Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

# Shibboleth autentizace v Javě

***Kamil Maleček***

Vedoucí práce: Ing. Ondřej Guth, Ph.D.

1. května 2016

# Poděkování

Děkuji vedoucímu bakalářské práce Ing. Ondřeji Guthovi, Ph.D. za zajímavé téma, ochotu a cenné rady.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 1. května 2016 .....

### Odkaz na tuto práci

Maleček, Kamil. *Shibboleth autentizace v Javě*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

# Abstrakt

Bakalářská práce se zabývá návrhem a implementací autentizačního řešení pro Java Enterprise Edition (Java EE) aplikace se systémem jednotného přihlášení Shibboleth, s využitím standardních nástrojů Java EE a aplikačního serveru GlassFish. V závěru je popsáno ověření funkčnosti a zhodnocení implementovaného řešení.

**Klíčová slova** autorizace, autentizace, Java Enterprise Edition, jednotné přihlášení, Shibboleth, aplikační server, GlassFish, SAM, SAML

# Abstract

Bachelor thesis deals with the design and implementation of an authentication solution for Java Enterprise Edition (Java EE) applications using single sign-on system Shibboleth, by means of standard tools of Java EE and application server GlassFish. In the conclusion, there is a description of the test scenario including pros and cons of the solution.

**Keywords** authorization, authentication, Java Enterprise Edition, single sign-on, Shibboleth, application server, GlassFish, SAM, SAML

# Obsah

# Seznam obrázků

# Úvod

Informační systémy jsou nedílnou součástí každodenního života nás všech a zabezpečení přístupu k citlivým datům, v nich uložených, je vždy důležitým požadavkem. Rozhodování o přístupu ke zdrojům informací se nazývá autorizace a pro chráněné zdroje ji často předchází autentizace, respektive ověření proklamované identity subjektu žádajícího o přístup k chráněnému zdroji. Ruku v ruce s narůstajícím počtem různých webových služeb, které denně používáme, se zvyšuje počet přihlašovacích údajů, jež si musíme pamatovat a opakovaně zadávat. Tento problém lze řešit použitím služby jednotného přihlášení čili single sign-on (SSO). Jedním ze zástupců je projekt Shibboleth [1], který je mimo jiné používán na Českém vysokém učení technickém v Praze (ČVUT), na Fakultě informačních technologií (FIT) [2].

Technologii Shibboleth využívá mnoho aplikací postavených na platformě Java Enterprise Edition (Java EE) [3]. Zatím však neexistuje řešení autentizace těchto aplikací pouze pomocí standardních nástrojů Java EE a možností dostupných v rámci aplikačních serverů. Práce se zabývá analýzou, návrhem a implementací řešení tohoto problému.

## Cíle práce

Cílem rešeršní části bakalářské práce je detailní seznámení se s principy projektu Shibboleth, dále prostudování všech dostupných způsobů autentizace, které umožňuje aplikační server GlassFish Full Platform (dále jen GlassFish, vysvětlení v kapitole 1.2.1.3) verze 4 [4]. Server byl zvolen, protože jde o referenční implementaci Java EE verze 7.

Cílem praktické části je s využitím poznatků z rešeršní části navrhnout a implementovat způsob autentizace, který by využíval k získání a ověření identity komponentu Shibboleth Identity Provider (IdP). Řešení musí být napsané s využitím standardních technologií jazyka Java EE ve verzi 7 a použitelné pro již zmíněný aplikační server GlassFish verze 4. Implementované

řešení je potřeba ověřit pomocí testovacího scénáře simulujícího reálné použití technologie Shibboleth.

## Struktura práce

První kapitola obsahuje jak stručný popis samotné technologie Shibboleth, tak průzkum současných řešení autentizace Java EE aplikací využívajících právě tuto technologii. V její druhé podkapitole je proveden rozbor autentizačních možností aplikačního serveru GlassFish verze 4.

Na základě získaných poznatků je v další části bakalářské práce navržen autentizační způsob, který by mohl nahradit současná řešení a přitom by využíval pouze standardní prostředky Java EE 7 a možnosti aplikačního serveru GlassFish 4. Kapitola obsahuje také výběr vhodných knihoven pro implementaci.

Výstupem práce je funkční řešení, jehož implementaci dle návrhu se věnuje třetí kapitola.

V poslední kapitole je popsán testovací scénář simulující reálné prostředí. Závěrečná kapitola obsahuje navíc zhodnocení implementovaného řešení, popis jeho výhod, nevýhod a srovnání s aktuálně používaným řešením.

Kapitola 1

# Technologie

## 1.1 Shibboleth

Služba Shibboleth je jedním z nejpoužívanějších zástupců systémů jednotného přihlášení. Tyto systémy umožňují přístup k několika chráněným zdrojům na webu či různým webovým aplikacím pomocí jednoho přihlášení, jedné autentizace. Odpadá tedy potřeba přihlašování se pro každou aplikaci zvlášť a pamatování si několika přihlašovacích údajů. Typické použití lze najít například v akademickém prostředí, kde se jedním přihlášením získá přístup k emailové schránce, portálu na podporu výuky, elektronickému indexu a dalším službám. Celý projekt Shibboleth je zdarma, open source a je dostupný pod licencí Apache Software License. Vývoj projektu začal v roce 2000 jako aktivita americké asociace Internet2 [5]. Dnes vývoj zastřešuje Shibboleth Consortium, které má mnoho členů, mimo jiné původní Internet2 či českou instituci CESNET [6].

### 1.1.1 Komponenty

Vedle samotného uživatele přistupujícího pomocí svého webového prohlížeče k chráněným webovým zdrojům se v rámci technologie Shibboleth rozlišují dvě komponenty:


Obrázek 1.1: Shibboleth logo [5]

- Poskytovatel identity (Identity Provider): Komponenta zodpovědná za autentizaci uživatele. Mezi jeho činnosti patří přijímání autentizačních žádostí od poskytovatelů služeb, jež chtějí získat přístup, dále samotná autentizace uživatelů typicky proti Lightweight Directory Access Protocol (LDAP) či Kerberos, a zisk uživatelských atributů jako uživatelského jména a skupin nově přihlášeného uživatele, případně dalších dat. Získaná data zpracovává a zabezpečeně odesílá spolu s rozhodnutím o úspěchu autentizace zpátky poskytovatelům služeb. Důležitou vlastností je, že přihlašovací údaje se zadávají přímo na IdP, pracuje se s nimi pouze v rámci poskytovatele identity a dále se nikam neodesílají. Často je označován jako Shibboleth IdP [7].

- Poskytovatel služeb (Service Provider): Komponenta sloužící k ochraně zdrojů, výběru poskytovatele identity podle konkrétní aplikace, vytvoření a odeslání autentizační žádosti na zvoleného IdP. Od něj také přijímá odpovědi, které zpracovává a získané uživatelské informace poskytuje žádajícím aplikacím. Shibboleth Service Provider (SP) může být jak součástí aplikace, tak samostatná aplikace běžící na jiném serveru. Často je označován jako Shibboleth SP [8].

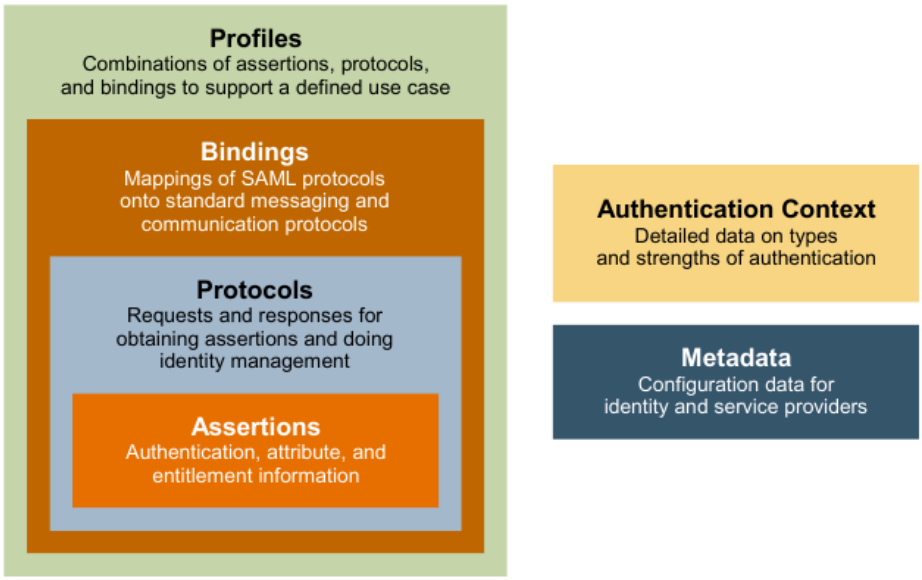
### 1.1.2 Vzájemná interakce

SSO systémy se dříve spoléhaly na cookies v prohlížeči uživatele pro udržení stavu přihlášení, zmizela tedy potřeba opakování autentizace při dalším přístupu k aplikaci. Nicméně cookies nejsou nikdy sdílené mezi více doménami. Starší SSO systémy podporovaly multi-domain SSO (MDSSO), tedy předávání stavu o autentizaci mezi jednotlivými doménami. V heterogenním prostředí se při používáních vlastních protokolů ukázalo toto řešení jako nepraktické. Řešení přinesl Security Assertion Markup Language (SAML) [9].

#### 1.1.2.1 SAML

SAML standard vznikl pod záštitou neziskového konzorcia OASIS a dodnes je vyvíjen Security Services Technical Commitee of OASIS (SSTC) [10]. SAML framework je založený na Extensible Markup Language (XML) a slouží ke komunikaci mezi různými objekty [11], lze použít například pro Google Apps [12]. SAML se skládá z několika komponent, které umožňují vytvořit mnoho případů užití. Jednotlivé komponenty znázorňuje obrázek 1.2. Primárním účelem však zůstává přenos identity, atributů a autentizačních zpráv. SAML 2.0 definuje prohlášení, protokoly, bindings a profily.

SAML umožňuje vytvořit prohlášení (assertion) skládající se z tvrzení (statements). Strukturu naznačuje kód 1.1. Tvrzení se týkají předmětu prohlášení. Předmětem může být například jméno uživatele a v dalších datech mohou být například skupiny, do nichž patří. Tvrzení se dále dělí na tři kategorie tvrzení:




Obrázek 1.2: SAML 2.0 komponenty [13]

Kód 1.1: Ukázka struktury SAML 2.0 prohlášení

```
<saml2:Assertion ...>
  ...
  <saml2:AuthnStatement ...>
    ...
  </saml2:AuthnStatement>
  <saml2:AttributeStatement ...>
    ...
  </saml2:AttributeStatement>
</saml2:Assertion>
```

- Authentication statements: Vytváří jej strana, která úspěšně autentizovala uživatele. Obsahuje informaci o způsobu provedené autentizace, čas a volitelně další položky.
- Attribute statements: Obsahuje specifické atributy spojené s předmětem prohlášení. Typicky se jedná o uživatelské jméno, skupiny, kontakty.
- Authorization decision statements: Obsahují povolení či odmítnutí žádosti o provedení akce předmětem prohlášení.

Protokoly slouží k definování žádostí a vracení odpovídajících odpovědí. Důležitými protokoly jsou:

- Authentication Request Protocol: Využívá se v souvislosti s autentizačním prohlášením, definuje odeslání takového požadavku a vyžadování

odpovědi obsahující prohlášení. Je využíván při přesměrování z SP na IdP v rámci Web Browser SSO Profile.

- Single Logout Protocol: Definuje přenos zpráv, jenž ukončí všechny aktivní session spojené s přihlášeným uživatelem, slouží tedy k odhlášení na úrovni IdP. Odhlášení může iniciovat samotný uživatel nebo SP či IdP kvůli nastavené době vypršení session.

- Assertion Query and Request Protocol: Určuje dotazy, pomocí kterých se může entita dotázat na existující nebo nové prohlášení použitím jeho ID, respektive na základě předmětu prohlášení.

Dalšími protokoly jsou Artifact Resolution Protocol, Name Identifier Management Protocol a Name Identifier Mapping Protocol. Práce se jim blíže nevěnuje.

SAML bindings říkají, jaký prostředek bude použit pro přenesení zpráv SAML protokolu. Mezi nejpoužívanější patří:

- HTTP Redirect Binding: Vytváří HTTP GET požadavek obsahující SAML zprávu jako URL parametr.

- HTTP POST Binding: Zpráva SAML protokolu je zakódována pomocí base64 a odeslána metodou HTTP POST.

- SAML URI Binding: Definuje prostředky k získání existujícího SAML prohlášení podle URI.

Neuvedené bindings jsou HTTP Artifact Binding, SAML SOAP Binding, Reverse SOAP (PAOS) Binding. Práce se jim dále nevěnuje, především kvůli nevyužití SOAP protokolu.

Profily slouží k definici byznysových případů užití. Říkají, jakým způsobem lze kombinovat prohlášení, protokoly a bindings. V podstatě vymezují omezení pro jednotlivé komponenty. Nejdůležitějším profilem je Web Browser SSO Profile, jemuž se podrobně věnuje následující kapitola. Dalším zajímavým profilem je Single Logout Profile, který definuje, jak může být použit Single Logout Protocol pomocí SOAP, HTTP Redirect, POST či HTTP Artifact Binding. Zbylé profily nejsou pro práci podstatné [14].

Pro používání SAML jsou důležité další dvě komponenty:

- Metadata: Definují konfigurace jednotlivých zúčastněných entit. Obsahují mimo jiné entitou podporované SAML bindings, informace sloužící pro identifikaci, podporované atributy, případně mohou obsahovat údaje potřebné k šifrování a podepisování zpráv. Tyto konfigurace si entity navzájem sdílí, aby správně komunikovaly s využitím podporovaných prostředků.

- Authentication context: SP jej využívá pro držení detailních informací o autentizaci, díky nim může například požádat IdP o speciální autentizaci, jakou může být vícefázové ověření identity.

Neméně důležité jsou také nástroje pro zabezpečení přenosu SAML zpráv. Kdyby komunikace přes protokol HTTP nebyla vůbec zabezpečená, nic by nebránilo útoku *man-in-the-middle*. Útočník by mohl zachytit zprávy posílané mezi entitami, přečíst obsah, případně upravit, a poslat dále původnímu příjemci. Obě entity by si však stále myslely, že komunikují přímo se svým protějškem. Jako prevenci nejenom proti tomuto útoku SAML nabízí několik bezpečnostních opatření. Podrobněji se jim práce věnuje v dalších kapitolách, pro úplnost mezi ně patří šifrované HTTP spojení pomocí SSL 3.0 nebo TLS, případně použití digitálně podepsaných zpráv [13].

### 1.1.3 Web Browser SSO Profile

Profil definuje, jakým způsobem lze používat SAML zprávy pro nejdůležitější případ užití, kterým je bezpochyby SSO na webu. Use case začíná ve stavu, kdy je uživatel přihlášen na webové adrese *test1.example.com* a při přesměrování v rámci předem vytvořené federace na adresu *test2.example.com* nemusí opakovat proces autentizace.

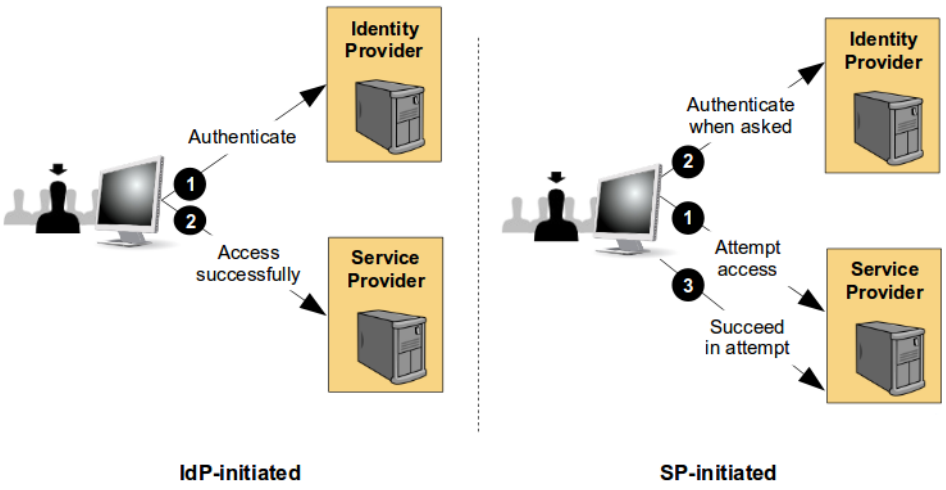
Realizace se primárně dělí podle toho, zda proces výměny zpráv iniciuje IdP, nebo SP. Primární dělení znázorňuje obrázek 1.3. Sekundární dělení je potom dle použitých bindings při komunikaci. Profil dále definuje určitá omezení pro komunikaci, zejména říká, že SAML Authentication Request je odesílána z SP na IdP pomocí HTTP Redirect Binding, HTTP POST Binding či HTTP Artiface Binding. Pro SAML Response putující v opačném směru však nemůže být použit HTTP Redirect Binding. Jednotlivé bindings této komunikace mohou být tedy kombinovány více způsoby:

1. IdP zahajuje komunikaci:

   - POST Binding je použité pro přenos SAML Response zprávy na SP, naopak od SP není žádná SAML Authentication Request posílána. Uživatel nepřistupuje přímo na zdroj umístěný na SP, ale přistupuje nejdříve na IdP, které má pro každý SP zvláštní konfiguraci. Na IdP se uživatel přihlásí a vybere zdroj, na nějž chce přistoupit. Tím je spuštěno odeslání SAML Response zprávy ke zvolenému SP a uživatel získává přístup ke zdrojům na vybraném SP.

2. SP zahajuje komunikaci:

   - Použití HTTP Redirect Binding pro odeslání SAML AuthnRequest na IdP a HTTP POST Binding pro doručení SAML Response na SP. Proces začíná pokusem uživatele přistoupit ke zdroji na SP,




Obrázek 1.3: Průběh komunikace dle iniciátora [15]

uživatel zde však není přihlášen a je přesměrován na IdP, odkud je po přihlášení vrácen zpět na SP.

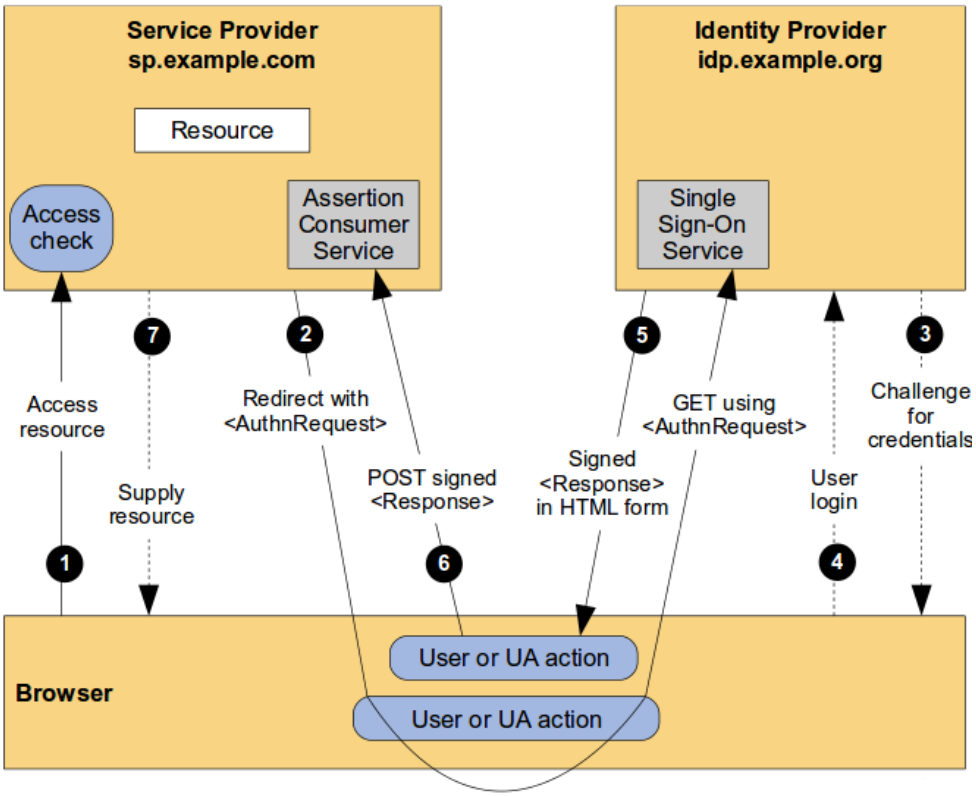
- Využití HTTP POST Binding pro přenos SAML AuthnRequest zprávy na IdP. HTTP Artifact Binding potom slouží k odeslání SAML Response zprávy od IdP. Po doručení artefaktu na SP je pro získání odpovídající SAML zprávy použit SOAP binding. Průběh procesu je jinak stejný jako v předchozím případě. Tento způsob komunikace, přesněji použití HTTP POST Binding pro doručení SAML AuthnRequest zprávy na IdP je nutné i v případě, kdy je zpráva moc dlouhá pro použití HTTP Redirect Binding [15].

#### 1.1.3.1 Komunikace zahájená SP: Redirect/POST Bindings

Pro další kapitoly práce je nezbytné bližší seznámení se s procesem SSO inicializovaného SP a využívajícího HTTP Redirect Binding a HTTP POST Binding. Průběh procesu je zobrazen také na obrázku 1.4.

1. Uživatel se pokusí o přístup ke zdroji umístěnému na určitém SP. Uživatel ale nemá autentizační kontext pro tuto stránku. SP si zapamatuje URL požadovaného zdroje.

2. Poskytovatel služeb vytvoří SAML AuthnRequest zprávu a odpoví prohlížeči HTTP statusem 302, nebo 303, jenž znamená redirect. *Location* hlavička odpovědi obsahuje URI prokytovatele identity. Na tuto adresu je poté poslán HTTP GET požadavek obsahující deflate kódováním zakódovanou SAML AuthnRequest zprávu jako URL parametr s názvem *SAMLRequest*. Jako hodnota dalšího parametru *RelayState* je odeslána uložená URL adresa uživatelem požadovaného zdroje [16].



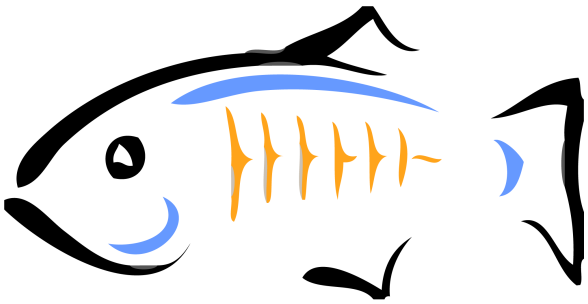

Obrázek 1.4: SP iniciátorem: HTTP Redirect/POST schéma [15]

3. Pokud na IdP neexistuje autentizační kontext, jenž by splňoval požadavky dané v přijaté SAML AuthnRequest zprávě, je uživatel vyzván k přihlášení.

4. Po ověření údajů se pro uživatele vytvoří nový autentizační kontext na úrovni IdP.

5. Podle tohoto kontextu IdP vytvoří SAML Assertion. Vytvořené tvrzení digitálně podepíše a vloží do SAML Response zprávy. Zpráva je zakódována pomocí base64 a je vložena do těla HTTP POST požadavku pod názvem *SAMLResponse*. V případě, že IdP obdržel *RelayState* parametr od SP, musí vrátit jeho nezměněnou hodnotu opět pod stejným názvem v těle požadavku. Vytvořený požadavek je odeslán na adresu tzv. Assertion Consumer Service (ACS), jež zpracovává profily Authentication Request protokolu, v tomto případě pro HTTP POST Binding. Definici ACS obsahují metadata daného SP.

6. ACS provede ověření digitálního podpisu a zpracuje SAML Response zprávu. Na základě jejího obsahu je vytvořen lokální autentizační kontext



Obrázek 1.5: GlassFish logo [18]

společně s principle (přihlášená aplikační identita). Služba díky *RelayState* parametru získává URL adresu uživatelem požadovaného zdroje a pošle prohlížeči HTTP redirect s touto adresou.

7. Před tím, než prohlížeč na URL přistoupí, proběhne kontrola, zda přihlášený principle má oprávnění na zdroj přistoupit (autorizace) [15].

## 1.2 Aplikační server GlassFish

Projekt GlassFish byl spuštěn v roce 2005 společností Sun Microsystems. Po akvizici společnosti firmou Oracle vznikly dvě edice. Jednak komerční verze Oracle GlassFish Server, která byla nabízena jako levnější alternativa k Oracle WebLogic serveru, a jednak vznikl GlassFish Server Open Source Edition, který byl firmou Oracle sponzorován. V roce 2013 Oracle oznámil, že ukončí vývoj komerční verze a bude poskytovat pouze podporu pro jeho tehdy aktuální verzi 3. Důvodem mohlo být minimální zastoupení tohoto serveru v produkci.

Oracle dnes prezentuje GlassFish jako referenční implementaci poslední verze Java EE platformy, avšak neposkytuje k němu žádnou podporu. Aktuální verze GlassFish Server 4 je tedy open source projekt sponzorovaný firmou Oracle. GlassFish je rychlý a snadno použitelný aplikační server s plnou podporou platformy Java EE verze 7 [17].

### 1.2.1 Autentizační možnosti

Pro další kapitoly práce jsou důležité především autentizační možnosti zvoleného aplikačního serveru GlassFish Full Platform verze 4, jelikož cílem práce je pro autentizaci využít systém Shibboleth. Autentizace je ověření, že entita je tím, kým tvrdí že je, nebo-li ověření autentizačních údajů dané entity. Autentizačním údajem mohou být jak přihlašovací údaje (přihlašovací jméno a heslo), tak digitální certifikát, případně jiné údaje. Server spouští vybraný autentizační mechanismus v momentě, kdy se entita snaží přistoupit k chráněnému zdroji na serveru. V případě úspěšné autentizace je entita

přihlášena po zbytek trvání session. Autentizace však nerozhoduje o přístupu k požadovanému zdroji. Rozhodnutí o přístupu se nazývá autorizace [19].

#### 1.2.1.1 Single Sign-on

GlassFish má vestavěnou podporu pro SSO, to znamená, že uživatel přihlášený v jedné aplikaci má přístup i k dalším aplikacím běžícím na stejném (virtuálním) serveru. SSO je založené na rolích definovaných na serveru. Dalším omezením je nutnost použití stejného realm [19].

Realm je prostředek, pomocí kterého GlassFish definuje a aplikuje bezpečnostní politiku. Realm je v podstatě Java Authentication and Authorization Service (JAAS) security context, ve kterém aplikace běží. Existují předkonfigurované a zároveň lze vytvářet vlastní. Mezi výchozí patří:

- File realm: Defaultní realm, který ukládá přihlašovací údaje uživatelů lokálně do souboru *keyfile*.
- Administration realm: File realm pro přihlašovací údaje administrátorů, uloženy v souboru *admin-keyfile*.
- Certificate realm: Server získává údaje z klientských certifikátů.
- LDAP realm: Využití LDAP pro získání přihlašovacích údajů.
- JDBC realm: Získání údajů z databáze.
- Digest realm: Použití Digest autentizační metody, přenáší zašifrované heslo.
- Oracle Solaris realm: Podporováno pouze na Oracle Solaris 9/10, získání údajů přímo ze systému.
- PAM realm: Aplikace na GlassFish se mohou autentizovat s využitím Pluggable Authentication Module (PAM) přímo v Unix systému.

V případě jiných autentizačních požadavků přichází na řadu vlastní realm. Výstupem samotné realm třídy je informace o uživateli a jeho skupinách (principle). K realmu patří JAAS přihlašovací modul, který provádí vlastní autentizaci. Samotná třída přihlašovacího modulu má přístup ke zděděným proměným *_username* a *_password*. S využitím těchto údajů získaných od uživatele pomocí určitého *CallbackHandler* provede potřebnou autentizaci [20]. Správně umístěný a zaregistrovaný realm v GlassFish je poté k dispozici aplikacím. Volba realm se provádí přímo v aplikaci prostřednictvím souboru *web.xml* 1.2.

Vzhledem k tomu, že samotný princip realm je ověření údajů získaných od uživatele v rámci JAAS context proti určitému zdroji, nelze tento prostředek použít pro SSO Shibboleth. Přihlašovací údaje uživatel zadává na Shibboleth IdP, mimo tuto komponentu nejsou k dispozici.

Kód 1.2: Ukázka konfigurace autentizace Java EE aplikace (*web.xml*)

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>MyRealm</realm-name>
  <form-login-config>
    <form-login-page>login.jsp</form-login-page>
    <form-error-page>error.jsp</form-error-page>
  </form-login-config>
</login-config>
```

#### 1.2.1.2 Standardní typy autentizace

GlassFish podporuje více typů běžné HTTP autentizace:

- BASIC: Autentizace založená na přihlašovacím jménu a heslu. Internetové prohlížeče k získání údajů typicky používají vestavěný dialog. Údaje jsou poté v pevně daném tvaru zakódovány pomocí base64 a odeslány v hlavičce odpovědi na server. Z hlediska bezpečnosti je metoda nevyhovující, je vhodné použití HTTPS pro zabezpečení přenosu.

- FORM: Pro autentizaci lze použít vlastní přihlašovací stránku, na které musí být formulář s políčky pojmenovanými *j_username* a *j_password*. Údaje jsou na server přenášeny jako obyčejný text a proto je vhodné použití HTTPS. Vzorovou konfiguraci Java EE aplikace používající FORM metodu zobrazuje kód 1.2.

- CLIENT-CERT: Autentizace prováděná na základě certifikátů ve formátu X.509. Využívá HTTPS.

- DIGEST: Metoda podobná BASIC autentizaci, avšak heslo se od klienta přenáší zašifrované.

Ani jeden z těchto typů není pro bakalářskou práci vhodný. U všech metod se přihlašovací údaje zadávají na klientovi a poté odesílají na server, kde se provede nejprve autentizace pomocí zvoleného realm (viz výše), a následně autorizace na bázi skupin definovaných na serveru [19].

#### 1.2.1.3 Server Authentication Module

Společným problémem dříve zmíněných autentizačních řešení je, že neposkytují způsob, jak přenést autentizaci mimo kontejner aplikačního serveru, získat poté autentizovaného uživatele a předat ho kontejneru pro aplikování bezpečnostních politik aplikace.

Již v roce 2002 vznikl požadavek Java Specification Request (JSR) na definici standardu Service Provider Interface (SPI), jenž by sloužil k integraci

Kód 1.3: Soubor *glassfish-web.xml*: konfigurace SAM

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish
    Application Server 3.1 Servlet 3.0//EN" "http://glassfish.org/dtds
    /glassfish-web-app_3_0-1.dtd">

<glassfish-web-app
  httpservlet-security-provider="SAM-id-from-domain.xml-file">
  ...
</glassfish-web-app>
```

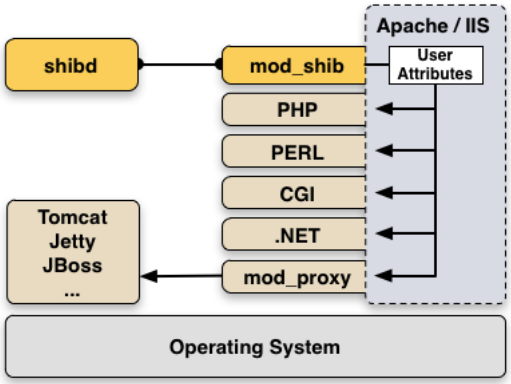
autentizačních modulů s kontejnery aplikačních serverů, které rozhodují o přístupu ke zdrojům (autorizace). Požadavek byl pojmenován Java Authentication Service Provider Interface for Containers (JASPIC), dostal označení JSR 196 a cílovou platformou měla být tehdy ještě Java 2 Enterprise Edition (J2EE) verze 1.4 [21]. Vývoj se však protáhl a standard byl představen až s příchodem Java EE 6 v roce 2009. JASPIC je též někdy označován jako JASPI (JBoss, IBM) a není součástí takzvané Web Profile podmnožiny Java EE technologií. Web Profile implementace jako například server TomEE či GlassFish Web Profile proto JASPIC vůbec neobsahují [22].

GlassFish Full Platform verze 4, jakožto referenční implementace Java EE 7 Full Platform obsahuje implementaci JASPIC verze 1.1 [19]. SPI slouží k delegování transportního mechanizmu v rámci kontejneru na Server Authentication Module (SAM), který v podstatě pracuje s *HttpServletRequest* a *HttpServletResponse*. SAM je volán pro každý příchozí požadavek (request) o přístup k jakémukoliv zdroji [23] a může být používán více aplikacemi zároveň, je tedy zcela bezestavový. Moduly je potřeba stejně jako realm ukládat do k tomu určeného adresáře. Následné zaregistrování modulu se projeví v hlavním konfiguračním souboru domény *domain.xml*. Jednotlivé aplikace si volí konkrétní SAM prostřednictvím souboru *glassfish-web.xml*, případně souboru s původním pojmenováním *sun-web.xml*, které slouží i k další konfiguraci webových aplikací v GlassFish. Konfigurační soubor má hierarchickou strukturu [24], část souboru s volbou SAM zobrazuje kód 1.3.

## 1.3 Současná řešení

V současné době není k dispozici referenční implementace Shibboleth 2 SP pro platformu Java. Neexistuje ani podpora SSO Shibboleth ze strany GlassFish či jiných aplikačních serverů pro platformu Java EE [25].

Java EE aplikace, které chtějí použít SSO Shibboleth mají dnes dvě možnoti pro vytvoření SP. Jednou je referenční implementace poskytovatele služeb napsaná v C++ spolu s webovým serverem a druhou je Spring Security SAML [26]. Obě řešení přibližují následující řádky.




Obrázek 1.6: Apache HTTP schéma [28]

### 1.3.1 Nativní řešení

Nativní Shibboleth SP je momentálně implementován jako C++ modul pro webové servery Apache HTTP, Internet Information Server (IIS) a Netscape Server Application Programming Interface (NSAPI) [25]. Webový server vystupuje jako reverzní proxy server. Jeho klienty mohou být právě aplikační servery s běžícími aplikacemi. Tyto servery se udržují ve spojení s webovým serverem pomocí Apache JServ Protocol version 1.3 (APJ13) protokolu, jenž slouží k udržení TCP spojení pro jeho opakované použití. Tím se eliminuje časová náročnost vytváření jednotlivých spojení. Protokol slouží pro předávání autentizačních informací od SP aplikacím, respektive kontejnerům aplikačních serverů, kde jsou dále zpracovávány pomocí JAAS login modulů [27].

Schéma na obrázku 1.6 ukazuje potřebné moduly pro Apache HTTP/IIS. Ke komunikaci s aplikačními servery slouží modul *mod_proxy*, zatímco modul *mod_shib* tvoří spolu s démonem *shibd* nativní implementaci SP [29].

### 1.3.2 Spring Security SAML

Spring Security SAML rozšíření umožňuje aplikacím vystupovat jako poskytovatel služeb v SSO systémech používajích protokol SAML 2.0. Rozšíření je možné použít i v aplikacích nevyužívajících Spring Security framework, případně aplikacích používajících i další formy autentizace. Spring Security SAML poskytuje mnoho možností a jde pravděpodobně o nejkomplexnější open source SAML 2.0 SP implementaci. Rozšíření umožňuje například generování SP metadat (viz ukázka nastavení 1.4), zpracování SAML zpráv (využívá knihovnu OpenSAML popsanou v další kapitole) či komunikaci zahájenou jak SP, tak IdP. Mezi podporované poskytovatele identit patří Shibboleth, OpenAM, ADFS, Okta a další [30].

Ač je Spring framework dnes velmi úspěšné komplexní řešení pro webové Java aplikace, nelze její komponentu považovat za řešení zadání této práce,

Kód 1.4: Vzorové nastavení Spring generátoru SP metadat [31]

```
<bean id="metadataGeneratorFilter" class="org.springframework.
  security.saml.metadata.MetadataGeneratorFilter">
  <constructor-arg>
    <bean class="org.springframework.security.saml.metadata.
        MetadataGenerator">
      <property name="entityId" value="replaceWithUniqueIdentifier"/>
      <property name="extendedMetadata">
        <bean class="org.springframework.security.saml.metadata.
            ExtendedMetadata">
          <property name="signMetadata" value="false"/>
          <property name="idpDiscoveryEnabled" value="true"/>
        </bean>
      </property>
    </bean>
  </constructor-arg>
</bean>
```

jelikož Spring framework není součástí Java EE standardu [32].

Kapitola 2

# Návrh řešení

V předchozí kapitole nebylo nalezeno existující řešení, které by splňovalo zadání práce, je tedy potřeba navrhnout řešení nové a vybrat prostředky k jeho realizaci.

## 2.1 Poskytovatel služeb

Nativní implementace SP využívající webový server s modulem napsaným v C++ jako reverzní proxy je zcela nevyhovující a právě tuto komponentu je potřeba nahradit. Určitou inspiraci poskytuje Spring Security SAML, kde poskytovatel služeb nevystupuje jako samostatná komponenta, nýbrž je součástí samotné aplikace. Na funkčnost toto řešení nemá žádný vliv, z pohledu poskytovatele identit se také nic nemění.

### 2.1.1 Servlet Filters

Pro zajištění primární funkcionality SP, tedy odesílání, přijímání HTTP požadavků a jejich dodatečné úpravě slouží v jazyku Java takzvané Servlet Filters.

Servlet je v podstatě Java třída dědící od *javax.servlet.http.HttpServlet*, která představuje vrstvu mezi HTTP klientem a aplikací na serveru. Klientem je typicky prohlížeč sloužící odpovídající servlet třídě ke sběru dat a jejich prezentaci, respektive k tvorbě dynamického obsahu webové stránky.

Servlet filtr je třída implementující rozhraní *javax.servlet.Filter*. Deklarace filtru (viz kód 2.1) se nachází v souboru *web.xml*, stejně jako jeho mapování na vybrané servlet třídy podle URL vzoru nebo vyjmenováním názvů servlet tříd (viz kód 2.2). Filtry slouží k zachycování a zpracovávání požadavků před jejich odesláním k samotným servlet třídám, dále také v opačném směru k zachytávání odpovědí poté, co je proveden kód servlet třídy, a zároveň před zpracováním odpovědi kontejnerem aplikačního serveru.

Kód 2.1: Deklarace servlet filtru v *web.xml* [33]

```
<filter>
  <filter-name>RequestLoggingFilter</filter-name> <!-- mandatory -->
  <filter-class>com.example.servlet.filters.RequestLoggingFilter
    </filter-class> <!-- mandatory -->
  <init-param> <!-- optional -->
    <param-name>test</param-name>
    <param-value>testValue</param-value>
  </init-param>
</filter>
```

Kód 2.2: Mapování servlet filtru v *web.xml* [33]

```
<filter-mapping>
  <filter-name>RequestLoggingFilter</filter-name> <!-- mandatory -->
  <url-pattern>/*</url-pattern>
  <!-- either url-pattern or servlet-name is mandatory -->
  <servlet-name>LoginServlet</servlet-name>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

Filtry se používají k logování, úpravě těl či hlaviček požadavků nebo také k autentizaci a autorizaci [33]. Použití servlet filtrů se nabízí jako potenciální řešení.

Rozpoznání chráněného zdroje lze provést v rámci HTTP protokolu a jeho stavových kódů. Při pokusu o přístup k nechráněnému zdroji s nepřihlášeným principle v security kontextu by klient obdržel HTTP stavový kód 401 Unauthorized. Jako reakce na tuto odpověď by se vytvořila potřebná SAML zpráva, která by se dále odeslala na Shibboleth IdP. Inicializace autentizace pomocí Shibboleth by tedy potřebovala dva HTTP požadavky.

Po úspěšné autentizaci na IdP, navrácení se do aplikace, zpracování a ověření přijaté SAML zprávy by mělo následovat přihlášení získané identity v rámci security kontextu kontejneru serveru. Uvnitř filtru lze pomocí volání *HttpServletRequest.getUserPrincipal()* získat aktuálně autentizovaného principle. Problémem však je nastavení této hodnoty, tedy provedení přihlášení přímo z filtru. Dostupná metoda *HttpServletRequest.login(username, password)* vyžaduje uživatelské jméno a heslo, které ověřuje pomocí určitého realm proti jemu dostupným identitám [34]. Shibboleth IdP však přihlašovací údaje z bezpečnostních důvodů nikam neodesílá. Filtry bohužel neřeší dříve zmíněný problém realm, tedy nutnost použití standardních metod HTTP autentizace. Z tohoto důvodu nelze filtrem provést autentizaci pomocí Shibboleth.

### 2.1.2 SAM

Na základě první kapitoly se jeví jako jediný způsob řešení implementace SAM. Modul řeší problém filtru, respektive realm, jelikož SPI poskytuje rozhraní pro integraci autentizace s kontejnerem aplikačního serveru. Vlastní SAM je implementací *javax.security.auth.message.module.ServerAuthModule* rozhraní, které dědí od *javax.security.auth.message.ServerAuth* rozhraní a společně poskytují mnoho možností [35].

Rozpoznání chráněného zdroje je dostupné již při inicializaci modulu pomocí instance třídy *MessagePolicy*, která definuje autentizační politiku v rámci zprávy (dvojice *HttpServletRequest*, *HttpServletResponse*) a obsahuje atribut *mandatory*, který říká, zda se politika má uplatnit či nikoliv [36].

Získání aktuálně přihlášeného principle v kontextu aplikace je stejné jako u filtru, tedy pomocí volání *HttpServletRequest.getUserPrincipal()*. Přihlášení nového principle provádí *CallbackHandler* s pomocí *CallerPrincipalCallback* určeného pro použití právě v SAM a umožňujícího nastavení autentizovaného principle v kontejneru [37].

Vzhledem k tomu, že se SAM volá pro každý přístup k jakémukoliv zdroji, nic nebrání ani opuštění aplikace a opětovnému navrácení se spolu s následnou autentizací identity v kontejneru. Modul řeší problémy jiných návrhů řešení a jeho implementaci se věnuje třetí kapitola.

## 2.2 Komunikace

Tvorba a zpracování SAML zpráv potřebných pro komunikaci mezi Shibboleth komponentami je důležitou funkcionalitou SP. Jako první se nabízí možnost ručního vytváření příslušných SAML tvrzení, správného zakódování a jejich vložení do odesílaných požadavků. Realizace takového řešení by byla však poměrně náročná a dávala by zároveň velký prostor pro chyby. Pro eliminaci této nevýhody slouží hotové knihovny, které poskytují potřebnou funkčnost. Java knihoven pro jazyk SAML existuje více a výběru té nejvhodnější se věnují následující řádky.

### 2.2.1 OpenSAML

Opět se lze inspirovat frameworkem Spring Security SAML, který používá, ač jako tranzitivní závislost, knihovnu OpenSAML [38] [30]. Knihovna umí vytvářet SAML zprávy, zpracovávat a exportovat SAML objekty jako XML, podporuje podepisování spolu s šifrováním zpráv a konečně také zakódování a přenos SAML zpráv. Vývoj knihovny zastřešuje stejně jako projekt Shibboleth americká asociace Internet2. Shibboleth je tedy jedním z SSO řešení, které knihovna podporuje. Existuje implementace v jazycích C++ a Java. OpenSAML podporuje SAML verze 2.0, 1.1 a 1.0. Slabou stránkou knihovny je její dokumentace. Knihovna je dostupná pod licencí Apache 2.0 [39].

Kód 2.3: Maven závislost na OpenSAML (*pom.xml*)

```
<dependency>
  <groupId>org.opensaml</groupId>
  <artifactId>opensaml</artifactId>
  <version>2.6.4</version>
</dependency>
```

### 2.2.2 OneLogin Java SAML toolkit

Další dostupnou utilitou pro práci s jazykem SAML je OneLogin Java SAML toolkit (dostupné na GitHub [40]). Společnost OneLogin primárně poskytuje vlastní SSO řešení a správce identit, nicméně její produkty také využívají pro komunikaci jazyk SAML. Pro vývojáře společnost poskytuje jednoduchou utility knihovnu napsanou v různých jazycích, mezi kterými nechybí Java. Knihovna však poskytuje pouze rozhraní pro jednoduchou tvorbu a zpracování autentizačních SAML zpráv [41] [40]. Právě kvůli omezené funkčnosti nelze knihovnu dále použít v bakalářské práci.

### 2.2.3 LastPass SAML SDK for Java

Společnost LastPass se zabývá centralizací přístupů ke službám a prací s přihlašovacími údaji [42]. Obdobně jako OneLogin poskytuje vývojářskou sadu nástrojů pro jednoduchou práci s SSO systémy založenými na jazyku SAML 2.0. Software development kit (SDK) je postavený na knihovně OpenSAML a je dostupný pod licencí Apache 2.0 na GitHub [43]. Vzhledem k přímé závislosti na OpenSAML není důvod k použití této knihovny, která neposkytuje rozšířené možnosti, ale pouze upravuje rozhraní knihovny jiné.

### 2.2.4 Volba knihovny

Z dostupných řešení byla pro implementaci vybrána knihovna OpenSAML v poslední verzi 2.6.4 (viz Maven závislost 2.3), která poskytuje nejvíce možností. OpenSAML není žádná novinka, připravuje se již třetí verze. Současná verze je prověřena několika produkty, jež ji využívají, ať už framework Spring Security SAML nebo knihovna LastPass SAML SDK for Java.

Kapitola 3

# Realizace

Kapitola se podrobně věnuje realizaci návrhu z kapitoly předchozí, a sice SAM využívajícího knihovnu OpenSAML pro práci se SAML zprávami. Podrobný popis SAM je uveden v kapitolách 1.2.1.3 a 2.1.2, informace o knihovně OpenSAML jsou pak v kapitole 2.2.1.

## 3.1 Autentizace

Jako náhrada za Shibboleth komponentu poskytovatele služeb byl v předchozí kapitole zvolen SAM, jehož implementaci se věnují následující odstavce.

### 3.1.1 SAM projekt

Pro správu projektu implementujícího SAM byl použit nástroj Apache Maven verze 3 [44]. K dalšímu vývoji jsou nezbytné závislosti na Java EE API verze 7 a OpenSAML verze 2.6.4 (viz kód 2.3). Výsledný modul musí být ve formátu *jar* uložený v pevně daném adresáři v rámci GlassFish serveru, konkrétně v *<gf-install-dir>/glassfish/domains/<domain-name>/lib*. Vzhledem k tomu, že modul je závislý na již zmíněných knihovnách, je potřeba, aby měl tyto závislosti k dispozici i na aplikačním serveru, na kterém bude spouštěn. Zejména knihovna OpenSAML není běžně k dispozici.

K sestavení projektu byl v bakalářské práci použit Apache Maven Assembly Plugin verze 2.6 [45], za účelem vytvoření jednoho výstupního *jar* souboru obsahujícího zkompilované *class* soubory jak projektu, tak všech jeho závislostí [46]. Tímto způsobem jsou potřebné závislosti poskytnuty kontejneru společně s modulem. Konfiguraci Maven pluginu zobrazuje kód 3.1.

### 3.1.2 SAM konfigurace

Nezbytným krokem je následné zaregistrování modulu na serveru. První možností registrace je využití grafické konzole GlassFish serveru defaultně

Kód 3.1: Maven Assembly plugin (*pom.xml*)

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>2.6</version>
  <configuration>
    <descriptorRefs>
      <descriptorRef>jar-with-dependencies</descriptorRef>
    </descriptorRefs>
  </configuration>

  <executions>
    <execution>
       <id>make-assembly</id>
       <phase>package</phase>
       <goals>
         <goal>single</goal>
       </goals>
    </execution>
  </executions>
</plugin>
```

dostupné na portu 4848. Příklad registrace modulu v konzoli zobrazuje obrázek 3.1. Alternativou je použití příkazu *asadmin* sloužícího ke konfiguraci domény serveru. Oba způsoby registrace vytvoří odpovídající záznam v hlavním konfiguračním souboru domény *domain.xml*. Vzor této konfigurace spolu s atributy ukazuje kód 3.2. Posledním krokem je volba konkrétního SAM. Modul lze nastavit jako výchozí pro celou doménu, možnost lze najít na obrázku 3.1. Vhodnějším řešením je však volba určitého modulu přímo v dané aplikaci pomocí souboru *glassfish-web.xml* či *sun-web.xml*, kterou zobrazuje kód 1.3 [47]. Další možnosti tohoto konfiguračního souboru budou zmíněny v následujících kapitolách.

### 3.1.3 Implementace ServerAuthModule rozhraní

Vytvoření vlastního autentizačního modulu spočívá v implementování *javax.security.auth.message.module.ServerAuthModule* dědícího od rozhraní *javax.security.auth.message.ServerAuth*. Ve výsledku to znamená implementaci následujících pěti metod.

#### 3.1.3.1 Initialize

Provádí inicializaci modulu a poskytuje důležitá data (viz kapitola 2.1.2). Mezi data patří *CallbackHandler* pro přihlášení, dále informace zda má modul vynucovat autentizační politiku vázanou k příchozí zprávě a konečně také


Obrázek 3.1: GlassFish konzole: SAM konfigurace

Kód 3.2: Konfigurace SAM v souboru *domain.xml*

```
<provider-config provider-type="server"
  provider-id="CustomShibbolethSAM"
  class-name="cz.cvut.fit.shibboleth.CustomServerAuthModule">
  <property description="AssertionConsumerServiceURL" name="ACS_URL"
      value="https://localhost:8181/">
  </property>
  <property description="SSO redirect endpoint" name="SSO_ENDPOINT"
    value="https://localhost:8443/idp/profile/SAML2/Redirect/SSO">
  </property>
  <property description="Issuer for AuthnRequest" name="ISSUER"
    value="https://users-test-app.com/shibboleth">
  </property>
  <property description="Keystore password" name="KEYSTORE_PASSWORD"
      value="changeit">
  </property>
  <property description="Keystore absolute location"
    name="KEYSTORE_PATH"
    value="/home/malecka1/Documents/spCredentials/SPkeystore.jks">
  </property>
  <property description="Private key alias" name="PRIVATE_KEY_ALIAS"
      value="users-test-app.com">
  </property>
  <property description="Private key password"
    name="PRIVATE_KEY_PASSWORD" value="changeit">
  </property>
  <property description="IdP's certificate absolute location"
    name="IDP_CRT_PATH"
    value="/home/malecka1/Documents/spCredentials/idp.crt">
  </property>
  <request-policy></request-policy>
  <response-policy></response-policy>
</provider-config>
```

Kód 3.3: Explicitní registrace session

```
messageInfo.getMap().put("javax.servlet.http.registerSession",
                         Boolean.TRUE.toString());
```

mapu obsahující konfiguraci modulu, které se věnuje následující sekce 3.2. Všechna zmíněná data jsou uchována jako třídní atributy pro použití v dalších metodách.

#### 3.1.3.2 getSupportedMessageTypes

Metoda vracející pole tříd, jež jsou podporované modulem. V tomto případě se jedná o *HttpServletRequest* a *HttpServletResponse* [48].

#### 3.1.3.3 validateRequest

Nejdůležitější metoda obsahující veškerou autentizační logiku. Je volána pro každý požadavek a využívá atributů získaných při inicializaci modulu. V bakalářské práci existují tři možné scénáře. Jejich rozpoznání v následujícím pořadí je důležité, aby nedocházelo například k opětovné autentizaci.

1. Již přihlášený principal: První možnost, bez ohledu na chráněnost zdroje. Pomocí *HttpServletRequest.getSession(true).getUserPrincipal()* se získá aktuálně přihlášený principal. Pokud není *null*, pokračuje se využitím této identity. Návratovou hodnotou je vždy *AuthStatus.SUCCESS*. Hodnota říká, že request byla úspěšně validována.

2. Nechráněný zdroj: V kontextu není autentizovaný principal, nicméně cílový zdroj je nechráněný a zároveň se nečeká na odpověď od IdP. V tomto případě se pokračuje bez přihlášení jako anonymní principal. Vrací také *AuthStatus.SUCCESS*.

3. Chráněný zdroj: Zde je třeba provést Shibboleth autentizaci. Lze rozdělit na dvě situace:

   a) Odeslání SAML AuthnRequest: Vytvoření je přiblížené v další sekci 3.2.1. Návratovou hodnotou je *AuthStatus.SEND_CONTINUE* indikující nedokončené ověření zprávy.

   b) Příjem SAML Response: Zpracování se věnuje sekce 3.2.2. Po úspěšném zpracování následuje navrácení *AuthStatus.SUCCESS*.

      Vzhledem k bezestavovosti SAM je nutné získané údaje uložit. Tato vlastnost je důležitá pro budoucí rozhodování o přístupu a zabránění opakované autentizaci pro každý zdroj. Zaregistrování session se však musí kontejneru explicitně nastavit zavoláním kódu 3.3.

Kód 3.4: Mapování skupin na role v souboru *glassfish-web.xml*

```
<security-role-mapping>
  <role-name>admin</role-name>
  <group-name>administrators</group-name>
</security-role-mapping>

<security-role-mapping>
  <role-name>user</role-name>
  <group-name>users</group-name>
</security-role-mapping>
```

V případě úspěšné Shibboleth autentizace a přihlášení pomocí *CallbackHandler* kontejner provádí autorizaci na základě mapování získaných skupin na role definované v aplikaci. Mapování se nachází v souboru *glassfish-web.xml* a ukazuje ho kód 3.4.

V případě chyby při vytváření SAML AuthnRequest či jiných chyb metoda vyhazuje výjimku *AuthException* symbolizující neúspěšné zpracování zprávy bez vytvoření odpovídající chybové response.

#### 3.1.3.4 secureResponse

Slouží k zabezpečení response, je volána jako úplně poslední před odesláním. Jejím výstupem je stejně jako u předchozí metody buď autentizační status, nebo vyhození výjimky *AuthException* v případě chyby. Práce metodu nijak nevyužívá a pouze vrací *AuthStatus.SEND_SUCCESS* indikující úspěšné zabezpečení response.

#### 3.1.3.5 cleanSubject

Druhá metoda obsahující logiku slouží k odebrání principle z kontextu. Volání probíhá při odhlašování uživatele. Implementace v práci provádí kromě zavolání *HttpServletRequest.logout()* také invalidaci existující session. V případě chyby v průběhu procesu odhlášení vyhodí stejně jako předchozí metody výjimku *AuthException*. Obsah metody lze vidět v kódu 3.5 [49].

## 3.2 SAML zprávy

Na základě první kapitoly, kde byl proveden rozbor několika typů SAML zpráv, byla vytvořena třída *SAMLFactory.java*. Třída vystupuje jako předek dalším třídám, respektive potomkům, jež pracují s konkrétními typy zpráv. Příkladem může být třída *AuthenticationService.java*, která v bakalářské práci pracuje s autentizačními zprávami.

Kód 3.5: Logika metody *ServerAuthModule.cleanSubject()*

```
try {
    request.logout();
} catch (ServletException e) {
    LOGGER.warning("Could not logout.");
    throw new AuthException();
}
final HttpSession session = request.getSession(false);
if (session != null) {
    session.invalidate();
}
```

Kód 3.6: Inicializace OpenSAML knihovny (*SAMLFactory.java*)

```
public SAMLFactory() {
  try {
     DefaultBootstrap.bootstrap();
  } catch (ConfigurationException e) {
     throw new RuntimeException("Cannot initialize OpenSAML library.",
          e);
  }
}
```

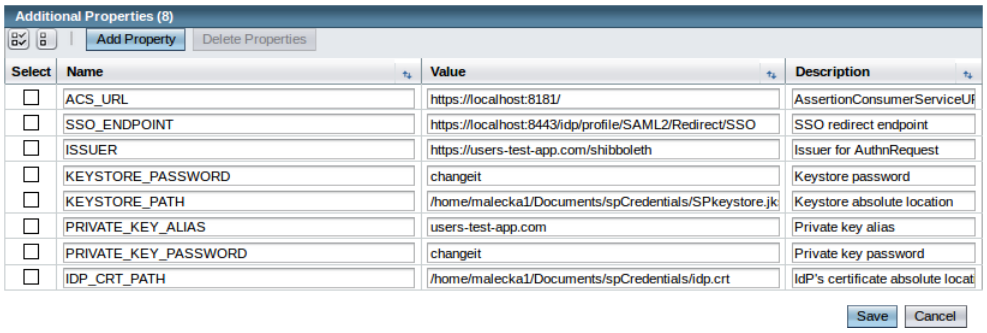
Kód 3.7: Vytváření XML objektů (*SAMLFactory.java*)

```
  protected <T> T create(Class<T> tClass, QName qname) {
    return (T) Configuration.getBuilderFactory().getBuilder(qname).
        buildObject(qname);
  }
```

Úkolem předka je správná inicializace knihovny OpenSAML a poskytnutí nástroje pro tvorbu jednotlivých XML objektů. Implementaci metod zobrazuje kód 3.6, respektive kód 3.7.

Pro zajištění komunikace modulu s IdP prostřednictvím SAML zpráv je potřeba poskytnout IdP metadata aplikace, namísto metadat samostatné SP komponenty v nativním řešení. Metadatům se blíže věnuje čtvrtá kapitola při příležitosti realizace konkrétního testovacího scénáře.

S hodnotami v metadatech úzce souvisí nejenom obsah posílaných SAML zpráv, správnou konfiguraci proto potřebují oba komunikující subjekty. V referenční implementaci se konfigurace SP předává ve formě dalších metadat. Vzhledem k tomu, že je SAM součástí aplikačního serveru, lze využít konfigurační mapu dostupnou v jeho inicializační metodě (viz sekce 3.1.3.1). Hodnoty lze nastavit pomocí GlassFish konzole, kterou zachycuje obrázek 3.2. Konfigurace se opět projeví v souboru *domain.xml*, jak lze vidět v kódu 3.2.




Obrázek 3.2: GlassFish konzole: SAM konfigurace atributů

Za účelem autentizace byla v práci vytvořena již zmíněná třída *AuthenticationService.java*, dědící od *SAMLFactory.java*. Třída má za úkol vytváření a zpracovávání autentizačních SAML zpráv. V jejím konstruktoru je volán konstruktor předka, který zajišťuje správnou inicializaci knihovny OpenSAML. Druhým úkolem konstruktoru je získání hodnot z konfigurační mapy pomocí pevně daných klíčů.

### 3.2.1 SAML Request

Vytvoření a odeslání zakódované SAML AuthnRequest iniciuje proces autentizace v souladu s Web Browser SSO Profile 1.1.3. Sekce se věnuje implementaci několika důležitých položek této zprávy. Reálný příklad implementace ukazuje kód 3.8 a výstup v podobě celé SAML Request je uveden v závěrečné kapitole v kódu 4.15.

Zpráva musí obsahovat element *Issuer*, jehož hodnotou musí být unikátní identifikátor SP (aplikace). Hodnota se musí shodovat s povinným atributem *entityID* některých z metadat, jež má IdP k dispozici. Ten zároveň ověřuje, zda hodnota ACS v nalezených metadatech odpovídá *AssertionConsumerServiceURL* atributu zprávy [50].

#### 3.2.1.1 ID

Slouží jako identifikátor jednotlivých tvrzení, respektive SAML Request v tomto případě. Hodnota musí zaručovat mizivou pravděpodobnost, že stejnou hodnotu bude mít jiné tvrzení. Specifikace nepředepisuje způsob dosažení takových identifikátorů a nechává ho na implementaci [51]. V implementaci byl použit XORShift pseudogenerátor a jako seed posloužilo volání *System.nanoTime()*. Perioda tohoto pseudogenerátoru je $2^n - 1$ pro n-bitový seed. Podávané výsledky jsou znatelně lepší než výstupy třídy *java.util.Random*, avšak horší než *java.security.SecureRandom*, který je ale několikanásobně pomalejší [52]. Vygenerovaná hodnota byla dále zahashována funkcí SHA-1, která vytváří 160 bitů dlouhý výsledek [53].

Kód 3.8: Tvorba SAML Request (*AuthenticationService.java*)

```
// AuthnRequest
final AuthnRequest authnRequest = create(AuthnRequest.class,
    AuthnRequest.DEFAULT_ELEMENT_NAME);
authnRequest.setAssertionConsumerServiceURL(ACS_URL);

// ID
final String idHash = getSHAHash(getRandomLong());
request.getSession(false).setAttribute(ID_SESSION_NAME, idHash);
authnRequest.setID(idHash);

// Date, version
authnRequest.setIssueInstant(new DateTime());
authnRequest.setVersion(SAMLVersion.VERSION_20);

// Issuer
final Issuer issuer = create(Issuer.class,
    Issuer.DEFAULT_ELEMENT_NAME);
issuer.setValue(ISSUER);
authnRequest.setIssuer(issuer);
```

#### 3.2.1.2 Parametr RelayState

Druhý URL parametr *RelayState* slouží k uložení adresy požadovaného zdroje. IdP nesmí měnit hodnotu tohoto parametru a musí ji připojit k SAML Response, kde bude použita jako cílová URL. Aby nebyl posílán pouze otevřený text, implementace používá blokovou šifru AES v operačním módu CBC. Jako 192 bitový klíč byla použita část GlassFish session ID, které je dlouhé 28 bytů. Vygenerovaný inicializační vektor byl uložen do session pro budoucí dešifrování. Samotné šifrování však nezajišťuje integritu dat a nechrání tak data proti neautorizované změně [54]. Přesměrování se na chybnou URL však nepředstavuje bezpečnostní riziko. Nakonec je parametr zakódován pomocí base64 kódování [16].

### 3.2.2 SAML Response

Druhým úkolem třídy *AuthenticationService.java* je zpracování SAML Response. Obsah zprávy je však závislý na nastavení IdP. Následující postup tedy odpovídá reálnému procesu použitému ve čtvrté kapitole při příležitosti testování. Odpověď je reakcí na SAML Request popsanou v předchozích odstavcích.

Ještě před zahájením zpracování samotné SAML zprávy přichází na řadu druhý parametr *RelayState*. Parametr musí být v nezměněné formě součástí odpovědi [16]. Hodnota je nejprve dekódována a poté pomocí session ID a inicializačního vektoru uloženého v session dešifrována. Tím je získána URL

Kód 3.9: Dešifrování SAML Response (*AuthenticationService.java*)

```
// Create decrypter
final BasicX509Credential decryptionCredential =
    new BasicX509Credential();
decryptionCredential.setPrivateKey(privateKey);
final StaticKeyInfoCredentialResolver keyInfoCredentialResolver =
    new StaticKeyInfoCredentialResolver(decryptionCredential);
final Decrypter decrypter = new Decrypter(null,
    keyInfoCredentialResolver, new InlineEncryptedKeyResolver());
decrypter.setRootInNewDocument(Boolean.TRUE); // decrypted Assertion
    will have properly rooted DOM that allow signature verification!

// Decrypt assertion
final Assertion decryptedAssertion;
try {
  decryptedAssertion = decrypter.decrypt(samlResponse.
      getEncryptedAssertions().get(0));
} catch (DecryptionException e) {
  LOGGER.warning("Assertion decryption failed.");
  throw new AuthException();
}
```

cílového zdroje, na kterou se aplikace přesměruje v případě úspěšného přihlášení v aplikaci.

Následujícím krokem je dekódování SAML Response zakódované pomocí base64. Výsledek lze vidět v kódu 4.16. Základní validací je kontrola atributu *inResponseTo*, jehož hodnota musí odpovídat *ID* atributu SAML Request, který je uložen v session. Díky tomu se začne zpracovávat správná SAML Response.

Vzhledem k použití šifrování i podpisu je potřeba začít dešifrováním zprávy pomocí privátního klíče aplikace (viz kód 3.9). Poté následuje ověření podpisu s využitím veřejného klíče IdP. Oba klíče třída získává z konfigurační mapy.

Posledním krokem je extrakce atributů z *AttributeStatement* části rozšifrované zprávy, kterou zobrazuje kód 3.10. Získání se provádí na základě SAML atributů. Získané hodnoty jména a skupin jsou dále v SAM použity k přihlášení.

Kód 3.10: Atributy v SAML Response

```
<saml2:AttributeStatement>
  <saml2:Attribute FriendlyName="uid"
    Name="urn:oid:0.9.2342.19200300.100.1.1"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:type="xs:string">user1</saml2:AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="employeeType"
    Name="urn:oid:2.16.840.1.113730.3.1.4" NameFormat="
        urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml2:AttributeValue
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:type="xs:string">users</saml2:AttributeValue>
    <saml2:AttributeValue
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="xs:string">teachers</saml2:AttributeValue>
  </saml2:Attribute>
</saml2:AttributeStatement>
```

Kapitola 4

# Otestování

Součástí zadání je ověření funkčnosti implementovaného řešení a právě tomuto bodu se věnuje čtvrtá kapitola.

## 4.1 Komponenty

K otestování byl použit následující software:

- GlassFish verze 4.1
- Shibboleth IdP verze 2.4.4 + Tomcat verze 8.0.28
- ApacheDS verze 2.0.0-M21

Pro simulaci reálného prostředí byla vytvořena jednoduchá Java EE aplikace zobrazující seznamy dní a měsíců. K realizaci byly použity především technologie JavaServer Faces (JSF) 2.2 a Enterprise JavaBeans (EJB) 3.2, které jsou součástí Java EE 7 specifikace. Aplikace obsahuje zabezpečené zdroje pomocí rolí definovaných v deployment descriptoru *web.xml* (viz kód 4.1). Ukázku konfigurace omezení přístupu z tohoto souboru lze vidět v kódu 4.2. EJB moduly jsou zároveň zabezpečené pomocí anotací na úrovni metod (viz zdrojový kód souboru *PrintBean.java* 4.3). Aplikace běží na GlassFish doméně používající TLS. Bez ohledu na obsah lze aplikaci považovat za plnohodnotnou Java EE aplikaci pro účely otestování Shibboleth autentizace. Náhled poskytuje obrázek 4.1.

## 4.2 Konfigurace

Sekce přibližuje nastavení a propojení jednotlivých komponent vyjma klientské aplikace představené v předchozí sekci.

Kód 4.1: Definice aplikačních rolí v souboru *web.xml*

```
<security-role>
  <role-name>user</role-name>
</security-role>

<security-role>
  <role-name>admin</role-name>
</security-role>
```

Kód 4.2: Omezení přístupu ke zdroji v souboru *web.xml*

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Days section</web-resource-name>
        <description/>
        <url-pattern>/webpages/private/days/*</url-pattern>
    </web-resource-collection>

    <auth-constraint>
        <role-name>user</role-name>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>
```

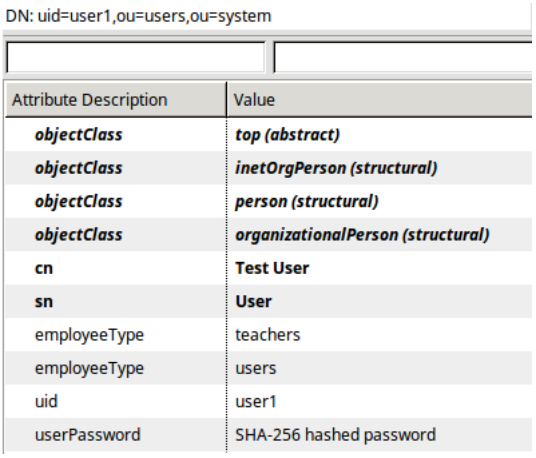
Kód 4.3: Zabezpečení EJB metod anotacemi (*PrintBean.java*)

```
/**
 * Simple view bean providing simple data.
 */
@Named
@Stateless
public class PrintBean {

    @RolesAllowed({"user", "admin"})
    public List<DayOfWeek> getDays() {
        return Arrays.asList(DayOfWeek.values());
    }

    @RolesAllowed("admin")
    public List<Month> getMonths() {
        return Arrays.asList(Month.values());
    }
}
```


Obrázek 4.1: Testovací aplikace: *admin1*, přístup povolen




Obrázek 4.2: LDAP záznam (ApacheDS)

### 4.2.1 LDAP

ApacheDS [55] byl použit jako LDAPS server sloužící k ověřování identit. Jedná se o LDAP s přenosem zabezpečeným pomocí SSL. V rámci LDAP byla vytvořena organizační jednotka *users*, v níž byli vytvořeni testovací uživatelé *user1* a *admin1* (objekty typu *inetOrgPerson*). Společně s přidáním atributu *userPassword* byl přidán atribut *employeeType* vyjadřující příslušnost k určité skupině. Atribut může být přítomný vícekrát. Testovacím uživatelům byly přiřazeny skupiny *users* a *teachers*, respektive *administrators* v případě uživatele *admin1*. Výsledný záznam je na obrázku 4.2. Právě zde definované skupiny jsou nakonec použity při mapování skupin na role v samotné Java aplikaci, jak ukazuje kód 3.4.

Kód 4.4: Tomcat konektor využívající TLS (*server.xml*)

```
<Connector port="8443" protocol="HTTP/1.1" maxThreads="150"
      SSLEnabled="true" scheme="https" secure="true"
      clientAuth="false" sslProtocol="TLS"
      keystoreFile="/home/malecka1/idpself.keystore"
      keystorePass="malecka1" />
```

Kód 4.5: IdP *LoginHandler* (*handler.xml*)

```
<ph:LoginHandler xsi:type="ph:UsernamePassword"
  jaasConfigurationLocation="file:///home/malecka1/idp-2.4.4/conf/
      login.config">
  <ph:AuthenticationMethod>urn:oasis:names:tc:SAML:2.0:ac:classes:
    PasswordProtectedTransport</ph:AuthenticationMethod>
</ph:LoginHandler>
```

### 4.2.2 Shibboleth IdP

Po instalaci stažené komponenty verze 2.4.4 vznikne *war* soubor určený k nasazení na aplikační server. Oficiálně podporovanými aplikačními servery pro IdP jsou Jetty alespoň verze 7 [56] a Tomcat [57] verze 6 a vyšší. V práci byl zvolen Tomcat verze 8.0.28. Pro potřeby IdP byl nakonfigurován zvláštní konektor používající TLS (viz kód 4.4) [58].

Nezbytnou konfigurací je napojení na vytvořený LDAP za účelem ověřování identit získaných od uživatelů. K tomu slouží takzvaný *LoginHandler* v rámci IdP. Kód 4.5 zobrazuje konfiguraci, která používá dvojici přihlašovací jméno a heslo. Z konfigurace je patrné, že vlastní přihlašování řeší JAAS (viz kapitola o realm 1.2.1.1) [59]. Kód 4.6 *LdapLoginModule* se připojí k LDAP a podle umístění získá *userAttribute*, v podstatě namapované uživatelské jméno [60]. Zajímavostí jsou dvě po sobě jdoucí připojení k LDAP a mapování *userRoleAttribute*. Při nahlédnutí do IdP logu 4.7 lze vidět, že JAAS tento atribut získal, nicméně JAAS opouští nově vytvořený *LoginContext* [61] obsahující instanci třídy *Subject* [62], součástí které je přihlášený principal. To je objekt třídy implementující standardní rozhraní *java.security.Principal* [63], které však obsahuje pouze atribut pro jméno. Vytváření vlastní implementace rozhraní *Principal* by nebylo pro další práci IdP vhodné, třída by nebyla nijak standardizována vzhledem k různým SAML definicím požadovaných atributů [64]. Následuje vytvoření session pro principal [65].

Druhé připojení realizuje *DataConnector* nakonfigurovaný v kódu 4.8 [66]. Ten již umožňuje IdP získat požadovaná data v závislosti na definici množiny SAML atributů v souboru *attribute-resolver.xml*, jak ukazuje kód 4.10 a tomuto procesu odpovídající log 4.9.

Dalším krokem zpracování získaných atributů je jejich definice v rámci SAML Attribute statements. Kód 4.11 z konfigurace IdP zobrazuje vytvoření

Kód 4.6: JAAS LDAP login modul (*login.conf*)

```
edu.vt.middleware.ldap.jaas.LdapLoginModule required
  ldapUrl="ldaps://localhost:10636"
  ssl="true"
  baseDn="ou=users,ou=system"
  bindDn="uid=admin,ou=system"
  bindCredential="secret"
  userAttribute="uid"
  userRoleAttribute="employeeType";
```

Kód 4.7: IdP log: JAAS

```
[JaasAuthenticator] - Authentication succeeded for dn:
  uid=user1,ou=users,ou=system
[JaasAuthenticator] - Returning attributes:
[JaasAuthenticator] - [employeeType]
[SearchDnResolver] - Looking up DN using userField
[SearchDnResolver] - Search with the following parameters:
[SearchDnResolver] - dn = ou=users,ou=system
[SearchDnResolver] - filter = (uid={0})
[SearchDnResolver] - filterArgs = [user1]
[LdapLoginModule] - Committed the following principals:
  [user1[employeeType[users, teachers]]]
[LdapLoginModule] - Committed the following roles: [teachers, users]
[UsernamePasswordLoginServlet] - Successfully authenticated user user1
[AuthenticationEngine] - Returning control to authentication engine
[AuthenticationEngine] - Completing user authentication process
[AuthenticationEngine] - Create shibboleth session for principal user1
```

Kód 4.8: Konfigurace LDAP data konektoru (*attribute-resolver.xml*)

```
<resolver:DataConnector id="myLDAP" xsi:type="dc:LDAPDirectory"
  ldapURL="ldaps://localhost:10636"
  baseDN="ou=users,ou=system"
  principal="uid=admin,ou=system"
  principalCredential="secret">

  <dc:FilterTemplate>
    <![CDATA[ (uid=$requestContext.principalName) ]]>
  </dc:FilterTemplate>
</resolver:DataConnector>
```

Kód 4.9: IdP log: *AttributeResolver*

```
[AbstractSAML2ProfileHandler] - Resolving attributes for principal
  'user1' for SAML request from relying party
  'https://users-test-app.com/shibboleth'
[ShibbolethAttributeResolver] - Resolving attribute employeeType
[ShibbolethAttributeResolver] - Resolving data connector myLDAP
[LdapDataConnector] - Search filter: (uid=user1)
[LdapDataConnector] - myLDAP - Retrieving attributes from LDAP
[LdapDataConnector] - myLDAP - Found: uid[user1]
[LdapDataConnector] - myLDAP - Found: employeeType[users, teachers]
```

Kód 4.10: Zpracování atributů získaných z LDAP (*attribute-resolver.xml*)

```
<resolver:AttributeDefinition xsi:type="ad:Simple" id="uid"
    sourceAttributeID="uid">
  <resolver:Dependency ref="myLDAP" />
  <resolver:AttributeEncoder xsi:type="enc:SAML1String"
    name="urn:mace:dir:attribute-def:uid" />
  <resolver:AttributeEncoder xsi:type="enc:SAML2String"
    name="urn:oid:0.9.2342.19200300.100.1.1" friendlyName="uid" />
</resolver:AttributeDefinition>

<resolver:AttributeDefinition xsi:type="ad:Simple" id="employeeType"
    sourceAttributeID="employeeType">
  <resolver:Dependency ref="myLDAP" />
  <resolver:AttributeEncoder xsi:type="enc:SAML1String"
    name="urn:mace:dir:attribute-def:employeeType" />
  <resolver:AttributeEncoder xsi:type="enc:SAML2String"
    name="urn:oid:2.16.840.1.113730.3.1.4"
    friendlyName="employeeType" />
</resolver:AttributeDefinition>
```

skupiny atributů, jejichž distribuce může být omezena pravidly jak na úrovni celé skupiny, tak jednotlivých atributů [67].

Odesílání SAML Attribute statements v SAML zprávách je potřeba povolit pro daný SAML profil. Ukázka z testovací instalace IdP 4.12 povoluje zahrnutí tohoto druhu SAML prohlášení. V kódu lze vidět i další konfiguraci, jako vynucení podepisování či šifrování SAML prohlášení. Testovací scénář používá toto nastavení vzhledem k jeho zpětnému odvození ze SAML komunikace na FIT ČVUT.

### 4.2.3 Metadata

Poslední konfigurací je vzájemná výměna metadat ve formátu *xml* mezi aplikací a IdP. Detailní popis jednotlivých atributů přesahuje rámec bakalářské

Kód 4.11: Definice politiky o poskytování skupiny atributů (*attribute-filter.xml*)

```
<afp:AttributeFilterPolicy id="LDAPcredentials">
  <afp:PolicyRequirementRule xsi:type="basic:ANY" />

  <afp:AttributeRule attributeID="uid">
    <afp:PermitValueRule xsi:type="basic:ANY" />
  </afp:AttributeRule>

  <afp:AttributeRule attributeID="employeeType">
    <afp:PermitValueRule xsi:type="basic:ANY" />
  </afp:AttributeRule>
</afp:AttributeFilterPolicy>
```

Kód 4.12: Nastavení profilu (*relying-party.xml*)

```
<rp:ProfileConfiguration xsi:type="saml:SAML2SSOProfile"
  includeAttributeStatement="true" assertionLifetime="PT5M"
  assertionProxyCount="0" signResponses="never"
  signAssertions="always" encryptAssertions="always"
  encryptNameIds="never" includeConditionsNotBefore="true" />
```

Kód 4.13: Konfigurace metadat testovací aplikace (*relying-party.xml*)

```
<metadata:MetadataProvider id="users-test-app-metadata"
  xsi:type="FilesystemMetadataProvider"
  xmlns="urn:mace:shibboleth:2.0:metadata" metadataFile="/home/
      malecka1/idp-2.4.4/metadata/users-test-app-metadata.xml" />
```

práce a proto zde budou zmíněny pouze pro práci důležité položky. Podrobnosti k metadatům lze najít na webu OASIS [68].

Hodnoty z IdP metadat automaticky vytvořených při instalaci jsou poskytnuty aplikaci, respektive SAM, pomocí GlassFish konzole dle sekce 3.2. Uvnitř metadat IdP je důležitou položkou certifikát standardu X.509 sloužící k ověření podpisu zpráv přijatých od IdP. Nezbytná je také *SingleSignOnService* poskytující endpoint pro HTTP-Redirect binding používaný modulem v rámci Web Browser SSO Profile 1.1.3.

Metadata aplikace jsou IdP předávána pomocí souboru, jejž si při startu načítá díky konfiguraci 4.13. Povinným atributem je *entityID*. Pro realizaci testovacího scénáře jsou dále potřeba X.509 certifikát, sloužící k zašifrování odchozích zpráv od IdP, a ACS pro HTTP-POST binding, na který jsou zprávy odesílány [69]. Metadata testovací aplikace ukazuje kód 4.14.

Kód 4.14: Metadata testovací aplikace

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
  ID="https://users-test-app.com/shibboleth"
  entityID="https://users-test-app.com/shibboleth">
  <md:SPSSODescriptor
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <md:KeyDescriptor>
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:X509Data>
          <ds:X509Certificate>...</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </md:KeyDescriptor>

    <md:AssertionConsumerService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
      Location="https://localhost:8181/" index="1"/>
  </md:SPSSODescriptor>
</md:EntityDescriptor>
```


Obrázek 4.3: Testovací aplikace: nepřihlášený uživatel

## 4.3 Ukázka funkčnosti

Na základě provedené konfigurace bude ukázán jednoduchý případ užití. Jeho průběh odpovídá Web Browser SSO Profile 1.1.3 s komunikací zahájenou SP (aplikací se SAM) a skládá se z několika kroků.

1. Běžící aplikace s nepřihlášeným uživatelem. Obrázek 4.3.

2. Přístup na *Show days*, jakožto k chráněnému zdroji dle kódů 4.2 a 4.3, iniciuje vytvoření zakódované SAML Request (viz kapitola 3.2.1 a kód 4.15) a její odeslání spolu s *RelayState* parametrem na IdP za účelem zadání přihlašovacích údajů (viz obrázek 4.4).

3. Úspěšné přihlášení uživatele *user1* znamená vytvoření SAML Response, která je nejprve podepsána privátním klíčem IdP, poté zašifrována veřejným klíčem z X.509 certifikátu poskytnutého aplikací (úryvek podepsané, zašifrované zprávy viz 4.16) a nakonec zakódována. Zpráva mimo jiné obsahuje LDAP atributy *uid* a *employeeType* na základě konfigurace pro-

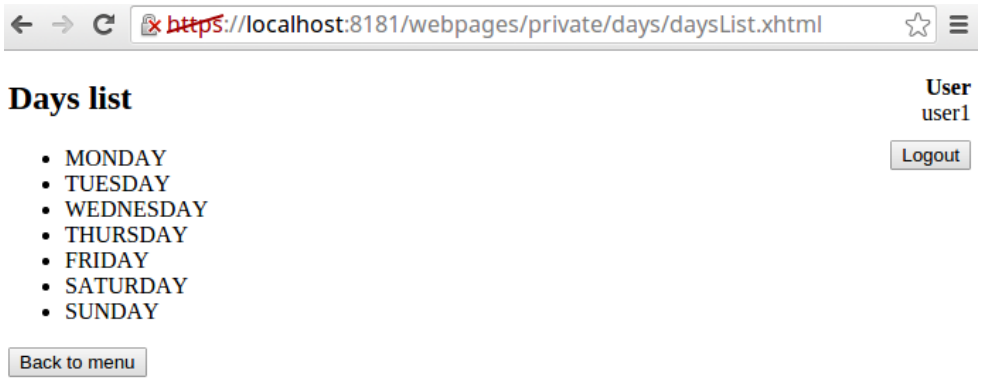
Kód 4.15: Dekódovaná SAML Request

```
<saml2p:AuthnRequest
  AssertionConsumerServiceURL="https://localhost:8181/"
  ID="8b78ee93bc919089766ecedeab2268dcd4f75f8d"
  IssueInstant="2016-04-09T15:26:15.784Z"
  Version="2.0"
  xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" >
  <saml2:Issuer xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
      https://users-test-app.com/shibboleth</saml2:Issuer>
  <saml2p:NameIDPolicy AllowCreate="1" />
</saml2p:AuthnRequest>
```

Kód 4.16: Zjednodušená dekódovaná SAML Response

```
<saml2p:Response
  Destination="https://localhost:8181/"
  ID="_cba60027ab32f3aef67d7af0500d5c74"
  InResponseTo="8b78ee93bc919089766ecedeab2268dcd4f75f8d"
  IssueInstant="2016-04-09T15:26:29.610Z"
  Version="2.0"
  xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" >
  <saml2:Issuer
    Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity"
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
    https://localhost:8443/idp/shibboleth</saml2:Issuer>
  <saml2p:Status>
    <saml2p:StatusCode
      Value="urn:oasis:names:tc:SAML:2.0:status:Success" />
  </saml2p:Status>
  <saml2:EncryptedAssertion
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
    <xenc:EncryptedData ...>
      ...
      <ds:X509Certificate>...</ds:X509Certificate>
      ...
    </xenc:EncryptedData>
  </saml2:EncryptedAssertion>
</saml2p:Response>
```


Obrázek 4.4: Testovací aplikace: přihlášení na IdP




Obrázek 4.5: Testovací aplikace: *user1*, přístup povolen

vedené v sekci 4.2.2. Následuje odeslání POST požadavku na odpovídající adresu ACS definovanou v metadatech aplikace (viz 4.14).

4. Přijetí SAML Response modulem, který dekódovanou zprávu nejprve dešifruje privátním klíčem aplikace a poté ověří podpis veřejným klíčem IdP získaným z konfigurace SAM. Po extrakci atributů ze zprávy dojde k přihlášení pomocí *CallbackHandler* a přesměrování na cílovou URL získanou z *RelayState* parametru (více viz kapitola 3.2.2).

5. Přihlášený uživatel *user1* může přistoupit k seznamu dnů (viz obrázek 4.5), naopak k seznamu měsíců přístup nemá (viz obrázek 4.6). Na základě konfigurace zabezpečení zdrojů v aplikaci má druhý uživatel *admin1* přístup i k seznamu měsíců a potvrzuje to obrázek 4.1.

## 4.4 Zhodnocení

Navržené řešení se podařilo úspěšně realizovat, otestovat a získat tak alternativu k současným řešením 1.3. Výhodou je použití pouze nástrojů


Obrázek 4.6: Testovací aplikace: *user1*, přístup zakázán

dostupných v Java EE 7 specifikaci a aplikačním serveru GlassFish 4, odpadá tedy potřeba dalšího software. Na druhou stranu lze najít několik nedostatků implementace:

- SAM vyžaduje Full Platform implementaci Java EE specifikace. Web Profile servery neobsahují potřebnou JASPIC technologii. Nativní řešení či Spring Security SAML toto omezení nemají.
- Omezená funkčnost je zřejmou nevýhodou současné implementace. Řešení podporuje pouze jeden scénář komunikace v rámci Web Browser SSO Profile 1.1.3. Množina podporovaných SAML protokolů je také menší, zahrnuje jen Authentication Request Protocol.
- Složitá konfigurace. Obě současná řešení mají předpřipravený konfigurační *xml* soubor, do kterého se pouze doplňují hodnoty požadovaných, případně dalších volitelných atributů. Představené řešení sice umožňuje použití GlassFish konzole nebo příkazové řádky, nicméně pro očekávanou funkčnost je nezbytné poskytnout klíče vzniklé konfigurační mapy přímo modulu, čili upravit jeho zdrojový kód a znovu ho zkompilovat.
- Úprava zdrojového kódu je nutná i v rámci současné funkčnosti modulu, například v případě vypnutí šifrování SAML Response zpráv přijímaných od IdP. Bez ohledu na bezpečnost tohoto nastavení by taková změna vyžadovala zásah do kódu, konkrétně vynechání části provádějící dešifrování při zpracovávání přijaté SAML Response.

# Závěr

## Shrnutí

Hlavním cílem bakalářské práce bylo na základě nabytých poznatků navrhnout a implementovat způsob Shibboleth autentizace pro Java EE aplikace, který by využíval pouze standardních technologií jazyka Java EE verze 7 a možností aplikačního serveru GlassFish verze 4.

V práci byl proveden podrobný popis technologie Shibboleth spolu s rozborem jazyka SAML, jakožto používaným komunikačním prostředkem. Rozebrány byly rovněž autentizační možnosti vybraného aplikačního serveru GlassFish. Rešeršní část práce završil přehled současných řešení Shibboleth autentizace Java EE aplikací. Nebylo nalezeno žádné vyhovující řešení.

Kapitola zabývající se návrhem se mírně inspirovala Spring Security SAML. Výstupem kapitoly bylo rozhodnutí o využití technologie JASPIC dostupné v Java EE 7 specifikaci, ovšem pouze ve Full Platform implementaci, jíž však zadáním specifikovaný aplikační server je. Konkrétní návrh spočíval v realizování vlastního SAM spolu s využitím knihovny OpenSAML pro práci se SAML zprávami.

Implementace přinesla autorovi detailní seznámení se s bezpečnostními principy aplikačního serveru GlassFish a bezpečnostními technologiemi platformy Java EE.

Testovací scénář byl inspirován použitím technologie Shibboleth na FIT ČVUT. Naimplementované řešení testovací scénář splnilo. Tím byly úspěšně splněny všechny body zadání bakalářské práce.

## Možnosti rozšíření

Do budoucna by bylo vhodné rozšíření funkčnosti, ať již množství podporovaných SAML profilů nebo protokolů. Jako další vylepšení se nabízí nový způsob konfigurace, který by nevyžadoval zásah do zdrojového kódu modulu.

Tomu by musel odpovídat i robustnější návrh práce se SAML zprávami. Implementace nevyžadující žádné další úpravy, kromě prvotního nastavení SAM, by se pak stala důstojnou alternativou k současným řešením.

# Literatura


[1] Shibboleth. [online], [cit. 2016-04-02]. Dostupné z: https://shibboleth.net/

[2] ICT FIT ČVUT: Shibboleth SSO. [online], 2016, [cit. 2016-03-18]. Dostupné z: https://ict.fit.cvut.cz/~web/current/web/identity/shibboleth-sp/

[3] Java Enterprise Edition. [online], [cit. 2016-04-02]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/overview/index.html

[4] GlassFish. [online], [cit. 2016-04-02]. Dostupné z: https://glassfish.java.net/

[5] Internet2: Shibboleth. [online], [cit. 2016-03-18]. Dostupné z: http://www.internet2.edu/products-services/trust-identity/shibboleth/#service-faq

[6] Shibboleth Consortium: Members. [online], [cit. 2016-03-19]. Dostupné z: http://shibboleth.net/consortium/

[7] Shibboleth Consortium: Identity Provider. [online], [cit. 2016-03-19]. Dostupné z: http://shibboleth.net/products/identity-provider.html

[8] Shibboleth Consortium: Service Provider. [online], [cit. 2016-03-19]. Dostupné z: http://shibboleth.net/products/service-provider.html

[9] OASIS®: *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. March 2008, s. 11 [cit. 2016-03-19]. Dostupné z: https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf

[10] XML.org, S.: History of SAML. [online], [cit. 2016-03-19]. Dostupné z: http://saml.xml.org/history

[11] OASIS®: OASIS Security Services (SAML) TC. [online], [cit. 2016-03-19]. Dostupné z: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[12] Norris, W.: Achieving Single Sign-on Google Apps and Shibboleth 2.0. [online], January 2008, University of Southern California, [cit. 2016-03-19]. Dostupné z: https://shibboleth.usc.edu/docs/google-apps/

[13] OASIS®: *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. March 2008, s. 16-19, 25 [cit. 2016-03-19]. Dostupné z: https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf

[14] Gajasinghe, K.: Introduction to Security Assertion Markup Language 2.0. [online], February 2014, WSO2, [cit. 2016-03-19]. Dostupné z: http://wso2.com/library/articles/2014/02/introduction-to-security-assertion-markup-language-2.0/

[15] OASIS®: *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. March 2008, s. 26-30 [cit. 2016-03-19]. Dostupné z: https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf

[16] OASIS®: *Bindings for the OASOS Security Assertion Markup Language (SAML) V2.0 Errata Composite*. September 2015, s. 16-18 [cit. 2016-04-17]. Dostupné z: https://www.oasis-open.org/committees/download.php/56779/sstc-saml-bindings-errata-2.0-wd-06.pdf

[17] Aboulnaga, A.: An overview of GlassFish Server. [online], [cit. 2016-03-20]. Dostupné z: http://www.otechmag.com/magazine/2015/fall/achmed-aboulnaga.html

[18] Oracle Corporation and/or its affiliates: GlassFish Server. [online], 2015, [cit. 2016-03-27]. Dostupné z: https://glassfish.java.net/

[19] Oracle and/or its affiliates: *The GlassFish Server Open Source Edition Security Guide*. May 2013, s. 15-19 [cit. 2016-03-22]. Dostupné z: https://glassfish.java.net/docs/4.0/security-guide.pdf

[20] Oracle and/or its affiliates: *The GlassFish Server Open Source Edition Security Guide*. May 2013, s. 47-50 [cit. 2016-03-22]. Dostupné z: https://glassfish.java.net/docs/4.0/security-guide.pdf

[21] Oracle Corporation and/or its affiliates: JSR 196: JavaTM Authentication Service Provider Interface for Containers. [online], [cit. 2016-03-24]. Dostupné z: https://www.jcp.org/en/jsr/detail?id=196

[22] Oracle Corporation and/or its affiliates: Java EE 7 Technologies. [online], [cit. 2016-03-25]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/tech/index.html

[23] Overton, A.: Using JASPIC to Secure a Web Application in GlassFish. [online], [cit. 2016-03-25]. Dostupné z: https://dzone.com/articles/using-jaspic-secure-web

[24] Oracle Corporation and/or its affiliates: The glassfish-web.xml File. [online], [cit. 2016-03-25]. Dostupné z: https://docs.oracle.com/cd/E18930_01/html/821-2417/beaql.html

[25] Klingenstein, N.: NativeSPJavaInstall. [online], [cit. 2016-03-25]. Dostupné z: https://wiki.shibboleth.net/confluence/pages/viewpage.action?pageId=4359770

[26] Sprin Security SAML. [online], [cit. 2016-04-02]. Dostupné z: http://projects.spring.io/spring-security-saml/

[27] The Apache Software Foundation: mod_proxy_ajp. [online], [cit. 2016-03-25]. Dostupné z: https://httpd.apache.org/docs/2.4/mod/mod_proxy_ajp.html

[28] SWITCH: Shibboleth Service Provider Deployment. [online], [cit. 2016-03-25]. Dostupné z: https://www.switch.ch/aai/guides/sp/

[29] Cantor, S.: NativeSPArchitecture. [online], [cit. 2016-03-25]. Dostupné z: https://wiki.shibboleth.net/confluence/pages/viewpage.action?pageId=4358191

[30] Schäfer, V.: Spring Security SAML Extension. [online], [cit. 2016-03-25]. Dostupné z: http://docs.spring.io/spring-security-saml/docs/current/reference/html/chapter-introduction.html#section-when-to-use

[31] Schäfer, V.: Spring Security SAML Extension. [online], [cit. 2016-03-25]. Dostupné z: http://docs.spring.io/spring-security-saml/docs/current/reference/html/chapter-quick-start.html#quick-start-sp-metadata

[32] Data Geekery GmbH: Java EE or Spring? Neither! We Call Out For a Fresh Competitor. [online], [cit. 2016-03-25]. Dostupné z: http://blog.jooq.org/2015/06/10/javaee-or-spring-neither-we-call-out-for-a-fresh-competitor/

[33] Kumar, P.: Java Servlet Filter Example Tutorial. [online], August 2013, [cit. 2016-03-26]. Dostupné z: http://www.journaldev.com/1933/java-servlet-filter-example-tutorial

[34] Oracle Corporation and/or its affiliates: Using Programmatic Security with Web Applications. [online], [cit. 2016-03-26]. Dostupné z: https://docs.oracle.com/javaee/6/tutorial/doc/gjiie.html

[35] Oracle Corporation and/or its affiliates: ServerAuth (Java EE 7). [online], [cit. 2016-03-27]. Dostupné z: https://docs.oracle.com/javaee/7/api/javax/security/auth/message/ServerAuth.html

[36] Oracle Corporation and/or its affiliates: ServerAuthModule (Java EE 7). [online], [cit. 2016-03-27]. Dostupné z: https://docs.oracle.com/javaee/7/api/javax/security/auth/message/module/ServerAuthModule.html

[37] Oracle Corporation and/or its affiliates: CallerPrincipalCallback (Java EE 7). [online], [cit. 2016-03-27]. Dostupné z: http://docs.oracle.com/javaee/7/api/javax/security/auth/message/callback/CallerPrincipalCallback.html

[38] OpenSAML. [online], [cit. 2016-04-02]. Dostupné z: https://wiki.shibboleth.net/confluence/display/OpenSAML/Home

[39] Rasmusson, S.: SAML Security: OpenSAML. [online], [cit. 2016-03-27]. Dostupné z: http://blog.samlsecurity.com/p/opensaml.html

[40] OneLogin's SAML Java SAML. [online], [cit. 2016-03-28]. Dostupné z: https://github.com/onelogin/java-saml

[41] OneLogin, Inc.: Code Your Java App to Provide SSO via OneLogin. [online], [cit. 2016-03-28]. Dostupné z: https://developers.onelogin.com/saml/java

[42] LastPass: LastPass. [online], [cit. 2016-03-28]. Dostupné z: https://lastpass.com/

[43] LastPass SAML SDK for Java. [online], [cit. 2016-03-28]. Dostupné z: https://github.com/lastpass/saml-sdk-java

[44] The Apache Software Foundation: Welcome to Apache Maven. [online], [cit. 2016-04-03]. Dostupné z: https://maven.apache.org/index.html

[45] The Apache Software Foundation: *Apache Maven Assembly Plugin - Introduction*. October 2015, [cit. 2016-04-03]. Dostupné z: https://maven.apache.org/plugins/maven-assembly-plugin/

[46] The Apache Software Foundation: *Apache Maven Assembly Plugin - Predefined Assembly Descriptors*. October 2015, [cit. 2016-04-03]. Dostupné z: https://maven.apache.org/plugins/maven-assembly-plugin/descriptor-refs.html#jar-with-dependencies

[47] Oracle Corporation: Adding Authentication Mechanism to the Servlet Container. [online], 2010, [cit. 2016-04-03]. Dostupné z: http://docs.oracle.com/cd/E19798-01/821-1752/gizel/index.html

[48] Oracle Corporation and/or its affiliates: ServerAuthModule (Java EE 7 Specification APIs). [online], [cit. 2016-04-08]. Dostupné z: https://docs.oracle.com/javaee/7/api/javax/security/auth/message/module/ServerAuthModule.html

[49] Oracle Corporation and/or its affiliates: ServerAuth (Java EE 7 Specification APIs). [online], [cit. 2016-04-08]. Dostupné z: https://docs.oracle.com/javaee/7/api/javax/security/auth/message/ServerAuth.html

[50] OASIS®: *Profiles for the OASOS Security Assertion Markup Language (SAML) V2.0 Errata Composite*. September 2015, s. 18 [cit. 2016-04-17]. Dostupné z: https://www.oasis-open.org/committees/download.php/56782/sstc-saml-profiles-errata-2.0-wd-07.pdf

[51] OASIS®: *Assertions and Protocols for the OASOS Security Assertion Markup Language (SAML) V2.0 Errata Composite*. September 2015, s. 10-11 [cit. 2016-04-17]. Dostupné z: https://www.oasis-open.org/committees/download.php/56776/sstc-saml-core-errata-2.0-wd-07.pdf

[52] Coffey, N.: XORShift random number generators in Java. [online], 2013, [cit. 2016-04-17]. Dostupné z: http://www.javamex.com/tutorials/random_numbers/xorshift.shtml

[53] prof. Ing. Róbert Lórencz, CSc.: BI-BEZ - Přednáška č.5 - Hašovací funkce, SHA-x. 2013, [cit. 2016-04-17].

[54] prof. Ing. Róbert Lórencz, CSc.: BI-BEZ - Přednáška č.4 - AES, Operační módy. 2013, [cit. 2016-04-17].

[55] The Apache Software Foundation: Welcome to ApacheDS. [online], [cit. 2016-04-08]. Dostupné z: https://directory.apache.org/apacheds/

[56] The Eclipse Foundation: Jetty - Servlet Engine and Http Server. [online], 2016, [cit. 2016-04-08]. Dostupné z: http://www.eclipse.org/jetty/

[57] The Apache Software Foundation: Apache Tomcat - Welcome! [online], [cit. 2016-04-08]. Dostupné z: http://tomcat.apache.org/

[58] Cantor, S.: IdPInstall - Shibboleth 2 - Confluence. [online], [cit. 2016-04-08]. Dostupné z: https://wiki.shibboleth.net/confluence/display/SHIB2/IdPInstall

[59] Oracle Corporation and/or its affiliates: JAAS Authentication Tutorial. [online], [cit. 2016-04-09]. Dostupné z: http://docs.oracle.com/javase/1.5.0/docs/guide/security/jaas/tutorials/GeneralAcnOnly.html#ClientLC

[60] Nishimura, T.: IdPAuthUserPass. [online], [cit. 2016-04-09]. Dostupné z: https://wiki.shibboleth.net/confluence/display/SHIB2/IdPAuthUserPass

[61] Oracle Corporation and/or its affiliates: LoginContext (Java 2 Platform SE 5.0). [online], [cit. 2016-04-10]. Dostupné z: http://docs.oracle.com/javase/1.5.0/docs/api/javax/security/auth/login/LoginContext.html

[62] Oracle Corporation and/or its affiliates: Subject (Java 2 Platform SE 5.0). [online], [cit. 2016-04-10]. Dostupné z: http://docs.oracle.com/javase/1.5.0/docs/api/javax/security/auth/Subject.html

[63] Oracle Corporation and/or its affiliates: Principal (Java 2 Platform SE 5.0). [online], [cit. 2016-04-10]. Dostupné z: http://docs.oracle.com/javase/1.5.0/docs/api/java/security/Principal.html

[64] Berkeley Lab Commons: Attribute Definitions - Identity Management. [online], [cit. 2016-04-09]. Dostupné z: https://commons.lbl.gov/display/IDMgmt/Attribute+Definitions#AttributeDefinitions-uiduid

[65] Oracle Corporation and/or its affiliates: JAAS Reference Guide. [online], [cit. 2016-04-10]. Dostupné z: http://docs.oracle.com/javase/1.5.0/docs/guide/security/jaas/JAASRefGuide.html

[66] Bantz, D.: ResolverLDAPDataConnector. [online], [cit. 2016-04-09]. Dostupné z: https://wiki.shibboleth.net/confluence/display/SHIB2/ResolverLDAPDataConnector

[67] Bieber, B. T.: IdPAddAttribute - Shibboleth 2. [online], [cit. 2016-04-09]. Dostupné z: https://wiki.shibboleth.net/confluence/display/SHIB2/IdPAddAttribute

[68] OASIS®: *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. September 2015, [cit. 2016-04-09]. Dostupné z: https://www.oasis-open.org/committees/download.php/56785/sstc-saml-metadata-errata-2.0-wd-05.pdf

[69] InC-Federation - Internet2: X.509 Certificates in Metadata. [online], [cit. 2016-04-10]. Dostupné z: https://spaces.internet2.edu/display/InCFederation/X.509+Certificates+in+Metadata

Příloha **A**

# Seznam použitých zkratek

**ACS** Assertion Consumer Service

**ADFS** Active Directory Federation Services

**AES** Advanced Encryption Standard

**API** Application Programming Interface

**APJ13** Apache JServ Protocol version 1.3

**ČVUT** České vysoké učení technické v Praze

**EJB** Enterprise JavaBeans

**FIT** Fakulta informačních technologií

**HTTP** Hypertext Transfer Protocol

**HTTPS** Hypertext Transfer Protocol Secure

**IdP** Identity Provider

**IIS** Internet Information Server

**J2EE** Java 2 Enterprise Edition

**JAAS** Java Authentication and Authorization Service

**Java EE** Java Enterprise Edition

**JASPIC** Java Authentication Service Provider Interface for Containers

**JSR** Java Specification Request

**JSF** JavaServer Faces

**LDAP** Lightweight Directory Access Protocol

**LDAPS** LDAP over SSL

**MDSSO** Multi-Domain SSO

**NSAPI** Netscape Server Application Programming Interface

**PAM** Pluggable Authentication Module

**SAM** Server Authentication Module

**SAML** Security Assertion Markup Language

**SDK** Software Development Kit

**SHA-1** Secure Hash Algorithm 1

**SOAP** Simple Object Access Protocol

**SP** Service Provider

**SPI** Service Provider Interface

**SSO** Single Sign-On

**SSTC** Security Services Technical Commitee

**TCP** Transmission Control Protocol

**TLS** Transport Layer Security

**URI** Uniform Resource Identifier

**URL** Uniform Resource Locator

**XML** Extensible Markup Language

Příloha **B**

# Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu disku
- `custom-shibboleth-sam-1.0-SNAPSHOT-jar-with-dependencies.jar` implementovaný SAM
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `src` ..... zdrojové kódy SAM
    - `pom.xml` ..... Maven Project Object Model
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Maleček_Kamil_2016.pdf` ..... text práce ve formátu PDF