuil0DaGIqY6Ef7zordwbV5jCPWlVhI%2FUi
UwH0aNCr%2BkYwofEpslLY7W5YvW9h2Tw%3D%3D&SigAlg=http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxm
ldsig-more%23rsa-sha256&Signature=TGxDtIKeFtlqgE6uwS%2BBmgYWEX%2BVP8lFiGilqPrs5x1abCkd4
h0%2F72%2BoN3qOVIMuDz2WxaZz7m0z7Hq7EinOMY1QhOySt7GsTuYgJNjnyLGVWHzRrgPQuc6nGItRRZBoCWdN
Js24Pr9ExjLyiNuLuqjZH99ABPP7nK12IlVfvqp3dAyD2S8cU4doSlRYGiVPFr15W5zdUi1OuCAaMgJNr7%2BYF
TPBRdP6nu%2FvVO21cWecEBMZ%2Fi9JgeCYCXjOayN4STxiMuRObY%2F1suFmRQ5Ume0c9nVIFgC81jZKIkMQ8u
AZbgHX2Tha31ZOG25W8Jkc6UiBVVq%2F3eDnQ9oRqLrpmEha%2FhGakgGos%2F9E%2B6of8m8OZQ%2FXaq7bJDg
cZFk5NyOA3wA6Voyi6lXxPNvZHZ9XlD5P53Tp1ENzFOLU3FZhqUpmtEWCsO5JvlSnyh6fy7yn3nVphLQH9%2F%2
FJsXQ4PDtZb5Qtm4O53G9Vu6iqBvjuiBfH2rr1DhivYxLaGjcpaYQtX9wwJcc1yf7b8RR6x6DYOgiOQVJawl000
5Jfnu55BtrGX4OxBSny86WKrAPJzENsH3px8OHpQ8EoY5q1W2cM%2BJ%2B72%2BqYqhjLbTZ7pSPzYCk329XNwS
2MRVGKX6xjAAlfDbt9pE9YsaviqkL9u1PyrEP7CNGXsYa4cqTNZ3NeoTw%3D

```
2 references | Milica Djuric, 22 days ago | 1 author, 1 change
public async Task<string> CreateRedirectUrl(AuthenticationProperties authenticationProperties)
{
    AuthnRequest request = this.authnRequestFactory.Create();

    string requestXml = this.authnRequestXmlProvider.Get(request);

    bool signRequest = this.spConfigurationProvider.GetAuthenticationRequestSigned();

    SignatureAlgorithm signatureAlgorithm =
        signRequest
        ? this.spConfigurationProvider.GetAuthenticationRequestSigningAlgorithm()
        : null;

    RelayStateData relayStateData =
        new RelayStateData() {
            ReturnUrl = authenticationProperties.RedirectUri,
            AuthenticationProperties = authenticationProperties
        };

    SamlRedirectData samlRedirectData = this.samlRedirectDataFactory.Create(
        request.Destination,
        requestXml,
        CorrespondenceType.Request,
        signRequest,
        JsonSerializer.Serialize(relayStateData),
        signatureAlgorithm
    );

    await this.authnRequestStore.Insert(request.Id);

    return samlRedirectData.ToRedirectUrl();
}
```

Слика 7.6: Имплементација креирања аутентификационог захтева у оквиру библиотеке *Core.Saml2.Authentication*

Након извршеног URL, *base64* и *inflate* декодирања, можемо видети да послати захтев има следећу XML форму:

```
<samlp:AuthnRequest
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  ID="_996ee13d-371f-43ef-993f-883b10194216"
  Version="2.0"
  IssueInstant="2021-08-05T00:23:23.4304886Z"
  Destination="https://dev-7375101.okta.com/app/
    dev-7375101_milicatest_1/exk19fnc2urbp2lBa5d7/sso/saml"
  ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
  AssertionConsumerServiceURL="https://localhost:44369/saml/login/receive">
      <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
          https://localhost:44369
      </saml:Issuer>
</samlp:AuthnRequest>
```

7.2 Пријем и валидација аутентификационог одговора

Креирање аутентификационог захтева дешава се у оквиру методе *HandleRequestAsync* која је део класе *SamlAuthenticationHandler*. Имплементација ове методе приказана је на слици 7.7. Ово је метода која пресреће све захтеве и

```
0 references | Milica Djuric, 36 days ago | 1 author, 1 change
public async Task<bool> HandleRequestAsync()
{
    string receivePath = this.spConfigurationProvider.GetAuthenticationResponseLocation();

    if (!receivePath.EndsWith(Request.Path.Value, StringComparison.OrdinalIgnoreCase))
    {
        return false;
    }

    if (Request.Method != HttpMethods.Post)
    {
        return false;
    }

    string redirectUrl = await this.authnResponseHandler.Handle();

    Context.Response.Redirect(redirectUrl);

    return true;
}
```

Слика 7.7: Имплементација методе *HandleRequestAsync* у оквиру библиотеке *Core.Saml2.Authentication*

проверава да ли неки од њих има путању која је дефинисана у оквиру параметара везаних за SAML пружаоца услуга и која говори где је потребно да стигне аутентификациони одговор. Уколико то јесте та путања, позива се метода *Handle* која припада класи *AuthnResponseHandler*. Имплементација ове методе приказана је на слици 7.8. Ова метода чита параметар *SAMLResponse* из тела захтева, врши *base64* декодирање и валидира његову исправност позивањем низа регистрованих валидатора. Након извршене валидације, креира се сесија за корисника чије су информације прочитане из аутентификационог одговора. Страница на коју је потребно вратити корисника чита се из објекта *RelayStateData* прослеђеног током слања аутентификационог захтева. Пре него што се корисник успешно преусмери на другу страницу, библиотека брише из своје меморије идентификатор аутентификационог захтева који одговара добијеном одговору. На овај начин се постиже јединствена обрада аутентификационог одговора и спречава се понављање већ искоришћеног тврђења.

С обзиром да је валидација аутентификационог одговора битна из сигурносних разлога, њој је посвећена нарочита пажња приликом имплементације библиотеке. Неки од најважнијих корака валидације које библиотека имплементира су:

1. Примљени аутентификациони одговор мора имати структуру дефинисану XML схемом протокола SAML која се може наћи на адреси `https://docs.oasis-open.org/security/saml/v2.0/saml-schema-protocol-2.0.xsd`. Уколико одговор не задовољава структуру ове схеме, корисник неће бити аутентификован.

2. Аутентификациони одговор мора садржати атрибут *Destination* који се мора поклапати са путањом која је дефинисана у оквиру параметара везаних

```
2 references | Milica Djuric, 22 days ago | 2 authors, 5 changes
public async Task<string> Handle()
{
    IFormCollection form = Request.Form;

    SamlValidationGuard.NotNull(
        form,
        SamlValidationErrors.AuthnResponseBodyShouldNotBeNull
    );

    string encodedResponse = form[SamlConstant.SamlResponse];
    string encodedRelayState = form[SamlConstant.RelayState];

    SamlValidationGuard.NotNull(
        encodedResponse,
        SamlValidationErrors.AuthnResponseShouldNotBeNull
    );

    byte[] decodedBytes = Convert.FromBase64String(encodedResponse);
    string decodedResponse = Encoding.UTF8.GetString(decodedBytes);
    AuthnResponse xmlResponseObject =
        this.serializeXmlService.Deserialize<AuthnResponse>(decodedResponse);

    SamlValidationGuard.NotNull(
        xmlResponseObject,
        "Received serialized SAML authn xml response should not ne null"
    );

    this.authnResponseContext.StringifiedResponse = decodedResponse;
    this.authnResponseContext.Response = xmlResponseObject;

    IList<ISamlAuthnResponseValidator> validators =
        this.authnResponseValidatorListProvider.Get();

    foreach(ISamlAuthnResponseValidator validator in validators)
    {
        await validator.Validate(xmlResponseObject);
    }

    string returnUrl = await this.SignIn(encodedRelayState);

    if (xmlResponseObject.InResponseTo.IsNotNullOrWhitspace())
    {
        await this.authnRequestStore.Remove(xmlResponseObject.InResponseTo);
    }

    return returnUrl;
}
```

Слика 7.8: Имплементација обраде аутентификационог одговора у оквиру библиотеке *Core.Saml2.Authentication*

за SAML пружаоца услуга и која говори где је потребно да стигне аутентификациони одговор. Уколико одговор не садржи овај атрибут или ако је његова вредност погрешна, корисник неће бити аутентификован.

3. Уколико одговор садржи атрибут *InResponseTo*, библиотека проверава да ли је аутентификациони захтев са идентификатором назначеним у овом атрибуту послат од стране апликације која користи библиотеку. Због ове провере, библиотека чува све идентификаторе послатих аутентификационих захтева у дистрибуираној меморији дефинисаној од стране апликације. Библиотека пријављује грешку и уколико добије два одговора са истим атрибутом *InResponseTo* тако што за једном обрађени аутентификациони одговор брише сачувани идентификатор одговарајућег аутентификационг

захтева из меморије.

4. Сваки временски атрибут у оквиру одговора пролази кроз валидацију. Атрибути *IssueInstant*, *AuthnInstant* и *NotBefore* не могу имати вредности које су у будућности. Атрибут *NotOnOrAfter* не сме имати вредност која је у прошлости. Приликом валидације временских атрибута, додаје се или одузима конфигурисан број милисекунди које се могу јавити због разлике у временима између послужиоца стране која шаље и послужиоца стране која прима.

5. Уколико одговор не садржи елемент *Status* или је вредност овог елемента „неуспешан", корисник неће бити аутентификован.

6. Аутентификациони одговор мора садржати елемент *Issuer* како би се знало који је аутентификациони добављач аутентификовао корисника и чији јавни кључ библиотека треба да користи за валидацију потписа примљеног одговора.

7. Уколико одговор није потписан, библиотека проверава да ли су тврђења у оквиру њега потписана. Уколико нису, библиотека не прихвата такав одговор и корисник неће бити аутентификован. Валидација потписа се врши помоћу јавног кључа који је назначен у подешавањима регистрованог аутентификационог добављача.

8. Уколико је тврђење, именски идентификатор или атрибут шифрован, библиотека ће прво одрадити дешифровање коришћењем приватног кључа који је дефинисан у оквиру параметара везаних за SAML пружаоца услуга. Уколико елемент није шифрован правилно и не може се извршити дешифровање, аутентификација корисника неће проћи успешно.

9. Аутентификациони одговор мора садржати барем једно тврђење са аутентификационим исказом како би библиотеци био познат идентификатор корисника који је извршио аутентификацију.

10. Библиотека проверава да ли је добијено тврђење намењено одговарајућем пружаоцу услуга тако што проверава вредност елемента *Audience* и атрибута *Recepient*. Вредност елемента *Audience* мора садржати јединствен идентификатор пружаоца услуга који библиотека добија из конфигурисаних параметара. Вредност атрибута *Recepient* мора бити једнака конфигурисаном параметру који говори где је потребно да стигне аутентификациони одговор.

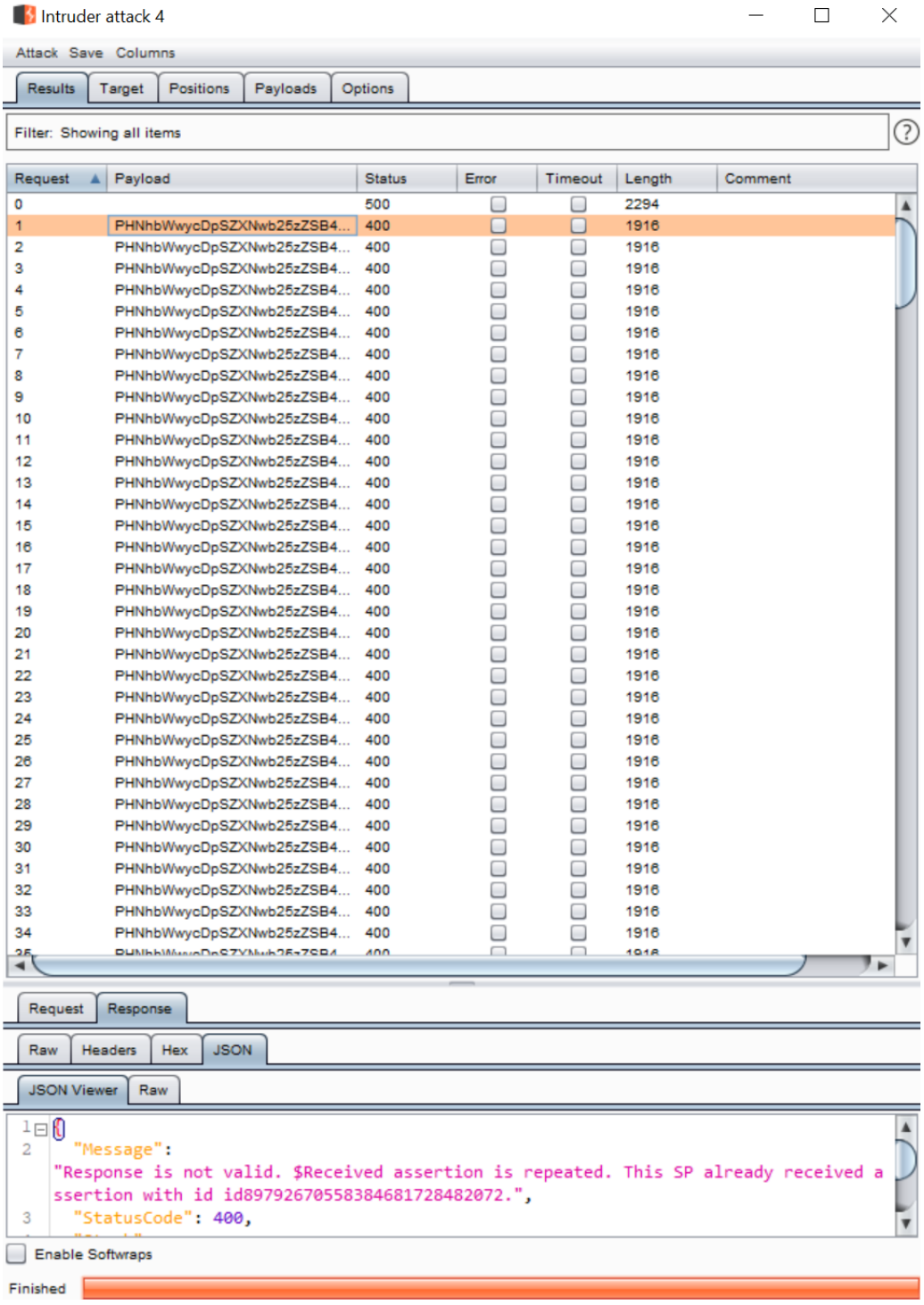
11. Библиотека у дистрибуираној меморији чува и све идентификаторе примљених тврђења. Уколико се тврђење са истим идентификатором понови, корисник неће бити аутентификован успешно.

Уколико би нека од ових валидација била прескочена, то би довело до озбиљних сигурносних пропуста. На пример, ако се прескочи први корак у коме се валидира XML схема, то може довести до многобројних напада познатих под називом *омотавање потписа* (енгл. *signature wrapping*) у којима злонамерни софтвер нпр. убацује непотписан аутентификациони одговор пре или после *Signature* елемента правог аутентификационог одговора. Уколико би се прескочила провера потписа, аутентификациони одговор би могао да пресретне неки злонамерни софтвер и да измени његов садржај пре слања жељеном пружаоцу услуга. Када добије одговор, пружалац услуга би веровао да је добијени садржај одговора креиран од стране аутентификационог добављача и креирао би сесију за потенцијално неаутентификованог корисника. Ако се прескочи провера поновљеног тврђења, злонамерни софтвер би могао да пресретне један аутентификациони одговор и да користи тврђење у оквиру њега како би се аутентификовао као корисник чијем налогу нема приступ. Провера временских атрибута сужава временски простор у коме злонамерни софтвер може на било који начин искористити аутентификациони одговор. Уколико се прескочи провера да ли је тврђење намењено одговарајућем пружаоцу услуга, злонамерни софтвер би могао да узме валидан аутентификациони одговор из неке друге апликације која користи истог аутентификационог добављача и креира сесију за корисника који припада другој апликацији.

Како би се проверили сигурносни аспекти креиране библиотеке, коришћен је алат *Burp Suite*[1] и његов додатак EsPReSSO[2]. Алат ради тако што пресретне један валидан аутентификациони одговор, затим прави измене над њим користећи своју базу најпознатијих SAML XML напада и шаље их пружаоцу услуга чији се сигурности аспекти тестирају. Алат је успео да пошаље 276 напада и ниједан напад није успешно прошао, тј. пружалац услуга ниједном није одговорио са статус кодом 200 што би значило да је сесија креирана. Пример једног од одговора на напад приказан је на слици 7.9.

[1]Више информација о алату Burp Suite може се наћи на адреси `https://portswigger.net/burp`.

[2]Више информација о додатку *EsPReSSO* може се наћи на адреси `https://portswigger.net/bappstore/e1d08d4ab1ea4c17be3431d7d2b20b30`.



Слика 7.9: Пример неуспешног SAML XML напада креираним од стране алата EsPReSSO Burp Suite

Глава 8

Закључак

Развој SSO протокола олакшао је приступ апликацијама и личним подацима у оквиру њих за многе кориснике. Највише користи добили су корисници апликација које се користе у пословне сврхе јер послодавци имају иницијативу да повежу апликације потребне за свакодневни рад како би уштедели време својим запосленима и како би смањили број проблема са лозинкама. У овом раду су описани неки од најзначајних SSO протокола као што су SAML, OAuth и OpenId Connect. Видели смо да је SAML комплексан протокол заснован на размени XML докумената, док су OAuth и OpenId Connect протоколи који су настали касније и чија је комплексност много мања од протокола SAML. Иако је протокол SAML комплексан и иако је давно настао, он се и даље користи, док OAuth и OpenId Connect имају широку примену у аутентификацији коришћењем друштвених мрежа. Овај рад анализирао је и подршку за поменуте протоколе у .NET Core-у, једном од најпознатијих радних оквира. Протоколи OAuth и OpenId Connect имају уграђену подршку, док SAML нема. Због тога је уз овај рад развијена и .NET Core библиотека која омогућава аутентификацију коришћењем протокола SAML. Библиотека је јавно доступна и спремна за коришћење. Тестирањем помоћу алата који мења аутентификациони одговор креирајући најпознатије сигурносне нападе је потврђен висок ниво сигурности библиотеке.

С обзиром на то да је сигурност корисника на вебу јако битна, даљи рад би укључивао приказ неких од најпознатијих врста напада на описане SSO протоколе, као и предлог метода за заштиту од ових напада. Такође, један од праваца даљег рада би био и приказ метода за обједињену одјаву са свих апликација повезаних једним централним аутентификационим добављачем.

Библиографија

[1] Scott Cantor и др. „Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite". у: *OASIS Open 2015* (2015.), стр. 1–93. URL: https://www.oasis-open.org/committees/download.php/56776/sstc-saml-core-errata-2.0-wd-07.pdf.

[2] Dipankar Dasgupta, Arunava Roy и Abhijit Nag. *Advances in User Authentication*. Springer, 2017.

[3] CTO at CloudSponge Graeme Rouse. *What is OAuth and Why do Companies Use It?* приступљено 14. децембра 2020. URL: https://www.cloudsponge.com/blog/what-is-oauth-and-why-do-companies-use-it/.

[4] Roger A. Grimes и Josh Fruhlinger. *What is OAuth? How the open authorization framework works*. 2019 (приступљено 14. децембра 2020.) URL: https://www.csoonline.com/article/3216404/what-is-oauth-how-the-open-authorization-framework-works.html.

[5] Frederick Hirsch, Rob Philpott и Eve Maler. „Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0". у: *OASIS Open 2015* (2005.), стр. 1–33. URL: http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf.

[6] John Hughes и др. „Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite". у: *OASIS Open 2015* (2015.), стр. 1–68. URL: https://www.oasis-open.org/committees/download.php/56782/sstc-saml-profiles-errata-2.0-wd-07.pdf.

[7] Internet Engineering Task Force IETF. *JSON Web Token (JWT)*. 2015 (приступљено 13. јуна 2021.) URL: https://datatracker.ietf.org/doc/html/rfc7519.

[8] D. Hardt Internet Engineering Task Force (IETF). *The OAuth 2.0 Authorization Framework*. 2012 (приступљено 22. marta 2021.) URL: https://tools.ietf.org/html/rfc6749.

[9] Kelly D. Lewis и James E. Lewis. „Web Single Sign-On Authentication using SAML". у: *IJCSI International Journal of Computer Science Issues* 2 (2009.), стр. 41–48. URL: https://arxiv.org/ftp/arxiv/papers/0909/0909.2368.pdf.

[10] Nate Lord. *Uncovering Password Habits: Are Users' Password Security Habits Improving?* 2020 (приступљено 3. октобра 2020.) URL: https://digitalguardian.com/blog/uncovering-password-habits-are-users-password-security-habits-improving-infographic.

[11] Christian Mainka и др. „SoK: Single Sign-On Security — An Evaluation of OpenID Connect". у: *2017 IEEE European Symposium on Security and Privacy (EuroS P)* (2017.), стр. 251–266. DOI: 10.1109/EuroSP.2017.32.

[12] Google / Harris Poll. *Online Security Survey Google / Harris Poll*. 2019 (приступљено 29. септембра 2020.) URL: https://services.google.com/fh/files/blogs/google_security_infographic.pdf.

[13] Nick Ragouzis и др. „Security Assertion Markup Language (SAML) V2.0 Technical Overview". у: *OASIS Security Services TC* (2008.), стр. 1–51. URL: https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf.

[14] N. Sakimura и др. *OpenID Connect Core 1.0*. 2014 (приступљено 6. јуна 2021.) URL: https://openid.net/specs/openid-connect-core-1_0-final.html.

[15] SplashData и TeamsID. *Top 50 Worst Passwords of 2018*. 2018 (приступљено 3. октобра 2020.) URL: https://www.teampassword.com/blog/worst-passwords-of-2018.

[16] SplashData и TeamsID. *Top 50 Worst Passwords of 2019*. 2019 (приступљено 3. октобра 2020.) URL: https://www.teampassword.com/blog/top-50-worst-passwords-of-2019.

[17] Cesar de la Torre, Bill Wagner и Mike Rousos. *.NET Microservices: Architecture for Containerized .NET Applications*. Microsoft Corporation, 2020. URL: https://aka.ms/microservicesebook.