Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj webové aplikace s využitím Web Forms

David MÜLLER

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

David Müller

Informatika

Název práce

Vývoj webové aplikace s využitím WebForms

Název anglicky

Using WebForm for web application development

Cíle práce

Cílem práce je vytvořit základní jádro webové aplikace využívající technologii WebForms. Dílčím cílem bude popsat možnosti této technologie v obecném kontextu tvorby webových aplikací a návazností na související části .NET frameworku.

Metodika

Metodiky práce je založena na analyticko-syntetickém přístupu. Bude provedeno studium a analýza odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude popsána technologie WebForms, možnosti jejího využití při tvorbě webových aplikací a její návaznosti na souvistející části .NET frameworku.

Dále bude provedena analýza a návrh jádra webové aplikace, které bude poté implementováno s využitím WebForms. Bude provedeno zkušební nasazení a otestování vytvořené aplikace. Ta bude dále zhodnocena a budou navrženy možnosti jejího dalšího rozšíření.

Doporučený rozsah práce

35-40 stran

Klíčová slova

.NET, C#, JavaScript, WebForm, SQL, webová aplikace

Doporučené zdroje informací

DUCKETT, Jon. HTML and CSS: Design and Build Websites. Wiley & Sons, 2011. ISBN 858-0-00104-171-1.

DUCKETT, Jon. Javascript & jQuery: Interactive Front-End Web Development. Wiley & Sons, 2013. ISBN 978-1-11853-164-8.

CHAFFER, Jonathan a Karl SWEDBERG. Learning jQuery, 4th Edition. Packt Publishing, 2013. ISBN 978-1-78216-315-2.

LERMAN, Julia. Programming Entity Framework. O'Reilly Media, 2010. ISBN 978-0-596-80726-9.

MISTRY, Ross a Stacia MISNER. Introducing Microsoft SQL Server 2014 [online]. Microsoft Press, 2014 [cit. 2015-05-21]. ISBN 978-0-7356-8475-1. Dostupné z: http://aka.ms/684751pdf

WENZ, Christian, Jason N. GAYLORD, Pranav RASTOGI, Miranda TODD a HANSELMAN. Professional ASP.NET 4.5 in C# and VB. Wiley / Wrox, 2013. ISBN 978-1-118-31182-0.

Předběžný termín obhajoby

2015/16 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 11. 03. 2016

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj webové aplikace s využitím Web Forms" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 11. 3. 2016 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu bakalářské práce Ing. Jiřímu Brožkovi, Ph. D. za odborné rady při psaní bakalářské práce.

Vývoj webové aplikace s využitím Web Forms

Souhrn

Tato bakalářská práce se zabývá problematikou vývoje bezpečných webových aplikací za použití ASP.NET a především pak Web Forms. V teoretické části se práce nejprve věnuje fungování frameworku ASP.NET a jeho součástí, které se pojí k bezpečnosti. Dále se pak zabývá častými problémy, kterými trpí velká část webových aplikací, a také útoky, kterými jsou tyto aplikace napadány.

V praktická části je postupně navrženo a implementováno jádro obecného informačního systému, které je schopno bezpečné autentizace i autorizace uživatelského účtu. Celá aplikace je poté otestována, aby se tak prokázala její funkčnost.

Klíčová slova: ASP.NET, Web Forms, ASP.NET Identity, ZAP, SQL, C#, webová aplikace

Using Web Forms for web application development

Summary

This bachelor thesis is focused on issues of developing secure web applications using ASP.NET and primarily Web Forms. The first section of theoretical part of this thesis describes ASP.NET Framework and its components, that are related to web security. The second section revolves around frequent mistakes made by many web developers, and also around attacks that can be waged against such vulnerable applications.

In the practical part, a core of universal IS is being designed and then created. This kernel is capable of authentication and authorization of a local user account. In the end, the entire application is fully tested to prove its functionality.

Keywords: ASP.NET, Web Forms, ASP.NET Identity, ZAP, SQL, C#, web application

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Vývoj webových aplikací je rozsáhlý a komplikovaný proces, při kterém je často kladen velký důraz na formu a funkce. Toto jsou jasně viditelné, a tudíž i ospravedlnitelné investice jak pro koncového zákazníka, tak i pro vyšší vedení softwarových firem. Problém nastává u těch součásti aplikace, které nejsou takto navenek prezentovatelné a jejich přínos není na první pohled patrný. Toto se týká především bezpečnostního aspektu aplikací.

Tato práce si klade za cíl analyzovat rizika spojená s bezpečnostními vadami webových aplikací a nabídnout možnosti, jak tato rizika minimalizovat.

Teoretická část sumarizuje frekventované útoky, a také zranitelnosti, jimiž trpí celá řada webových aplikací po celém světě. Popisuje, jak tyto útoky i zranitelnosti fungují a vznikají, a jak jim lze předejít.

Praktická část se věnuje vývoji jádra informačního systému, přičemž klade důraz na bezpečností stránku věci. Jádro bude do aplikace implementovat autentizační a autorizační modul, a bude odolné vůči útokým uvedeným v teoretické části práce, což bude prokázáno celkovým otestováním aplikace.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem práce je vytvořit zabezpečené základní jádro webové aplikace ve frameworku Web Forms, které bude disponovat integrovaným systémem řízení a správy uživatelských účtů. Funkčnost a flexibilita tohoto systému bude názorně demonstrována.

Dílčím cílem pak je popis Web Forms v obecném kontextu tvorby webových aplikací a jeho návazností na související části .NET frameworku.

2.2 Metodika

Metodika práce je založena na studiu a analýze odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude popsána framework Web Forms, možnosti jeho využití při tvorbě webových aplikací a jeho návaznosti na souvistející části .NET frameworku. Dále bude proveden rozbor nejčastějších rizik a zranitelností webových aplikací.

Na základě takto získaných informací bude navrženo jádro webové aplikace, které bude poté implementováno s využitím WebForms. Bude provedeno zkušební nasazení a otestování vytvořené aplikace. Ta bude dále zhodnocena a budou navrženy možnosti jejího dalšího rozšíření.

3 Teoretická východiska

3.1 .NET a jeho součásti

3.1.1 ASP.NET

ASP.NET je framework společnosti Microsoft určený k vývoji dynamických webových aplikací s využitím komponent .NET a HTML, CSS a Javascriptu (Gaylord, 2013, s. 3).

Aplikace založené na ASP.NET se kompilují, což přináší zvýšení jejich výkonnosti, bezpečnosti či stability. Kompilovaný kód je pak spouštěn Common Language Runtime, tedy CLR (Common Language Runtime, b.r.), který je stejný u všech .NET programů bez ohledu na jazyk, ve kterém byly napsány. Díky tomu je tedy možné vyvíjet ASP.NET v jakémkoliv jazyce, který je kompatibilní s CLR (ASP.NET Overview, b.r.).

3.1.1.1 State management

O hostování webových aplikací se v ASP.NET obvykle stará webový server Internet Information Services (IIS) a je k němu přistupováno skrze protokol HTTP, který je bezstavový. Pokaždé, když se pošle požadavek na server, vytvoří se nová instance stránky, která se zašle zpět klientovi k zobrazení a to znamená, že se veškeré změny vytvořené na straně klienta ztratí (ASP.NET State Management Recommendations, b.r.). Aby Microsoft obešel tento tradiční problém webového vývoje, musel být vytvořen systém na správu stavu aplikace, který nabízí několik možností, jak uchovávat data a to nejen na úrovni jednotlivých stránek, ale i na úrovni celé aplikace (Gaylord, 2013, s. 791).

Uložení na klientovi

Uložení dat pomocí metod spadajících do této kategorie nespotřebovává paměť serveru. Tato data jsou ale součástí těla HTTP požadavků a posílají se od klienta na server a poté zase zpět. Čím více informací je tedy potřeba posílat klientovi, tím déle trvá, než se stránka vykreslí a zobrazí (Gaylord, 2013, s. 793). Při práci s daty na straně klienta je také nutné mít na paměti, že jejich účinná ochrana je velmi problematická a existuje nespočet možností, jak s nimi manipulovat.

Příkladem budiž nástroje typu Firefox Tamper Data, Firebug nebo Web Developer dostupné v prohlížeči Firefox, či samotný skriptovací jazyk Javascript a vývojářské konzole jednotlivých prohlížečů.

Je třeba zmínit, že všechny následující způsoby uložení dat nejsou perzistentní (kromě Cookies). Vydrží pouze dokud bude stránka, na které byly vygenerovány, v prohlížeči otevřená.

Tento seznam je vypracován primárně na základě zdrojů (ASP.NET State Management Overview, b.r.) a (ASP.NET State Management Recommendations, b.r.):

- View state
    - Aktuální stav stránky, jejích ovládacích prvků (Controls) a libovolných proměnných se převede do zakódovaného tvaru do jednoho nebo více skrytých polí (Hidden Fields). Během inicializace stránky se pak tyto hodnoty opět obnoví do původního stavu (Dorans, 2010, s. 87 - 88).
    - Protože se data ukládají přímo do stránky, ukládání velkých objemů dat může velmi výrazně zpomalit její běh.
    - View state lze vypnout na úrovni celé stránky.
- Control state
    - Jedná se o způsob uložení specifický pro ovládací prvky (Controls) stránky, který funguje podobně, jako View state a je určený především k uložení informací kritických pro fungování těchto prvků (Gaylord, 2013, s. 265).
    - Lze jej vypnout pouze na úrovni jednotlivých ovládacích prvků, u kterých si vývojář vysloveně nepřeje ukládat jejich stav, či se o to stará nestandardním způsobem.
- Skrytá pole (Hidden Fields)
    - Ovládací prvek, který se ve webovém prohlížeči vykresluje, jako neviditelný prvek. Svoje hlavní využití nalézá při práci s Javascriptem, kdy do něj může vývojář ukládat hodnoty, se kterými potřebuje následně pracovat v serverové části aplikace.
- Cookies
    - Typicky malé množství dat, které se ukládá v textovém souboru nebo v paměti prohlížeče, které se posílá společně s požadavky na webový server

k vyřízení. Cookies se obvykle používá, jako ukazatel stavu autentizace uživatele ve webové aplikaci.
    - Cookies lze nastavit datum vypršení a nebo je ponechat perzistentní.
- Query stringy
    - Jedná se o informace, které se přidávají na konec URL ve formě parametrů a je to nejjednodušší způsob, jak si mezi jednotlivými stránkami předat hodnotu. Příkladem mohou být různé vyhledávací parametry, na jejichž základě se filtruje obsah stránky.
- HTML5 Storage (Gaylord, 2013, s. 793)
    - Data jsou ukládána lokálně u klienta a jsou vázaná na doménu původu. Oproti cookies lze do uložiště ukládat větší objem dat, která zároveň nejsou nikdy automaticky odesílána na server.

Uložení na serveru

Díky možnosti uložení dat v paměti serveru se redukuje objem informací, které je nutno s každým požadavkem zasílat zpět ke klientovi a zároveň je těžší se k nim, z pohledu útočníka, dostat a zneužít je. Ukládání příliš velkého objemu dat v paměti serveru však může spotřebovat nemalé zdroje, což může být u větších aplikací kritické (Gaylord, 2013, s. 791 - 793).

Následující seznam je vypracován primárně na základě zdrojů (ASP.NET State Management Overview, b.r.) a (ASP.NET State Management Recommendations, b.r.):

- Application state
    - Jedná se o úložiště, které je globálně přístupné v rámci celé aplikace z kterékoliv stránky (Gaylord, 2013, s. 793). Proto se velmi dobře hodí k uložení dat, ke kterým je nutné přistupovat velmi často a globálně, nezávisle na jejich roli. Příkladem může být třeba kolekce práv či uživatelských skupin.
- Session state
    - Stejně, jako Application state, i Session state umožňuje pracovat s uloženými daty v rámci celé aplikace a na kterékoliv její stránce. Každá klientská relace prohlížeče má ale přístup pouze k vlastnímu Session statu, který je separován od ostatních.

- Data v paměti vydrží i restart webového serveru (IIS), protože jsou udržována v jiném procesu.
- V souboru web.config aplikace lze nastavit dobu vypršení, po jejímž uplynutí (pakliže s těmito daty nepracuje žádný proces) se data vymažou a uvolní paměť serveru.

3.1.1.2 Web Forms

Web Forms je jedním z frameworků na vývoj webových aplikací, které ASP.NET nabízí. Je založený na modelu uživatelem vyvolávaných událostech (event-driven model) a skládá se z jednotlivých stránek ke kterým uživatelé přistupují skrze svůj webový prohlížeč (Introduction to ASP.NET Web Forms, b.r.).

Ve chvíli, kdy uživatel vyvolá nějakou událost (například ve formě kliknutí na serverovou komponentu, jakou může být tlačítko), zašle se na server požadavek, kde se vyhodnotí a poté se na serveru opět vygeneruje celá HTML stránka, kterou následně prohlížeč vykreslí (Gaylord, 2013, s. 792).

Jednotlivé webové stránky se skládají z ovládacích prvků (Controls), které obvykle využívají několik programovatelných událostí, skrze něž je vývojář schopen řídit celou aplikaci. Web Forms nabízejí již v základu poměrně široké spektrum těchto komponent od primitivních, které připomínají běžné HTML elementy, až po ty komplexní, které v sobě kombinují několik menších prvků a jsou schopné se připojit k databázovému zdroji dat. Existuje také nespočet bezplatných i komerčních rozšíření, kterými lze obohatit paletu ovládacích prvků (Introduction to ASP.NET Web Forms, b.r.).

3.1.1.3 ASP.NET Identity

ASP.NET nabízí různé systémy na správu členství (membership management) (Gaylord, 2013, s. 702). Jedním z nich je i ASP.NET Identity, který byl uveden s příchodem Visual Studia 2013 a ASP.NET verze 4.5 (Cutting Edge : A First Look at ASP.NET Identity, 2015).

Tento systém byl vytvořen s cílem nahradit předchozí ASP.NET Membership a Simple Membership systémy, které trpí nedostatky, jakými jsou problematická nebo přímo nemožná úpravá databázového schématu, použití ne-relačních úložišť k perzistenci dat, chybějící podpora Open Web Interface for .NET (OWIN) či propojení s externími účty (například Google, Facebook, Twitter, apod.). (Introduction to ASP.NET Identity, 2013)

Identity je rozšířitelný systém, jehož hlavními přednostmi jsou (Introduction to ASP.NET Identity, 2013):

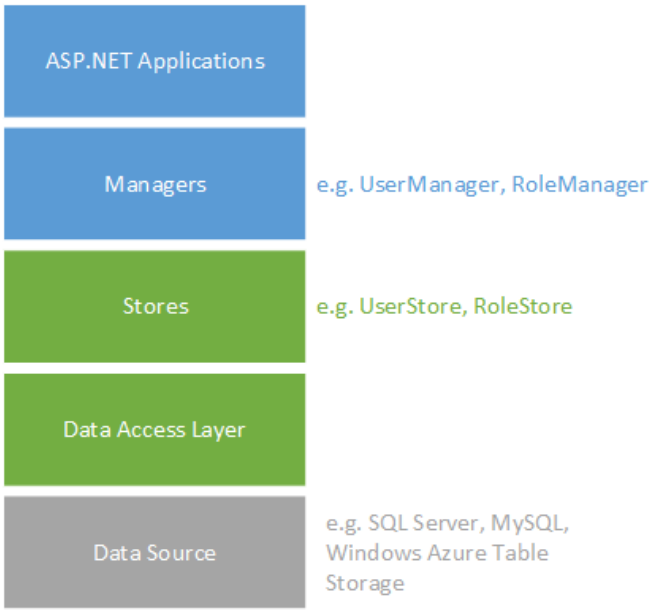
- Podpora OWIN
  - OWIN je zkratka pro Open Web Interface for .NET a definuje standardizované rozhraní mezi .NET webovými aplikacemi a webovými servery. Cílem je oddělit závislost ASP.NET aplikací na IIS a umožnit nasazení těchto aplikací i na jiných webových serverech, které OWIN implementují. (Open Web Interface for .NET, b.r.)
  - Identity už nevyužívá FormsAuthentication k vytváření autentizační cookie, ale spoléhá se na OWIN autentizaci při přihlašování a odhlašování uživatelů z aplikace.
- Podpora externích poskytovatelů autentizace
  - Zpřístupnění možnosti přihlásit se přes účet třetí strany, jako je Facebook, Google apod. a tudíž nenutit uživatele, aby si musel ve vaší aplikaci zakládat nový profil.
- Kontrola nad způsobem uložení dat
  - Identity primárně používá Entity Framework k realizaci perzistence dat. Na rozdíl od svých předchůdců ale umožňuje Entity Framework zcela odříznout a zavést jiný mechanismus perzistence. Počítá se dokonce i s databázemi nezaloženými na SQL.
  - Pokud by se vývojář rozhodl využít Entity Framework, má stále kontrolu nad schématem databáze a tudíž může snadno měnit například strukturu tabulek či datové typy jednotlivých sloupců.
- Využití jednotkových testů
  - Části aplikace, které využívají Identity, jsou jednotkově testovatelné.
- Claims-Based architektura
  - Identita uživatele je reprezentována řadou tvrzení (claims), které daného uživatele popisují daleko širším způsobem, než běžné role. Role je totiž schopná odpověď pouze na otázku typu ANO/NE, ale tvrzením může být cokoliv od jména, úrovně privilegií, členství ve skupinách, typu zařízení a další. (Claims-Based Identity for Windows, 2011)

- Podpora všech ASP.NET frameworků
    - Přestože byl Identity zamýšlen především pro ASP.NET MVC, podporuje i Web Forms, Web Pages a další.
- Distribuce pomocí NuGet balíčku
    - Identity se distribuuje jako sada samostatných stažitelných balíčků. Hlavním balíčkem je Microsoft.AspNet.Identity.Core, který obsahuje jádro celého systému a rozhraní potřebná pro jeho implementování do aplikace.

V tuto chvíli je Identity součástí mnoha šablon Visual Studia 2013 a společnost Microsoft jej doporučuje zavádět do nově vytvářených aplikací.

Architektura Identity

Identity se skládá ze dvou druhů tříd, které se nazývají sklady (stores) a správci (managers), přičemž sklady se starají o uložení dat, což znamená, že jsou svázány s mechanismy perzistence (Data Access Layer). Správci jsou třídy, které se využívají k provádění operací, jako je vytváření nebo mazání entit v rámci bezpečnostního systému. (Overview of Custom Storage Providers for ASP.NET Identity, 2014)



Obrázek 1: *Vrstvy Identity (Overview of Custom Storage Providers for ASP.NET Identity, 2014)*

Data, která Identity využívá, se rozdělují na 4 typy (Overview of Custom Storage Providers for ASP.NET Identity, 2014):

- Uživatelé (users):
  - Jedná se o reprezentaci uživatelů, kteří se registrovali přes interní registrační systém aplikace.
- Uživatelská tvrzení (user claims):
  - Řada tvrzení o konkrétním uživateli, která zpřesňují a rozšiřují informace o tom, kým uživatel skutečně je, což posléze usnadní autorizační procesy uvnitř aplikace.
- Uživatelská přihlášení (user logins):
  - Obsahuje informace o externím poskytovateli autentizace, jako je Facebook nebo Google, které jsou potřeba při přihlašování uživatele do aplikace.
- Role (roles):
  - Autorizační skupiny k nimž daný uživatel náleží.

3.2 OWASP

Open Web Application Security Project (OWASP) je otevřená mezinárodní komunita, která se zabývá bezpečností webových aplikací. Jejím hlavním cílem je poskytovat dokumentaci, rady, nástroje, doporučené postupy a další materiály komukoliv, kdo chce vyvíjet, provozovat a udržovat důvěryhodné aplikace. (OWASP *Top 10* - 2013, 2013)

3.2.1 Risk Rating

Jedná se o rozšiřitelný model analýzy rizika, který byl vytvořen za účelem co nejpřesnějšího stanovení rizik plynoucích z jednotlivých zranitelností zkoumané aplikace. Tato metodika se používá i k vyhodnocení vzorku dat, na němž se zakládá dokument Top 10.

Výstupem modelu je seznam jednotlivých slabin aplikace seřazený podle jejich závažnosti. Na základě tohoto seznamu je pak možné jednoznačně určit, které chyby opravovat a v jakém pořadí. Postup přípravy a řešení tohoto modelu lze rozdělit do několika kroků, které jsou blíže rozebrány v podkapitolách 3.2.1.1 – 3.2.1.4.

Kapitola 3.2.1 je vypracována na základě zdroje (OWASP *Risk Rating* Methodology, b.r.).

3.2.1.1 Identifikace rizika

Jako první probíhá fáze identifikace bezpečnostních trhlin. K tomu je potřeba mít informace o potenciálním útočníkovi či útočnících (kteří vůbec nemusejí být anonymními entitami na internetu, ale třeba interními či externími zaměstnanci firmy s různou motivací k útoku), typu útoku, který bude použit, samotné trhlině (z technického hlediska) a především možném dopadu na podnik. Při identifikaci rizik je dobré počítat vždy s nejhorším možným scénářem, který by nastal pokud by došlo k zneužití takové trhliny.

Faktory ovlivňující pravděpodobnost

Jakmile byly shromážděny všechny výchozí informace o zranitelnostech aplikace, je potřeba určit, jak závažné tyto zranitelnosti ve skutečnosti jsou. K tomu slouží 2 činitelé, kterými jsou **pravděpodobnost** a **dopad**. Při ohodnocování faktorů tvořících tyto činitele se používá celočíselná škála od 0 – 9, kdy 0 představuje nejpříznivější a 9 naopak nejnepříznivější možnost. K určení pravděpobnosti slouží řada faktorů, které se dělí do 2 následujících skupin:

Fatory útočníka

Cílem je určit pravděpodobnost úspěšného zneužití konkrétní bezpečnostní mezery útočníkem nebo skupinou útočníků.

- Úroveň schopností
  - Jaká je úroveň technických schopností, kterými útočníci oplývají?
  - 9 – Zkušenost s penetrací bezpečnostních opatření
  - 1 – Žádné technické schopnosti
- Motivace
  - Čeho útočník dosáhne, pokud zneužije tuto trhlinu?
  - 9 – Vysoká odměna
  - 1 – Mizivá odměna
- Příležitost
  - Jaké zdroje a příležitosti jsou nutné k tomu, aby se útok povedl?
  - 9 – Žádný přístup a levné zdroje
  - 1 – Úplný přístup a drahé zdroje

- Početnost
    - Jak početná musí být skupina útočníků?
    - 9 – Anonymní internetový uživatel
    - 1 – Skupina vedoucích pracovníků podniku

Faktory zranitelnost

Zde se hodnotí míra jednoduchosti odhalení a zneužití chyby z pohledu útočníka (nebo útočníků), který byl hodnocen v první skupině.

- Odhalitelnost
    - Jak jednoduché je pro útočníka odhalit tuto slabinu?
    - 9 – Pomocí automatických nástrojů
    - 1 – Prakticky nemožné
- Zneužitelnost
    - Jako jednoduché je pro útočníka samotné zneužití chyby?
    - 9 – Pomocí automatických nástrojů
    - 1 – Prakticky nemožné
- Povědomí
    - Jak dobře je tato slabina útočníkům známá?
    - 9 – Veřejně známá
    - 1 – Neznámá
- Detekce útoku
    - Jaká je šance detekování a odhalení útoku?
    - 9 – Velmi nízká, neukládají se žádné logy
    - 1 – Velmi vysoká, logy se ukládají a jsou pravidelně kontrolovány a prohlíženy

Faktory ovlivňující dopad

Existují dva druhy dopadu. Prvním je **technický dopad**, který se váže k samotné aplikaci, k datům, která používá a k funkcím, které poskytuje. Druhým druhem je **obchodní dopad**, který se týká samotného podniku, jež aplikaci provozuje.

Obecně platí, že obchodní dopad je důležitější než technický a ve výpočtu celkového dopadu by měl být upřednostněn, nicméně oba dopady se často prolínají, například poškození dat (která spadají do technického dopadu), může mít neblahý vliv na pověst

podniku (obchodní dopad). Proto je důležité získat co možná nejvíce detailů o technických dopadech, na jejichž základě bude pak možné stanovit obchodní dopady.

Faktory technického dopadu

Cílem je zjistit, jakého rozsahu by mohl být dopad na samotnou aplikaci v případě, že by byla tato slabina zneužita.

- Ztráta důvěryhodnosti
  - Jaké množství dat by mohlo být kompromitováno a jak citlivá jsou?
  - 9 – Veškerá data
  - 1 – Malé množství dat nízké významnosti
- Ztráta integrity
  - Jaké množství dat by se mohlo poškodit a jak ničivá by tato poškození byla?
  - 9 – Kompletní zničení nebo smazání dat
  - 1 – Minimální poškození malého množství dat
- Ztráta dostupnosti
  - Jaké služby či funkce by mohly být ztraceny?
  - 9 – Veškerá funkcionalita
  - 1 – Druhotné služby a funkce
- Ztráta zodpovědnosti
  - Jsou útočníkovy akce vystopovatelné až k jednotlivcům?
  - 9 – Úplná anonymita
  - 1 – Úplná dohledatelnost

Faktory obchodního dopadu

Obchodní dopad se základá na technickém dopadu a vyžaduje znalost podniku, který aplikaci provoze, a co je pro něj důležité. Obchodní dopady jsou hlavním důvodem pro investice do zabezpečení systému, protože přímo ovlivňují rentabilitu podniku.

- Finanční škoda
  - Jakou finanční škodu by tato slabina mohla způsobit?
  - 9 - Bankrot
  - 1 – Menší, než náklady na opravení chyby

- Poškození dobrého jména
    - Vedlo by zneužití chyby k poškození reputace podniku?
    - 9 – Zničení celé značky
    - 1 – Drobné poškození
- Neřešení problému
    - Kolik nechtěné pozornosti by nám přineslo ignorování tohoto problému?
    - 9 – Medializace
    - 1 – Drobná pozornost
- Narušení soukromí
    - Kolik informací o skutečných osobách by bylo kompromitováno?
    - 9 – Všichni uživatelé aplikace
    - 2 – Jeden člověk

3.2.1.2 Stanovení závažnosti rizika

V této fázi jsou připravené odhady pravděpodobnosti a dopadu, které spolu tvoří riziko. Konkrétní hodnoty pravděpodobnosti a dopadu se vypočítají, jako aritmetický průměr hodnot, kterými jsou ohodnoceny jednotlivé faktory, kde X je soubor jednotlivých ohodnocení faktorů a N je počet faktorů:

$$A = \frac{1}{n} \cdot \sum_{i=1}^{n} x^i$$

Pro lepší představu je možné si uvést příklad výpočtu celkové pravděpodobnosti a dopadu pro libovolnou slabinu. Volba faktorů a jejich ohodnocení vyplývá z předchozích kroků přípravy tohoto modelu, viz tabulka 1.

Faktory útočníka			
Úroveň schopností	Motivace	Příležitost	Početnost
3	5	4	9
Faktory zranitelnosti			
Odhalitelnost	Zneužitelnost	Povědomí	Detekce útoku
2	6	4	5
Celková pravděpodobnost = 4,75			

Tabulka 1: *Výpočet celkové pravděpodobnosti (vlastní zpracování)*

Faktory technického dopadu			
Ztrátra důvěryhodnosti	Ztráta integrity	Ztráta dostupnosti	Ztrátra zodpovědnosti
3	5	4	9
Celkový technický dopad = 5,25			
Faktory obchodního dopadu			
Finanční škoda	Poškození reputace	Neřešení problému	Narušení soukromí
2	4	4	1
Celkový obchodní dopad = 2,25			
Celkový dopad = 2,25			

Tabulka 2: *Výpočet celkového dopadu (vlastní zpracování)*

Hodnota závažnosti rizika (nebo zkráceněna pouze rizika) by se pak spočítala podle vzorce:

```
Riziko = Pravděpodobnost * Dopad
```

V tomto příkladě by měla počítaná zranitelnost riziko zneužití 10,6875, což je pouze ukazatel jejího pořadí v seznamu bezpečnostních slabin, které je na aplikaci potřeba ošetřit. Tento postup je nutné opakovat pro každou slabinu zvlášť.

3.2.1.3 Grafické znázornění rizika

Pro rychlejší orientaci lze zobrazit číselné hodnoty celkové pravděpodobnosti i dopadu do grafické podoby. Samotný převod se provede podle následující tabulky:

Úrovně pravděpodobnosti a dopadu	
0 – 2	Nízká
3 – 5	Střední
6 – 9	Vysoká

Tabulka 3: *Převod bodů na úrovně (OWASP Risk Rating Methodology, b.r.)*

Z příkladu tedy vyplývá, že pravděpodobnost zneužití fiktivní slabiny dosáhla střední (4,75) úrovně a celkový dopad by pak byl nízký (2,25). To dohromady dává riziko 10,6875, které odpovídá nízké úrovni:

Celková závažnost rizika				
Dopad	Vysoká	Střední	Vysoká	Kritická
	Střední	Nízká	Střední	Vysoká
	Nízká	Mizivá	Nízká	Střední
		Nízká	Střední	Vysoká
	Pravděpodobnost			

Tabulka 4: *Celková závažnost rizika (vlastní zpracování)*

3.2.1.4 Přízpůsobitelnost modelu

Smyslem OWASP Risk Rating modelu není poskytnout žebříček obecných zranitelností, podle kterého může jakýkoliv podnik upravit svoji aplikaci. Každý podnik musí provést svoji vlastní identifikaci rizik, podle které posléze sestaví takové sady faktorů určujících pravděpodobnost a dopad, které podle něj nejlépe reflektují problematiku, ve které pohybuje. Model lze rozšířit i o váhy jednotlivých faktorů podle priorit podniku. Vzorec pro výpočet celkové pravděpodobnosti a dopadu by se pak změnil z aritmetického průměru na vážený průměr podle vzorce, kde X je soubor jednotlivých ohodnocení faktorů a W jsou jejich váhy.

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i \cdot x_i}{\sum_{i=1}^{n} w_i}$$

3.2.2 Top 10

Jedním z hlavních projektů OWASP je Top 10, který se vydává ve formě dokumentu a představuje seznam deseti nejčastějších a nejkritičtějších bezpečnostích rizik webových aplikací (OWASP *Top 10* - 2013, 2013). Na tento dokument se odkazuje řada norem (např. Payment Card Industry Data Security Standard), knih (např. (Dorans, 2010, s. 10)) či organizací (např. IBM). Účelem projektu je upozornit na nejběžnější a zároveň nejnebezpečnější bezpečnostní rízika, kterými trpí řada aplikací po celém světě. Zároveň přináší i obecné rady, jak se těmto potenciálním rizikům vyhnout, případně jak je v aplikaci rozeznat a lokalizovat (OWASP *Top 10* - 2013, 2013).

Nejnovější verze dokumentu je z roku 2013 a byla vyhotovena na základě dat 7 firem specializujících se na zabezpečení aplikací, přičemž tato data dohromady přesahují 500 000 chyb tisíců aplikací. Jednotlivé položky Top 10 pak byly vybrány a uspořádány podle četnosti svého výskytu v kombinaci s jejich zneužitelností, detekovatelností a potenciálními dopady na zasaženou organizaci (OWASP Top 10 - *2013,* 2013, s. 3).

Riziko	Zneužitelnost	Rozšíření	Zjistitelnost	Dopad
1. Injektování	Vysoká	Střední	Střední	Vysoký
2. Autentizace	Střední	Vysoké	Střední	Vysoký
3. XSS	Střední	Kritické	Vysoká	Střední
4. Přímé odkazování	Vysoká	Střední	Vysoká	Střední
5. Konfigurace	Vysoká	Střední	Vysoká	Střední
6. Expozice citlivých dat	Nízká	Nízké	Střední	Vysoký
7. Řízení přístupu	Vysoká	Střední	Střední	Střední
8. CSRF	Střední	Střední	Vysoká	Střední
9. Komponenty	Střední	Vysoké	Nízká	Střední
10. Přesměrování	Střední	Nízké	Vysoká	Střední

Tabulka 5: *OWASP Top 10 (OWASP Top 10 - 2013, 2013, s. 21)*

Celý vzorek dat byl vyhodnocen pomocí modelu Risk Rating, kterým se zabývala kapitola 3.2.1. Pro potřeby obecného žebříčku Top 10 byly zvoleny 3 faktory pravděpodobnosti (rozšíření, detekovatelnost a snadnost) zneužití a 1 faktor dopadu (technický dopad). Pravděpodobnostní faktory byly vypočítány na základě poskytnutých dat. Autoři Top 10 poté odhadli průměrný technický dopad jednotlivých položek (OWASP Top 10 - 2013, 2013, s. 5). Výsledkem je žebříček zobrazený v tabulce č. 5.

3.2.2.1 Injektování

Útok injektováním funguje tak, že se přes neošetřený vstup podsune kód, který se složí s kódem původním a pozmění jeho zamýšlenou funkci (Dorans, 2010, s. 185). Nacházet se může všude tam, kde se očekává vstup od uživatele. Konkrétních typů injektování je celá řada a mezi nejběžnější patří injektování SQL, XML či LDAP. Zneužitelnost této chyby je přitom velmi jednoduchá, neboť útočník zneužívá syntaxi cílového interpretu a obvykle posílá jednoduché textové příkazy (viz příklad útoku) (OWASP Top 10 - 2013, 2013, s. 7).

Jak zranitelnost odhalit?

Problematická jsou místa, která vyžadují uživatelský vstup a následně jej nevalidují, či z uživatelského vstupu skládají dotazy (třeba SQL) (Howard, 2009). Odhalit tyto zranitelnosti v aplikaci vyžaduje kombinaci skenování penetračními nástroji a především analýzu zdrojového kódu, na čemž by se měli podílet i vývojáři dobře znalí zkoumané aplikace, kteří by měli být schopni poukázat na problematická místa. (OWASP Top 10 - 2013, 2013, s. 7)

Jak zranitelnost opravit?

V místech, kde je vyžadován uživatelský vstup používat pouze API, které podporuje parametrizované volání požadovaného typu. Další možností je tzv. escapování, které vlastně spočívá v kódování vstupů tak, aby byly bezpečné pro cílové intreprety. Třetí možností je použití tzv. whitelistů, což jsou vlastně seznamy povolených speciálních znaků pro daný vstup. Jejich zřejmou nevýhodou ale je, že občas je nutné v aplikaci použít znak, který se na whitelistu nenachází. (Gaylord, 2013), (OWASP Top 10 - 2013, 2013, s. 7)

Příklad útoku:

Na přihlašovací stránce aplikace jsou 2 textová pole pro zadání uživatelského jména a heslo. Zranitelné volání do databáze pak provádí následující SQL dotaz:

```
string sSql = "SELECT * FROM users WHERE username = '" + sUsername + "'
AND password = '" + sPassword + "';";
```

Útočník pak může textová pole vyplnit následujícím způsobem:

```
Uživatelské jméno: cokoliv
Heslo: cokoliv'; DROP TABLE users; --
```

Výsledný SQL dotaz by pak smazal celou tabulku users.

3.2.2.2 Chybná autentizace a správa relací

Autentizace je proces ověřování uživatelské identity. Pokud je uživatel takto ověřen, může systém pokračovat autorizací jeho akcí, což je proces povolování přístupu k různým součástem aplikace. (Dorans, 2010, s. 152) Při nesprávné implementaci může vést k tomu, že si útočník přisvojí některé (nebo všechny) uživatelské účty a díky tomu může v aplikaci provádět vše, k čemu jsou tyto účty oprávněny. Obvyklým cílem jsou účty s rozšířenými pravomocemi – například správcovské nebo manažerské. (OWASP Top 10 - 2013, 2013, s. 6)

Jak zranitelnost odhalit?

Stejně jako v případě injektování je nutné prozkoumat zdrojový kód a zaměřit se na autentizační a autorizační mechanismy aplikace. Častým poznávacím znamením zranitelných autentizačních systému je udržování klíčových informací v nezabezpečené podobě. Třeba v URL webových stránek ve formě parametrů (Howard, 2009, s. 76), v těle HTTP požadavků (Howard, 2009, s. 77) či v cookies (Howard, 2009, s. 78).

Problematické jsou i aplikace, jejichž relace jsou zranitelné útoky typu Session Fixation, které spočívají v získání platného ID relace, donucení skutečného uživatele se s tímto ID relace přihlásit a poté takto autentizovanou relaci ukrást skrze znalost jejího ID (OWASP Top 10 - 2013, 2013, s. 8).

Jak zranitelnost opravit?

Aplikaci je nutné ochránit šifrovaným spojením a autentizační data nevystavovat v prostém textu tam, kde by je mohl kdokoliv vidět a zneužít (Howard, 2009, s. 85 - 86). V ideálním případě je uchovávat pouze na serveru a ke klientovi přenášet informaci jen o tom, zda daný uživatel byl či nebyl autentizován (či autorizován). Pokud je data z nějakého důvodu nutné přenášet či uchovávat na straně klienta, je nutné je taktéž zabezpečit šifrováním a určitou dobou platnosti (OWASP Top 10 - 2013, 2013, s. 8).

Ošetření relace je pak možné například vytvářením nového ID při každém požadavku na server, vždy mít nastavenou určitou dobu vypršení ID relace a implementováním unikátních tokenů (viz Kapitola 3.1.2.8) (OWASP Top 10 - 2013, 2013, s. 8).

Příklad útoku:

Útočník zašle oběti odkaz:

```
http://nechejseokrast.cz/login/?sid= 64BDFS64318DSBGAS/
```

Oběť si tento odkaz zobrazí a přihlásí se do aplikace. Relace je v tuto chvíli autentizovaná a útočník se může v rámci aplikace vydávat za oběť.

Tento útok lze provést i za pomoci dalších zranitelností aplikace, jako je například XSS, kterým je útočník schopen krást ID relací, aniž by oběti musely otevírat nebezpečné odkazy.

3.2.2.3 Cross-Site Scripting (XSS)

Toto je nejrozšířenější bezpečnostní chyba webových aplikací (OWASP Top 10 - 2013, 2013, s. 9). Objevuje se u aplikací, které vkládají data, poskytnutá z nějakého vstupu (například uživatelem), do webové stránky, aniž by je dostatečně validovaly. Jakýkoliv vstup, který se vkládá na stránku, může vést k XSS zranitelnosti (Dorans, 2010, s. 45).

Útok spočívá v zasílání skriptů ve formě textových řetězců, které se vloží na stránku a prohlížeč je interpretuje jako aktivní obsah a dovolí jim vykonat se (OWASP Top 10 - 2013, 2013, s. 9), díky čemuž útočník může například krást ID relací nebo cookies, měnit vzhled stránky, přesměrovávat uživatele na jiné stránky a tak dále (Howard, 2009, s. 30). XSS se dělí na tři typy: stored, reflected a DOM based (Howard, 2009, s. 31).

Jak zranitelnost odhalit?

Podobně, jako v kapitole 3.2.2.1.

Jak zranitelnost opravit?

Pokud aplikace vyžaduje zobrazování uživatelem vkládaného obsahu, je nutné tento obsah, před vložením do webové stránky, nejprve escapovat (Howard, 2009, s. 47). Ideální variantou je použití whitelistů a nebo knihoven, které se zabývají ošetřováním vkládaného obsahu (OWASP Top 10 - 2013, 2013, s. 9).

Příklad útoku:

V aplikace se nachází stránka, kde je možné volně přidávat komentáře. Útočník do svého komentáře vloží následující skript:

```
<script src="http://nechejseokrast.cz/ukradnisessinu.js">
```

Tímto se každému, komu se načte stránka, která obsahuje útočníkův komentář, načte a spustí i javascriptový soubor, kterým útočník krade ID relace.

3.2.2.4 Nezabezpečené odkazy na objekt

Tato zranitelnost vzniká v aplikacích, které odkazují na objekty (soubory, záznamy, stránky, apod.) s omezeným přístupem pomocí jejich skutečných názvů či klíčů. Aplikace pak nedostatečně ověřuje opravnánění přístupu k takovým objektům (OWASP Top 10 - 2013, 2013, s. 6).

Jak zranitelnost odhalit?

Je potřeba analyzovat jednotlivé implementace odkazů na objekty a zjistit, zda ověřují přístupová oprávnění.

Jak zranitelnost opravit?

Ověřovat přístupová práva ke všem objektům, které by neměly být volně dostupné a k odkazování na objekty nepoužívat jejich skutečný název či jiný identifikátor, ale spíše generovaný řetězec znaků (OWASP Top 10 - 2013, 2013, s. 10). Dobrou metodou je přístupy k takovým objektům řídit přes centrálního správce, který vždy provede patřičná bezpečnostní opatření.

Příklad útoku:
Aplikace zobrazila webovou stránku, jejíž URL je:

```
http://nechejseokrast.cz/data/soubory?soubor=utocnik/
```

Útočník v této adrese změní hodnotu parametru *soubor* na jméno souboru, ke kterému nemá jinou cestou přístup a tento soubor se mu skutečně zobrazí.

3.2.2.5 Nebezpečná konfigurace

Tento problém může nastat na jakékoliv úrovni aplikace. Může se jednat o cokoliv od zbytečně povolených portů ve firewallu serveru, přes špatně nastavené frameworky, ve kterých je aplikace napsaná, až po zastaralý operační systém (OWASP Top 10 - 2013, 2013, s. 6).

Jak zranitelnost odhalit?
Odhalit tuto zranitelnost vyžaduje pozornost nejen vývojářů, ale i správců systému, protože aplikace nemusí být napadnuta pouze skrze chybu v jejím zdrojovém kódu, ale třeba skrze špatně zabezpečený databázový server.

Do této kategorie spadají i příliš slabá nebo běžně používaná hesla (například admin, 123456 apod.).

V hledání bezpečnostních mezer tohoto druhu je vhodné použít automatizované skenery, které jsou schopny odhalit továrně nastavené výchozí přístupové účty, nepotřebné služby a další (OWASP Top 10 - 2013, 2013, s. 11).

Jak zranitelnost opravit?
Udržovat software aktuální především co se týče chybových záplat a pravidelně provádět kontroly, například pomocí skenovacích nástrojů.

Příklad útoku:
V aplikačním serveru je stále aktivní nezměněný výchozí administrátorský účet. Útočník se přes tento účet může přihlásit a je tak schopen nad serverem převzít kontrolu.

3.2.2.6 Expozice citlivých dat

Potenciálnímu útoku na citlivá data svých uživatelů se vystavují ty aplikace, které tato data nedostatečně chrání. Konkrétně se jedná o chybějící, nebo velmi slabé šifrování dat v klidu i v pohybu. K takhle exponovaným datům se lze dostat třeba z prohlížeče uživatele, útokem typu man-in-the-middle, který spočívá v odchycení nešifrované komunikace mezi klientem a serverem (OWASP Top 10 - 2013, 2013, s. 12), či za pomoci jiných zranitelností aplikace, jako je injektování, XSS (Howard, 2009, s. 67), nezabezpečené odkazování apod.

Jak zranitelnost odhalit?

K odhalení této zranitelnosti obvykle nepomůže žádný automatizovaný skenovací software. Je zde zapotřebí analýzy ukládaných dat a způsobů jejich přenosu. Nejdříve je nutné jednoznačně určit, která data lze považovat za citlivá. Často jsou to taková data, která se dají přiřadit ke konkrétním reálným osobám a mají neveřejný charakter. Příkladem mohou být přístupová hesla, čísla kreditních karet, zdravotní záznamy apod.

Jak zranitelnost opravit?

Žádná citlivá data nesmějí nikdy být uložena nebo přenášena ve formě prostého textu (Howard, 2009, s. 204). K jejich ochraně také nepostačí jakákoliv forma šifrování a hashování, ale je potřeba vzít v potaz, že některé (převážně starší) algoritmy nejsou již bezpečné a útočník by je mohl prolomit. Kromě volby odpovídajícího šifrovacího algoritmu je potřeba i vytvářet silné šifrovací klíče, protože ty slabé by útočník mohl jednoduše uhodnout třeba pomocí útoku hrubou silou (Dorans, 2010, s. 118 - 119 a 124).

Příklad útoku:

Napadená aplikace nešifruje svoje data a je zároveň zranitelná útoky SQL injektováním. Útočník si jednoduše napíše SQL dotaz na vytažení všech záznamů z tabulky uživatelů, ve které jsou v prostém textu uložena i přístupová hesla k uživatelským účtům aplikace. Útočník tak nad těmito účty může získat kontrolu.

3.2.2.7 Chyby v řízení úrovní přístupů

Aplikace trpící těmito problémy obvykle ověřují přístupová oprávnění předtím, než se funkcionalita zviditelní na stránce, která se zasílá klientskému prohlížeči k zobrazení (OWASP Top 10 - 2013, 2013, s. 13). Problém nastává ve chvíli, kdy si informace o tom,

co se má zobrazit, posílají třeba v prostém textu v parametrech URL. Útočník může být ověřeným uživatelem systému a jednoduše upraví URL či některý z jejích parametrů, aby se dostal k obsahu či funkcionalitě, která mu nemá být dostupná (Dorans, 2010, s. 152), (Howard, 2009, s. 76).

Tato zranitelnost úzce souvisí s nezabezpečenými odkazy na objekty, které byly blíže rozebrány v podkapitole 3.2.2.4.

Jak zranitelnost odhalit?

Podobně, jako u expozice citlivých dat v podkapitole 3.2.2.6, je detekování těchto zranitelností velmi snadné. Problémem ale je najít místa (stránky i funkce), které mohou být takto napadnutelné. Nutností je analýza zdrojového kódu aplikace. Jedním z postupů je sledování konkrétních požadavků z pohledu omezeného i privilegovaného účtu a zjistit, jak se v obou případech chová autorizační systém (OWASP Top 10 - 2013, 2013, s. 13).

Kritickými místy bývají uživatelská rozhraní, která často dostatečně neošetřují přístup k funkcionalitám, které by měly být omezené.

Jak zranitelnost opravit?

Stejně jako v kapitole 3.2.2.4, i zde je dobré aplikaci opatřit centralizovaným autorizačním modulem, který bude snadno upravitelný i analyzovatelný. Na tento modul by pak měly být napojeny všechny ostatní funkce celé aplikace (Dorans, 2010, s. 154), (Introduction to ASP.NET Identity, 2013).

Příklad útoku:

Útočník se chce dostat do části s omezeným přístupem, která vyžaduje privilegovaný účet. Tím ale útočník nedisponuje:

```
http://nechejseokrast.cz/data/majitel=utocnik/
```

Útočník tedy pozmění parametr adresy a je aplikací vpuštěn:

```
http://nechejseokrast.cz/data/majitel=obet/
```

3.2.2.8 Cross-Site Request Forgery (CSRF)

Tento typ útoku spočívá v podsouvání podvržených požadavků k vykonání určitých akcí v napadané aplikaci skrze prohlížeč oběti. Využívá přitom spojení oběti s napadanou

aplikací. Oběť v takové aplikaci totiž obvykle bývá přihlášená (nebo k ní má legitimní přístup) a útočník ji nějakým způsobem (například za pomoci sociálního inženýrství) donutí otevřít si v prohlížeči zároveň i jím vytvořenou stránku, která samotný útok provádí (Dorans, 2010, s. 69).

Mnoho webových aplikací nezabrání útočníkovi, aby zjistil nebo odvodil podrobnosti jednotlivých akcí, které lze na stránce provádět. Ten je tudíž schopen vytvořit stránku, která je dokáže jednotlivé požadavky napodobit a zranitelná aplikace je nemůže rozeznat od těch skutečně legitimních (OWASP Top 10 - 2013, 2013, s. 6).

Jak zranitelnost odhalit?

K odhalení CSRF jsou velmi nápomocné penetrační skenery, které jsou schopné zranitelnost těmito útoky velmi snadno detekovat (OWASP Top 10 - 2013, 2013, s. 14).

Jak zranitelnost opravit?

Uznávaným řešením je používat tzv. unikátní tokeny, které se odesílají v HTTP požadavcích a jsou to náhodně generované řetězce znaků, které platí pouze pro aktuální relaci (Preventing Cross-Site Request Forgery (CSRF) Attacks in ASP.NET Web API, 2012). Možné je i generovat je znovu při každém požadavku na server a neumisťovat je do URL adresy, neboť tu by útočník mohl zjistit.

Tokeny je dobré doplnit dodatečnými ochranami, jako je šifrování komunikace či opakovaná autentizace (vložení hesla či CAPTCHA) u důležitých akcí, jako je třeba mazání záznamů nebo odesílání peněz z účtu (OWASP Top 10 - 2013, 2013, s. 14).

Příklad útoku:

Aplikace provádí přenos peněz mezi účty následujícím způsobem:

```
http://nechejseokrast.cz/data/prevod?zdrojovyUcet=obet&cilovyUcet=obet2&
mnozstvi=10000/
```

Útočník vytvoří stránku, na které nějakým způsobem (obrázkem, skriptem, apod.) vytvoří následující upravený požadavek:

```
http://nechejseokrast.cz/data/prevod?zdrojovyUcet=obet&cilovyUcet=utocni
k&mnozstvi=10000/
```

Oběť poté útočník nějakým způsobem příměje k otevření této stránky, čímž se podvržený požadavek odešle a jestliže je oběť k zranitelné aplikaci v tu dobu přihlášená, bude tento požadavek i autorizován a proveden.

3.2.2.9 Použití známých zranitelných komponent

Řada aplikací je vyvinuta s využitím knihoven či frameworků, které obsahují komponenty usnadňující vývoj i použití celé aplikace. Většina těchto komponent ale operuje s nejvyššími oprávněními, což souvisí s kapitolou 3.2.2.5. Pokud tyto komponenty trpí některými známými zranitelnostmi, může skrze ně dojít ke kompromitaci celé aplikace a jejích ochran (OWASP Top 10 - 2013, 2013, s. 6).

Jak zranitelnost odhalit?

Vývojáři musejí mít přehled o tom, jaké komponenty v systému používají. Samotná detekce zranitelností je pak reálně možná pouze sledováním informací autorů komponent, které jsou v aplikaci využívány (OWASP Top 10 - 2013, 2013, s. 15). Pokud se jedná o seriózní frameworky a knihovny, měly by včas varovat své zákazníky před bezpečnostními trhlinami ve svých výrobcích a vydávat pravidelné aktualizace a záplaty.

Jak zranitelnost opravit?

Pokud není možné se vyhnout využívání komponent třetích stran, je nezbytně nutné sledovat informace, které jsou v souvislosti s těmito komponentami k dispozici a pokud je vydána oprava, nainstalovat si ji. Řada tvůrců nevydává bezpečnostní záplaty ke starším verzím svých komponent a opravy chyb jsou vydávány až v nových verzích komponent. Proto je dobré číst i seznamy změn, které se v nových verzích komponent objeví. Dalším užitečným nástrojem jsou různé databáze známých bezpečnostních zranitelností a chyb, jako jsou CVE či NVD (OWASP Top 10 - 2013, 2013, s. 15).

3.2.2.10 Neošetřené přesměrování a předávání

Touto zranitelností trpí webové aplikace, které přesměrovávají uživatele na jiné stránky a cílovou adresu dostatečně nezabezpečí, takže si k ní útočník může zajistit přístup a pozměnit ji. Cílová adresa může být například obsažena v parametru adresy stránky (OWASP Top 10 - 2013, 2013, s. 16).

Tento typ zranitelnosti je nebezpečný především z hlediska možného neautorizovaného přístupu do některých části aplikace nebo v kombinaci se zranitelnostmi XSS či CSRF útoky. Ty mohou do počítače oběti stáhnout například škodlivý software, který by se stáhl právě po přesměrování na nebezpečnou stránku (Dorans, 2010, s. 41).

Jak zranitelnost odhalit?

Aplikace musí být v první řadě zabezpečená vůči útokům typu XSS a CSRF. Dále je potřeba zkontrolovat u všech přesměrování, která může aplikace iniciovat, zda je lze nějakým způsobem způsobem podvrhnout. Konkrétně, jestli adresa cílové stránky je potenciálním útočníkem dohledatelná například z parametru URL nebo třeba z nechráněného javascriptového kódu (Howard, 2009, s. 36).

V analýze mohou být nápomocné i nástroje typu crawler, které jsou schopné extrahovat informace z webových stránek.

Jak zranitelnost opravit?

Pokud není možné se vyhnout používání přesměrování v aplikaci, je nezbytné zajistit, aby cílová adresa nebyla závislá na uživatelských vstupech. Pokud toto není možné, mělo by před přesměrováním docházet k autorizaci požadavku, čehož lze dosáhnout tak, že tyto vstupy nejsou skutečnými adresami, ale pouze proměnnými, na jejichž základě se rozhoduje na serveru překládaný autorizační modul, jakou adresu přidělí a zda vůbec (OWASP Top 10 - 2013, 2013, s. 16).

Příklad útoku:

V aplikace probíhá přesměrování mezi jejími interními stránkami. Informace o tom, na kterou stránku se má uživatel přesměrovat příště, je k nalezení v parametru URL:

```
http://nechejseokrast.cz/moduly/stranka.aspx?url=stranka2.aspx
```

Útočník může upravit parametr URL dle své libosti.

3.3 Testování aplikace

Jedním z nejlepších způsobů, jak se vyhnout bezpečnostním chybám, je pamatovat na bezpečnost ve všech fázích životního cyklu vývoje softwaru (Software Development Life Cycle, tj. SDLC) (Meucci, 2015, s. 12). Jedním z procesů, které se touto problematikou blíže zabývají, je Security Development Lifecycle (SDL) společnosti Microsoft. Ten se skládá ze sedmi kroků, mezi něž patří i provádění penetračních testů (Dorans, 2010, s. 7).

3.3.1 Penetrační testování

Jedná se o formu technického auditu jehož hlavním cílem, v kontextu webových aplikací, je identifikovat možné zranitelnosti dané aplikace z pohledu útočníka. Provádí se formou útoku na testovanou aplikaci a to obvykle bez znalosti jejího vnitřního fungování (Meucci, 2015, s. 16). Při testování se využívají různé typy útoků, jako je injektování, XSS, CSRF a podobně. Výstupem penetračního testu pak je zpráva o stavu testované aplikace a nalezených potenciálních bezpečnostních chybách.

K provedení penetračních testů je možné využít automatizované nástroje, jako je WebScarab, Web Application Attack and Audit Framework (w3af) nebo OWASP Zed Attack Proxy (ZAP).

3.3.2 Zed Attack Proxy (ZAP)

ZAP je integrovaný penetrační nástroj určený k vyhledávání zranitelností webových aplikací a funguje, jako prostředník (proxy) mezi klientským prohlížečem a webovým serverem, což mu umožňuje vidět všechny požadavky klienta i odpovědi serveru (OWASP ZAP 2.4, b.r., s. 1). Aby bylo možné tímto nástrojem testovat i šifrované spojení, je třeba importovat tzv. certifikát ZAP Root CA do prohlížeče, který bude využit k testování aplikace. Všechny další certifikáty, které ZAP generuje pro jednotlivé stránky chráněné SSL, jsou s tímto kořenovým certifikátem propojené a tudíž budou automaticky uznány, jako důvěryhodné.

Před započetím testování je nutné stránky aplikace prozkoumat manuálně nebo pomocí robota na vyhledávání odkazů, kterému se říká pavouk (spider). Pavouk dokáže najít jednotlivé odkazy systematickým zkoumáním HTML odezvy aplikace. (OWASP ZAP 2.4, b.r., s. 2)

Jakmile ZAP získá přehled o stránkách aplikace, je možné přistoupit k samotnému testování, k čemuž slouží skenování dvojího druhu: pasivní a aktivní.

Pasivní skenování monitoruje všechny požadavky i odpovědi a žádným způsobem je nemění. Nejedná se tedy o útok a použití (na rozdíl od aktivního skenování) je bezpečné u jakékoliv stránky.

Aktivní skenování již využívá známé techniky útoků na konkrétní cíl (například frameworku). Ty nejběžnější a nejzávažnější byly rozebírány v kapitole 3.2.2. Z podstaty věci je možné aktivní skenování provádět pouze se svolením vlastníka testované aplikace a serveru, přestože je aktivní skenování pro aplikaci nezhoubné a neponičí například data v databázi. (OWASP ZAP 2.4, b.r., s. 2 - 3)

4 Praktická část

V této části práce bude vytvořena nová ASP.NET aplikace s využitím frameworku Web Forms. Bude navržen a zpracován správce přístupu k datům, který obstará komunikaci s databází a na řízení uživatelských přístupů bude implementovat systém ASP.NET Identity a jeho klíčová rozhraní. Závěrem bude aplikace otestována, včetně testu penetračním nástrojem ZAP 2.4.3.

Aplikace bude vypracována v prostředí Visual Studio 2013 v jazyce C# a jako databázový server bude použit MS SQL Server 2014. Jako webový server poslouží IIS 8.0 a webový prohlížeč k testování aplikace bude Mozilla Firefox verze 44.0.2. Použitá verze .NET frameworku bude 4.5.

4.1 Instalace a nastavení LocalDB

LocalDB je režim MS SQL Serveru speciálně určený pro vývoj aplikací. Výhodou je malá velikost souborů, které jsou nezbytné pro jeho běh a velmi snadná instalace a nastavení. Veškerá data se pak ukládají na lokální stroj vývojáře. LocalDB také podporuje sdílení instance mezi více uživateli počítače. (SQL Server 2014 Express LocalDB, b.r.)

Instalační balíček LocalDB Express je zdarma ke stažení na stránkách Microsoftu. Po jeho nainstalování je třeba vytvořit novou instanci, na které bude posléze založena vývojová databáze. Toto lze provést pomocí standardní příkazové řádky Windows příkazem SqlLocalDb, který umožňuje provádět několik operací na zadaných parametrech. Založení nové instance LocalDB a její nastartování se provede příkazem:

```
SqlLocalDb create "bp_core" -s
```

Operace *create* založí novou instanci s názvem *bp_core*. Parametr –s nově vytvořenou instanci spustí. Úspěšnost tohoto příkazu nám oznámí 2 hlášení, nejprve o založení a poté o nastartování instance. Alternativou je rozdělit postup na dva kroky:

```
SqlLocalDb create "bp_core"
SqlLocalDb start "bp_core"
```

Na další práci s nově založenou instancí bude použito Microsoft SQL Server 2014 Management Studio, které je také k dostání v express edici. Po jeho otevření je třeba zadat

jméno serveru, které v tomto případě zní *(localdb)\bp_core*. Způsob autentizace je třeba zatím ponechat na *Windows Authentication*, neboť nově vytvořená instance nemá žádné další uživatele.

Po úspěšném příhlášení se v levém panelu *Object Explorer* objeví jednotlivé instance SQL Serveru. Založení nového uživatele se provede kliknutím pravým tlačítkem na server *(localdb)\bp_core*, výběrem možnosti *New Query* a spuštěním následujícího dotaz, který založí nového uživatele *administrator* s heslem *cvx*-bdf67_RT-b6343ydh/+*. Spuštění dotazu se provede stisknutím tlačítka *Execute* nebo klávesou F5:

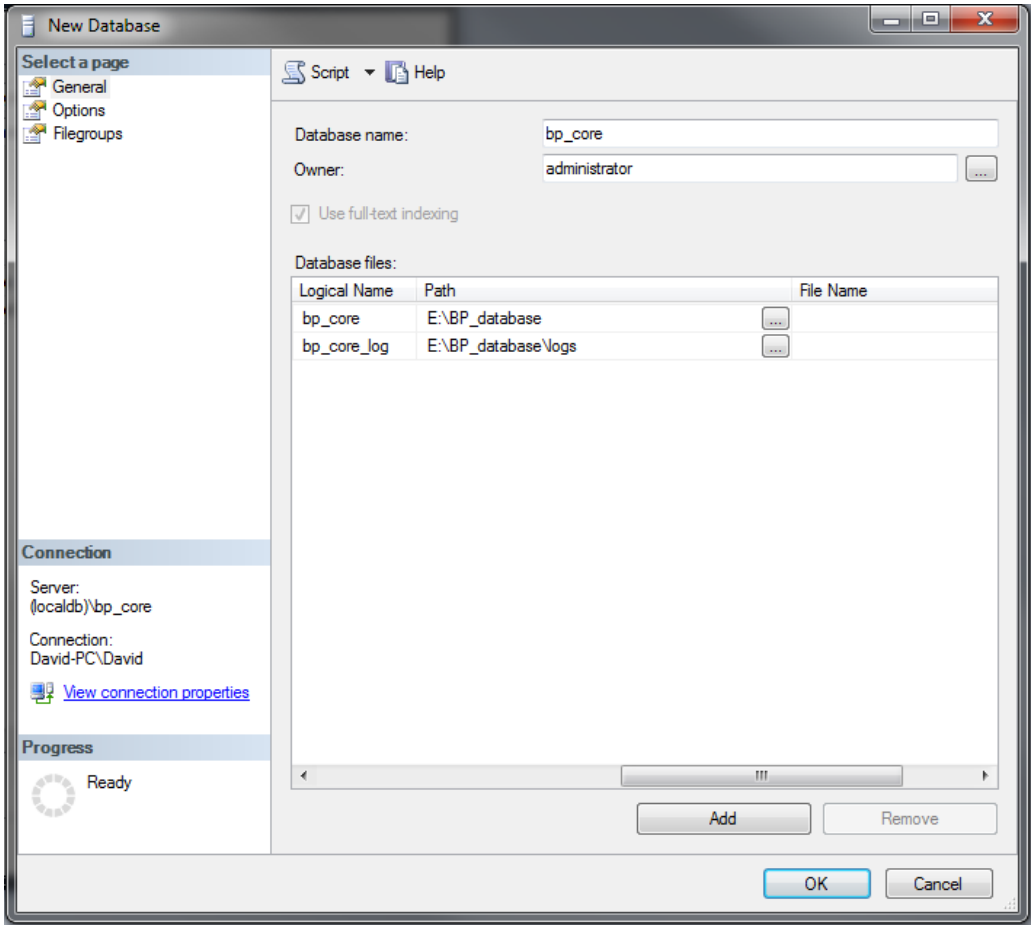
```
CREATE LOGIN administrator WITH PASSWORD = 'cvx*-bdf67_RT-b6343ydh/+';
```

Posledním nutným krokem v nastavování této instance LocalDB je přepnutí režimu autentizace serveru na možnost *SQL Server and Windows Authentication mode*, což umožní se k této instanci připojovat pomocí nově vytvořeného uživatele. Toto nastavení lze přepnout v okně vlastností serveru v záložce *Security*.

4.2 Založení databáze

Po úvodním založení a nastavení instance SQL Serveru, kterou se zabývala předchozí podkapitola, je nyní nutné vytvořit samotnout databázi. To se opět provede v panelu *Object Explorer* kliknutím pravým tlačítkem na položku *Databases* a vybráním první možnosti *New Database*.

V nově otevřeném okně je třeba vyplnit název databáze, vlastníka databáze a ideálně i změnit výchozí cesty k souborům v části *Database files*. Vlastníkem databáze bude v tomto případě nastaven nově vytvořený uživatel *administrator*, jak je ilustrováno na obrázku č. 2.



Obrázek 2: *Vytvoření databáze (vlastní zpracování)*

Jak bylo uvedeno v kapitole 3.1.1.3, ASP.NET Identity ve svém výchozím stavu potřebuje pracovat s uživateli, rolemi, uživatelskými přihlášeními a tvrzeními. V této práci se data budou ukládat do relační databáze a tudíž je nutné vytvořit tabulky a příslušné vazby mezi nimi.

Tabulka uživatelů (app_users) bude mít vazbu 1:m s uživatelskými přihlášeními (app_userLogins) a tvrzeními (app_userClaims), což znamená, že jedno tvrzení nebo přihlášení bude vázané právě na jednoho uživatele, ale uživatel může být spojen s neomezeným množstvím obou. Mezi uživateli a rolemi (app_roles) pak existuje vztah m:n, který je vyjádřen vazební tabulkou (app_users2Roles) a znamená, že jednotliví uživatelé mohou mít přiřazené libovolné množství rolí, tak i role může být přiřazena jakémukoliv počtu uživatelů.

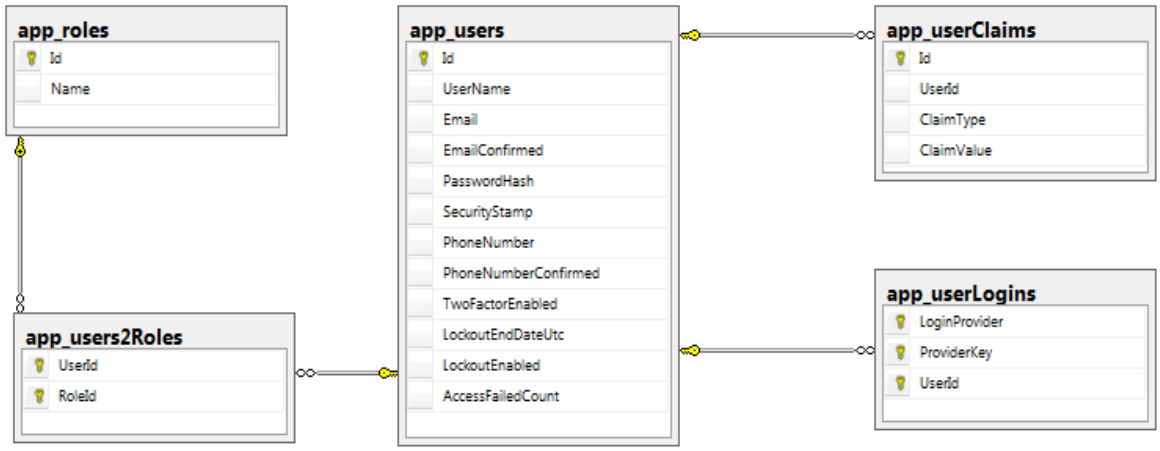
Celou strukturu lze vytvořit dvojím způsobem. První možností je napsání a spuštění skriptu, takový skript je k nahlédnutí v příloze B této práce. Druhou možností pak je využití grafického uživatelského rozhraní Management Studia, kde lze jednotlivé tabulky včetně

vazev ručně vytvořit. V panelu *Object Explorer* je třeba rozbalit nově vytvořenou databázi *bp_core* a pravým tlačítkem kliknout na *Tables* a poté *Table*. Po otevření nové záložky je možné vytvořit novou tabulku. Na obrázku č. 3 je vidět kompletní tabulka uživatelských tvrzení (userClaims), podle jejíhož vzoru je třeba vytvořit i tabulky ostatní.

Column Name	Data Type	Allow Nulls
Id	int	☐
UserId	int	☐
ClaimType	nvarchar(MAX)	☑
ClaimValue	nvarchar(MAX)	☑
		☐

Obrázek 3: *Tabulka app_userClaims (vlastní zpracování)*

Jakmile jsou jednotlivé tabulky vytvořené, lze je propojit vazbami. Toto není nikterak potřeba pro samotné fungování Identity, ale pro udržení integrity databáze. K vytvoření vztahů lze opět využít Management Studio. V panelu *Object Explorer* je třeba nejprve obnovit výpis tabulek kliknutím na položku *Tables* a vybráním možnosti *Refresh*. Poté je nutné postupně otevřít v režimu *Design* ty tabulky, ve kterých se nacházejí cizí klíče, což jsou tabulky uživatelských tvrzení a přihlášení a vazební tabulka mezi rolemi a uživateli. Samotná vazba se nastaví pomocí okna vazeb, které se pro jednotlivé tabulky otevírá kliknutím na ikonu *Relationships* v horní liště Management Studia. Výsledná struktura je názorně zachycena na obrázku č. 4.



Obrázek 4: *Databázový diagram (vlastní zpracování)*

Na závěr je třeba vytvořit roli *AccountAdministrator*, která bude sloužit k omezení přístupu na určité stránky aplikace. K tomu je nutné kliknout pravým tlačítkem na databázi *bp_core*

v panelu *Object Explorer* a vybrat možnost *New Query*. Dotaz k jejímu vložení do tabulky app_roles vypadá následovně:

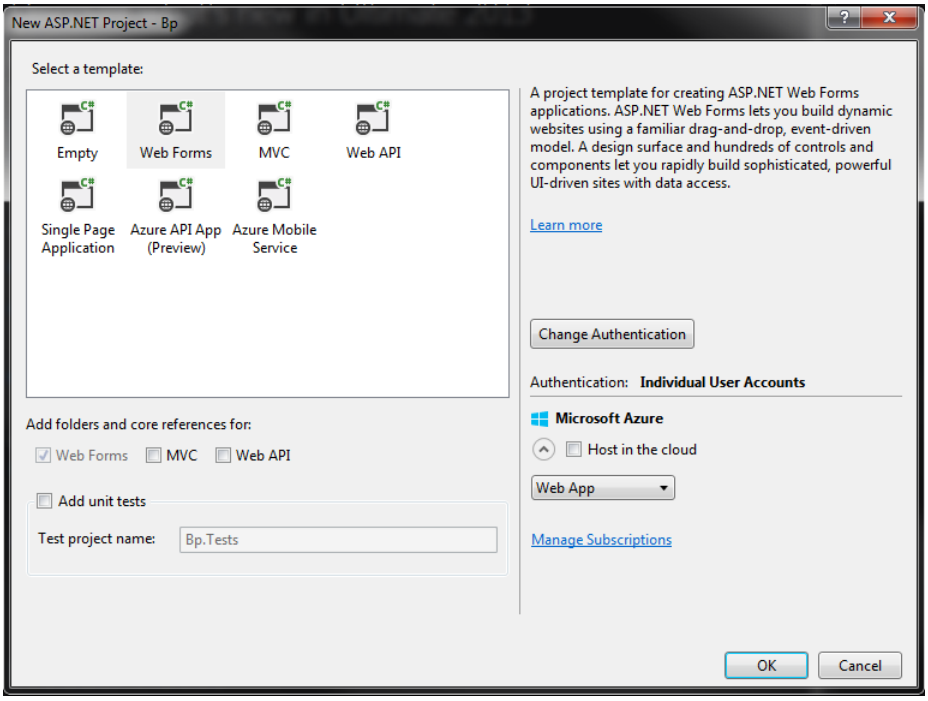
```
INSERT INTO [dbo].[app_roles] VALUES ('AccountAdministrator')
```

Dotaz se spustí tlačítkem *Execute* v horní liště Management Studia.

4.3 Založení a nastavení hlavního projektu

Po krocích uvedených v předchozích dvou podkapitolách je databáze nyní připravena k použití a je tedy vhodný čas přistoupit k vytvoření samotného projektu, k čemuž bude použita anglická verze programu Visual Studio 2013 s nainstalovaným balíčkem šablon.

Po otevření Visual Studia a zvolení možnosti *New Project* v levé horní části ovládací lišty programu, se objeví okno s nabídkou všech šablon, které jsou k dispozici. Nemusí se jednat pouze o šablony poskytnuté společností Microsoft, ale mohou mezi nimi být i takové, jež pro své produkty vytvořili například výrobci různých frameworků nainstalovaných na počítači. Pro vytvoření projektu je třeba vybrat možnost *ASP.NET Web Application* v jazyce C# v sekci *Installed*, vyplnit název do políčka *Name* a také umístění celého projektu přes tlačítko *Browse...* ve spodní části okna. Verze .NET frameworku musí být nastavená na verzi 4.5. Po stisknutí tlačítka *OK* se předchozí okno zavře a objeví se nové, které je vidět na obrázku č. 5.



Obrázek 5: *Vytváření nového projektu (vlastní zpracování)*

Zde je důležitý typ autentizace viditelný v pravé části okna, který se automatický přednastaví na *Individual User Accounts*. Tato možnost, kromě jiného, do nového projektu zavede ASP.NET Identity, Entity Framework, který je frameworkem objektově relačního mapování (ORM) společnosti Microsoft, a OWIN implementaci jak Identity, tak i hlavních externích poskytovatelů autentizace, jako jsou Google, Twitter či Facebook. Přehled o všech balíčcích, které se do projektu nainstalují, bude možné získat v souboru *packages.config* umístěném v kořenovém adresáři projektu po jeho založení.

Typ autentizace lze změnit nebo zcela vypnout pomocí tlačítka *Change Authentication*, ale pro potřeby této práce je třeba ponechat zvolenou možnost *Individual User Accounts*. Založení nového projektu se potvrdí tlačítkem *OK*.

Vytvořením projektu pomocí šablony došlo i k vytvoření mnoha aspx stránek, pro které nemá tato práce žádné využití. Proto je třeba tyto stránky odstranit přidržením klávesy CTRL a postupným označováním jednotlivých stránek určených ke smazání. Jedná se o všechny stránky kromě *Manage.aspx, ManagePassword.aspx, Default.aspx, Register.aspx, Login.aspx* a *Lockout.aspx*. Některé se nacházejí v adresáři *Account*. Následně je nutné adresář *Account* přejmenovat na *Modules* a vložit do něj nový podadresář *Restricted*, do kterého se budou vkládat stránky s omezeným přístupem. Vytvořit jej lze kliknutím pravým tlačítkem na nově přejmenovaný adresář *Modules* a výběrem možnosti *Add* a *New Folder*. Do této nové složky je pak třeba přesunout stránky *Manage.aspx* a *ManagePassword.aspx*. Po přejmenování adresářů a přesunu stránek, je třeba projít postupně veškeré soubory s příponou .cs, protože v řadě z nich je odkazováno na nyní již neexistující umístění stránek. Toto je k vidění například v *Startup.Auth.cs* u hodnoty vlastnosti *LoginPath*. Vhodné je rovnou přistoupit i k úpravám odkazování u zbylých stránek .aspx včetně *Site.Master*. Posledním úkonem bude přidání souboru *Web.config* přes možnost *Add, New File* a *Web Configuration File* dostupné opět klikem pravého tlačítka na složku *Restricted*. Tento konfigurační soubor bude obsahovat následující:

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <authorization>
      <allow roles="AccountAdministrator" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

Toto nastavení efektivně znemožní přístup ke stránkám v tomto adresáři každému, kdo nedisponuje rolí *AccountAdministrator*.

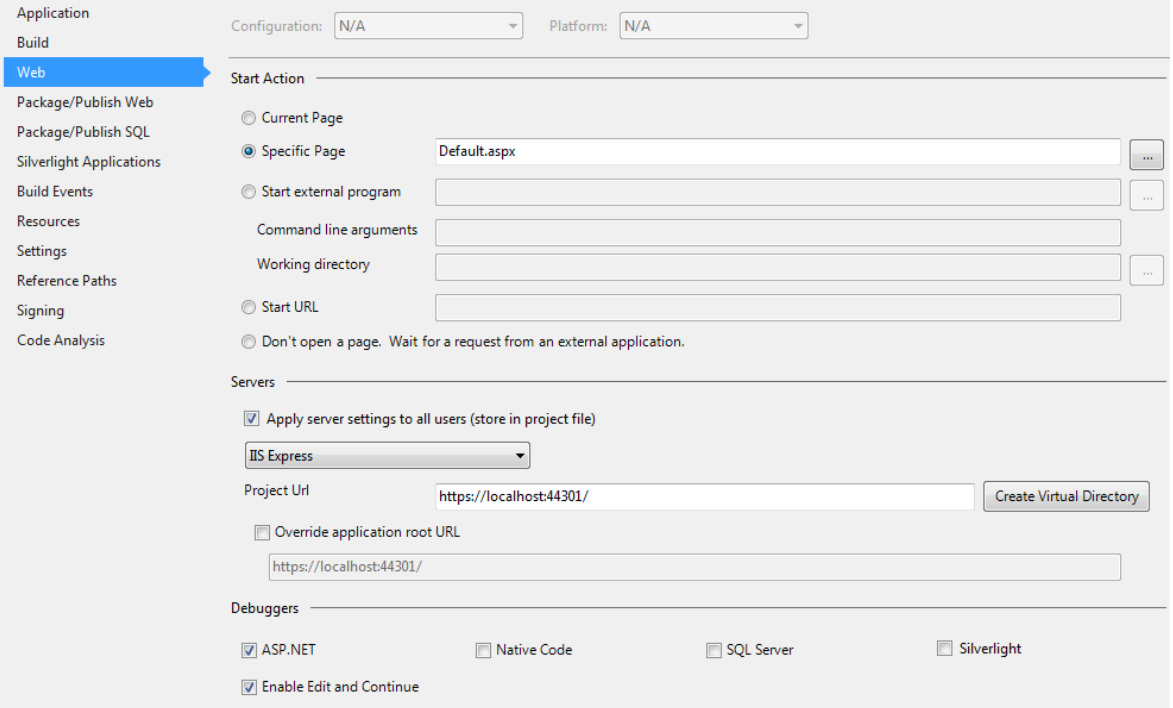
Následujícím krokem v nově vytvořeném projektu je nastavení připojení k databázi. Toto je možné provést v hlavním souboru *Web.config*, který obsahuje informace o konfiguraci aplikace ve formě XML elementů a je k nalezení v kořenovém adresáři projektu přes panel *Solution Explorer*. Pokud tento panel není viditelný, je nutné jej zapnout pomocí nabídky *View* v horní liště Visual Studia. V tomto souboru *Web.config* se nachází element *connectionStrings* a v něm podelement *add* s několika atributy, z nichž nejdůležitější je *connectionString*, jehož hodnota určuje, jak se aplikace bude připojovat ke svému uložišti dat. Tuto hodnotu je potřeba změnit podle vzoru:

```
<add name="DefaultConnection"
connectionString="server=(LocalDb)\bp_core;database=bp_core;uid=administ
rator;pwd=cvx*-bdf67_RT-b6343ydh/+;"
providerName="System.Data.SqlClient" />
```

Nová hodnota tak bude odkazovat na databázi a instanci SQL serveru LocalDB, jejichž vytvářením a nastavením se zabývaly podkapitoly 4.1 a 4.2.

Jelikož zaměřením této práce je bezpečnost, aplikace musí podporovat šifrovanou komunikaci. Toto je možné nastavit velmi snadno kliknutím na projekt *Bp* v panelu *Solution Explorer*, čímž se zobrazí jeho nastavení v panelu *Properties*. Zde je, na řádku *SSL Enabled*, třeba přepnout rozbalovací seznam na položku *True*. Tímto se vyplní i *SSL URL*. Pokud není panel *Properties* viditelný, lze jej zapnout, podobně jako panel *Solution Explorer*, přes nabídku *View* a výběr možnosti *Properties Window*.

Dalším krokem, je specifikování stránky, která se bude zobrazovat při startu aplikace a také URL adresy projektu. Obojí lze nastavit na stránce nastavení projektu, kterou lze otevřít kliknutím pravým tlačítkem na projekt *Bp* a poté výběrem možnosti *Properties*. Hlavní stránkou projektu je třeba nastavit stránku Default.aspx, což se provede jejím vyhledáním přes tlačítko ... na konci řádku *Specific Page*. Do řádku *Project URL* je pak nutné vyplnit *SSL URL* z panelu *Properties*, kde se aktivovalo SSL.

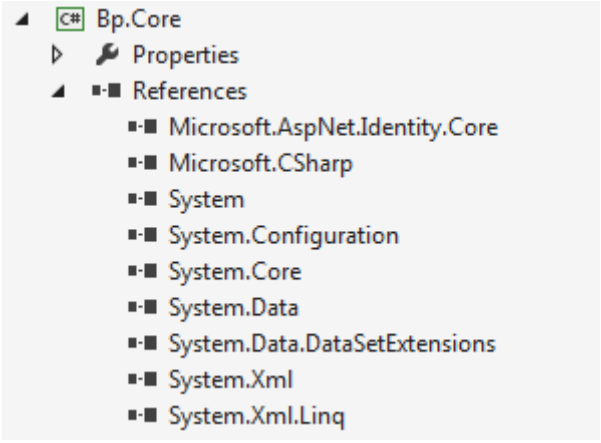


Obrázek 6: *Nastavení projektu (vlastní zpracování)*

4.4 Založení knihovny jádra

O komunikaci s databází se v řadě projektů starají ORM frameworky, jako je Entity Framework, NHibernate nebo Dapper. Jejich výhodou je schopnost pracovat s různými typy databází či podpora lazy loadingu. Především se ale starají o synchronizaci dat mezi objekty aplikace a databáze. K výše zmíněným frameworkům jsou také dostupné i jejich implementace rozhraní ASP.NET Identity. Jako výchozí ORM je v šablonách Microsoftu použit Entity Framework. Součástí hlavního cíle této práce ale je demonstrovat flexibilitu a základní funkcionalitu systému Identity a tudíž bude v následujících podkapitolách postupně Entity Framework nahrazen autorovou vlastní implementací jak přístupu k databázi a datům, tak i klíčových rozhraní ASP.NET Identity.

Po založení hlavního projektu je dalším krokem ve vývoji aplikace je založení knihovny s názvem *Bp.Core*, která bude obsahovat všechny třídy nezbytné k práci s Identity a také třídu obsahující CRUD operace nad databází. Vytvořit novou knihovnu lze kliknutím pravým tlačítkem na *Solution* v panelu *Solution Explorer*, výběrem možnosti *Add* a poté *New Project*. Tímto se vyvolá okno, přes které se v podkapitole 4.3 vybírala šablona pro projekt. Tentokrát je ale třeba v sekci *Installed* vybrat možnost *Class Library*. K nově vzniklé knihovně je nutné připojit dodatečné reference viz obrázek č. 7:



Obrázek 7: *Reference knihovny jádra (vlastní zpracování)*

Toto se provede kliknutím pravým tlačítkem myši na *References* a zvolením možnosti *Add Reference*.

System.Configuration je k nalezení přes vyhledávací pole v pravém horním rohu okna, ale knihovna *Microsoft.AspNet.Identity.Core* se nachází v kořenovém adresáři hlavního projektu ve složce *packages* a bude ji nutné vyhledat pomocí tlačítka *Browse…* v pravém dolním rohu.

Založení knihovny je investicí do budoucnosti, protože implementace jádra tak, jak je popsána v této práci, může být použita i v jiných projektech a je tedy rozumné tento vývoj již od začátku usměrnit, aby nebylo potřeba jednotlivé třídy mezi projekty kopírovat, ale jednoduše k nim připojit celou knihovnu.

4.5 Přístup k databázi

Komunikaci s databází bude řídit třída *DbAccessManager*, kterou lze vytvořit kliknutím pravým tlačítkem na nově vzniklý projekt knihovny a výběrem možnosti *Add* a poté *New Item…* a *Class*.

Nově vytvářený soubor i třídu je tedy třeba pojmenovat *DbAccessManager*. Třída bude implementovat rozhraní *IDisposable*, které obsahuje pouze hlavičku metody *Dispose*, jejímž smyslem je uvolňování nepotřebných zdrojů. V tomto případě bude uzavírat spojení s databází. Do třídy *DbAccessManager* je v první řadě třeba přidat dvě privátní proměnné a konstruktory:

```
private SqlConnection _connection;
private SqlCommand _command;

public DbAccessManager() : this("DefaultConnection")
{}

public DbAccessManager(string connectionStringName)
{
    if (string.IsNullOrWhiteSpace(connectionStringName))
        throw new ArgumentException("Nebyl poskytnut connectionString k
databázi!");

    string connectionString =
ConfigurationManager.ConnectionStrings[connectionStringName].ConnectionS
tring;
    _connection = new SqlConnection(connectionString);
}
```

První konstruktor bez parametru pouze odkazuje na druhý, kterému vkládá jako vstupní parametr hodnotu *DefaultConnection*, což je název connection stringu, jehož zaváděním se zabývala podkapitola 4.3. Konstruktor dále založí novou instanci *SqlConnection* a vloží ji do privátní proměnné.

Samotné spojení pak budou spravovat metody *Connect*, *Disconnect* a *ValidateConnection*. Metoda *Connect* otevírá spojení s databází a také vytváří objekt *SqlCommand*, kterému se předávají parametry a informace k SQL dotazům či procedurám. *SqlCommand* je poté vykoná a vrátí výsledek ke zpracování. Metoda *Disconnect* uzavírá otevřené spojení a metoda *ValidateConnection* kontroluje, zda je spojení s databází otevřené a pokud ne, otevře jej:

```
private bool ValidateConnection()
{
    if (_connection != null && _connection.State==ConnectionState.Open)
        return true;
    else
        return Connect();
}
```

Dalšími klíčovými metodami, které musí *DbAccessManager* obsahovat, jsou *ExecuteNonQuery*, *GetDataTable* a *ExecuteScalar*. Tyto metody budou použity pro samotnou práci s daty. Metoda *ExecuteNonQuery* vypadá následovně:

```
internal int ExecuteNonQuery(String sql, CommandType type)
{
    if (string.IsNullOrWhiteSpace(sql))
        throw new ArgumentException("Text příkazu musí být vyplněn!");

    ValidateConnection();

    _command.CommandText = sql;
    _command.CommandType = type;

    return _command.ExecuteNonQuery();
}
```

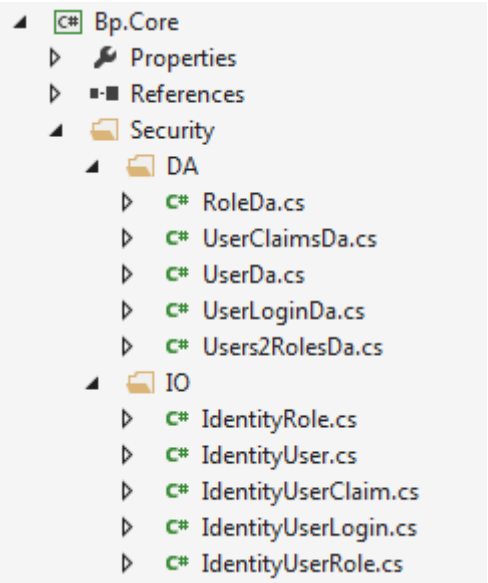
K zajištění všech potřeb tříd, jimiž se zabývají nadcházející podkapitoly, jsou zapotřebí i další metody, jež jsou k nahlédnutí ve zdrojovém kódu v příloze B.

4.6 Data adaptéry a třídy Identity

Jak již bylo řečeno na začátku podkapitoly 4.4, šablony Microsoftu používají ve výchozím stavu Entity Framework pro práci s databází. Aby jej bylo možné odstranit, je nutné vytvořit vlastní implementace ASP.NET Identity rozhraní. Tato podkapitola se zabývá implementací rozhraní *IRole* a *IUser* a pomocnými třídami, které budou vykonávat konkrétní CRUD operace nad příslušnými databázovými tabulkami. Tyto pomocné třídy budou, v rámci této práce, souhrnně nazývány data adaptéry a ponesou postfix *Da*.

Pro udržení přehlednosti je třeba v knihovně *Bp.Core* vytvořit novou složku *Security* s podsložkami *DA* a *IO*. První podsložka (DA) bude obsahovat všechny třídy typu data adaptér a druhá (IO) pak všechny třídy představující tabulky databáze, mezi kterými budou i třídy implementující rozhraní *IRole* a *IUser*. Tyto třídy ponesou prefix *Identity*.

V plném rozsahu je obsah složek DA a IO zachycen na obrázku č. 8:



Obrázek 8: *Data Access Layer (vlastní zpracování)*

4.6.1 Třídy Identity

První založenou třídou bude *IdentityRole*, která implementuje rozhraní *IRole* takto:

```
using Microsoft.AspNet.Identity;
using System;

namespace Bp.Core
{
    public class IdentityRole : IRole<int>
    {
        public IdentityRole()
        {}

        public IdentityRole(string name)
        {
            Name = name;
        }

        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

V definici rozhraní *IRole* je použit generický návratový typ primárního klíče Id, což umožňuje použití libovolného konkrétního datového typu. Výchozím datovým typem je string, a není jej nutné specifikovat ve špičatých závorkách, jako ostatní typy. V kapitole 4.2

byly ale primární klíče všech tabulek nastaveny na typ int (celé číslo), a tak je třeba jej použít i zde. Povinnými atributy jsou pouze *Id* a *Name*, přičemž třída může používat i další. To nicméně pro potřeby této práce není nutné, jelikož i databázová tabulka *app_roles*, která je v aplikaci představována touto třídou, nemá více sloupců.

Podobným způsobem je třeba implementovat i rozhraní *IUser* do třídy *IdentityUser*. V tomto případě ale bude třída, kromě povinných atributů *Id* a *UserName*, obsahovat i všechny ostatní atributy, které odpovídají svým protějškům v databázové tabulce *app_users*.

Všechny třídy s prefixem *Identity* jsou k dohledání ve zdrojovém kódu v příloze B.

4.6.2 Data adaptéry

Jedná se o pomocné třídy, které budou vykonávat konkrétní databázové operace. Každý data adaptér se věnuje práci s tabulkou a objektem, který mu logicky náleží. *RoleDa* tedy bude pracovat s tabulkou *app_roles* a objekty typu *IdentityRole*. Hlavička třídy *RoleDa*, její konstruktory a privátní proměnné budou vypadat takto:

```
public class RoleDa<TRole> where TRole : IdentityRole
{
    private DbAccessManager _database;

    public RoleDa()
    {
        _database = new DbAccessManager();
    }

    public RoleDa(DbAccessManager database)
    {
        _database = database;
    }
```

Jak je na zdrojovém kódu výše vidět, data adaptéry používají instanci třídy *DbAccessManager*, navržené v kapitole 4.5, které předávají k vykonávání jednotlivé operace připravené vždy v metodách, jež se dají podle typu rozdělit na insertovací, updatovací, deletovací a selectovací. Tyto metody je vždy třeba vytvářet podle následujícího vzoru:

```
public int Insert(TRole role)
{
    string commandText = "INSERT INTO app_roles (Name) OUTPUT
INSERTED.ID VALUES (@name)";

    try
    {
      _database.ClearParameters();

      _database.AddParameter("name", role.Name, SqlDbType.NVarChar);

        return (int)_database.ExecuteScalar(
            commandText, CommandType.Text);
    }
    catch
    {
        throw;
    }
    finally
    {
        _database.Disconnect();
    }
}
```

Výše je uvedena metoda *Insert* třídy *RoleDa*, která na svém samotném začátku definuje databázovou operaci, poté přípraví seznam parametrů a nechá *DbAccessManager* postarat se o vykonání. Závěrem je třeba vždy zavřít spojení s databází, které se před vykonáním operace otevře. Návratovou hodnotou metod Insert musí být Id, které bylo při vložení do tabulky záznamu přiřazeno.

Podle této předlohy musejí být navrženy i ostatní potřebné metody jakéhokoliv data adaptéru. Jednotlivé třídy je opět možné si prohlédnout v příloze B této práce.

4.7 Sklady (Stores)

Sklady jsou posledními třídami, které musejí implementovat některé z rozhraní systému ASP.NET Identity a právě kvůli nim byly nutné přípravy rozebrané v podkapitolách 4.5 a 4.6, neboť Identity používá sklady na přístup k datům. Nezbytné je zavedení dvou tříd, kterými jsou *RoleStore* a *UserStore*. Pro třídy typu sklad je třeba zavést nový podadresář *Stores* do adresáře *Security* v knihovně *Bp.Core*.

Třída *RoleStore* implementuje pouze rozhraní *IRoleStore* a to následujícím způsobem:

```
public class RoleStore<TRole> : IRoleStore<TRole, int>
    where TRole : IdentityRole
{
    private RoleDa<TRole> _roleDa;

    public RoleStore()
    {
        _roleDa = new RoleDa<TRole>();
    }

    public RoleStore(DbAccessManager database)
    {
        _roleDa = new RoleDa<TRole>(database);
    }
```

Toto rozhraní opět umožňuje práci s generickými typy a tak je možné využívat typ *IdentityRole*, kterým se (mimo jiné) zabývala předchozí podkapitola. Ten pracuje s celočíselnými primárními klíči a tudíž je v definici hlavičky třídy *RoleStore* nutné specifikovat i typ klíče, jímž je int.

Rozhraní *IRoleStore* vyžaduje zavedení řady metod s návratovým typem *Task*. Mezi ně patří i *CreateAsync*, která poslouží jako názorný příklad, jakým způsobem lze zavést i ostatní požadované metody:

```
public Task CreateAsync(TRole role)
{
    if (role == null)
    {
       throw new ArgumentNullException("Parametr role musí být zadán!");
    }

    int newId = _roleDa.Insert(role);
    role.Id = newId;

    return Task.FromResult(role);
}
```

Metoda *CreateAsync* bude v této implementaci požadovat nenulový vstupní parametr *role*, který poté předá data adaptéru k uložení. Na konci se poté vrátí *role* s nově vytvořeným Id. Přestože by metoda mohla podporovat asynchronní volání, v tomto případě se jedná o volání synchronní a tudíž je jako výstup použit *Task.FromResult*. Obdobným způsobem jsou vyřešeny i ostatní metody rozhraní *IRoleStore*.

Druhou zmiňovanou třídou je *UserStore*. Ta, oproti *RoleStore*, implementuje hned několik rozhraní najednou. Její hlavička vypadá následovně:

```
public class UserStore<TUser> : IUserStore<TUser, int>,
                                IUserLoginStore<TUser, int>,
                                IUserRoleStore<TUser, int>,
                                IUserPasswordStore<TUser, int>,
                                IUserSecurityStampStore<TUser, int>,
                                IUserEmailStore<TUser, int>,
                                IUserPhoneNumberStore<TUser, int>,
                                IUserTwoFactorStore<TUser, int>,
                                IUserLockoutStore<TUser, int>,
                                IUserClaimStore<TUser, int>
        where TUser : IdentityUser
```

Podobně jako v třídě *RoleStore*, i zde je nutné specifikovat typ primárního klíče. Zavedení rozhraní *IUserStore* je jediné povinné, nicméně ostatní je vhodné implementovat taktéž, neboť se vážou ke zbylým sloupcům databázové tabulky *app_users*, jako jsou PhoneNumber nebo Email.

Díky množství zaváděných rozhraní dojde k využití všech adaptérů:

```
private UserDa<TUser> _usersDa;
private UserLoginDa<TUser> _userLoginsDa;
private RoleDa<IdentityRole> _roleDa;
private Users2RolesDa<TUser> _users2RolesDa;
private UserClaimsDa<TUser> _userClaimsDa;
```

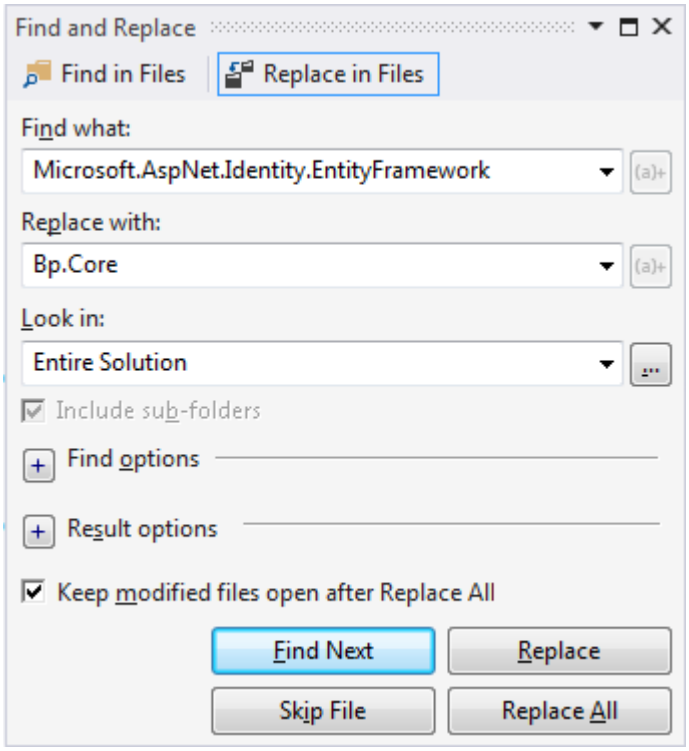
Metod vyžadujích implementaci je opět celá řada, nicméně celý proces je stejný, jako u tvorby *RoleStore*. Kompletní třída *UserStore* je obsažena v příloze B.

4.8 Napojení knihovny na hlavní projekt

V této fázi jsou u konce všechny nezbytné práce na knihovně jádra *Bp.Core* a tudíž ji lze napojit na hlavní projekt *Bp*. Toto se provede přes správce referencí, který byl popsán v kapitole 4.4. Tentokrát je ale třeba přidat referenci u hlavního projektu *Bp* a vybrat knihovnu *Bp.Core* ze seznamu *Solution* v levé části okna správce. Další krokem je odebrání NuGet balíčků vázajících se na Entity Framework. Je tedy nutné kliknout pravým tlačítkem na projekt *Bp* a vybrat možnost *Manage NuGet Packages…*, čímž se zobrazí okno správce balíčků v jehož levém panelu je třeba vybrat možnost *Installed packages* a zde odinstalovat balíčky s ID *Microsoft.AspNet.Identity.EntityFramework*, *Microsoft.AspNet.Providers.Core*

a *EntityFramework* v tomto pořadí. Balíčky lze také vyhledávat zadáním tohoto jejich ID do vyhledávače v pravém horním rohu.

Tímto krokem se aplikace dostala do nesestavitelného stavu, neboť odinstalace samozřejmě neodstranila použité obory názvů Entity Frameworku v jednotlivých třídách. Zde je oprava stále snadná, protože stačí stisknout klávesovou zkratku CTRL + SHIFT + F, čímž se objeví okno *Find and Replace*, kde je třeba vybrat záložku *Replace in Files*. Políčko *Find what:* je nutné vyplnit hodnotou *Microsoft.AspNet.Identity.EntityFramework* a políčko *Replace with:* poté hodnotou *Bp.Core*. Rozbalovací seznam *Look in:* musí být nastaven na položce *Entire Solution*. Celou operaci je možné potvrdit tlačítky *Replace* nebo *Replace All*.



Obrázek 9: *Okno Find and Replace (vlastní zpracování)*

Celkem by k nahrazení oboru názvů mělo dojít čtyřikrát, a to v souborech *IdentityModels.cs*, *IdentityConfig.cs*, *Startup.Auth.cs* a *Manage.aspx.cs*.

Další zásah je třeba provést v hlavním souboru *Web.config* v kořenovém adresáři projektu *Bp*. Odtud lze vymazat celé elementy *configSections* a *entityFramework*. Z elementu *assemblyBinding* pak jeho podelement *dependentAssembly* odkazující na Entity Framework.

Finální krok je zdaleka nejpracnější a vyžaduje řádově desítky úprav. Těmi nejvýraznějšími jsou úpravy v souborech *IdentityModels.cs* a *Startup.Auth.cs*.

V souboru *IdentityModels.cs* je třeba třídu *ApplicationDbContext* nově dědit od třídy *DbAccessManager*, která řeší komunikaci s databází na místo třídy *IdentityDbContext*, která spadá pod obor názvů *Microsoft.AspNet.Identity.EntityFramework*. Poslední úpravou třídy *ApplicationDbContext* je odebrání parametrů předávaných rodičovskému konstruktoru, neboť základní konstruktor třídy *DbAccessManager* tak, jak byl navržen v kapitole 4.5, žádné parametry nevyžaduje.

Změny v souboru *Startup.Auth.cs* se týkají volání metody *UseCookieAuthentication*, kdy je třeba upravit především hodnotu vlastnosti *LoginPath* tak, aby odkazovala na *Login.aspx* ve složce *Modules*. Druhou úpravou je nahrazení delegáta přiřazeného vlastnosti *OnValidateIdentity*, tak aby odpovídal následujícímu:

```
OnValidateIdentity =
SecurityStampValidator.OnValidateIdentity<ApplicationUserManager,Applica
tionUser,int>
(
    validateInterval: TimeSpan.FromMinutes(30),
    regenerateIdentityCallback: (manager, user) =>
            user.GenerateUserIdentityAsync(manager),
    getUserIdCallback: (id) => (id.GetUserId<int>())
)
```

Tato změna v souboru *Startup.Auth.cs* je nutná vzhledem k datovému typu Id uživatele, kterého v aplikaci představuje třída *IdentityUser* a její potomci. Obdobný problém lze pozorovat i v jiných zdrojových souborech projektu. Efektivní metodou řešení je pokusit se o sestavení celé aplikace klávesou F6 nebo výběrem možnosti *Build Solution* v horní liště Visual Studia. Toto sestavení selže, ale zároveň se příslušné chyby zobrazí v panelu *Error List*, který je možné vyvolat z nabídky *View*. Zobrazení chyby se provede dvojitým poklikem na onu chybu v seznamu. Tímto způsobem není nutné procházet všechny soubory manuálně, ale opravovat chyby systematicky, dokud sestavení neproběhne úspěšně. Většina chyb se týká pokusu o implicitní konverzi typu *int* na typ *string*. Oprava obvykle spočívá ve ve změně klíče uživatele z typu *string* na typ *int*. Pro názornost lze uvést tento příklad:

Chybové volání metody *GetUserId*:

```
User.Identity.GetUserId()
```

Opravené volání:

```
User.Identity.GetUserId<int>()
```

Po odstranění všech chyb tohoto typu již bude aplikace zcela připojena na nové jádro. Pokud tak již nebylo učiněno dříve jak naznačila kapitola 4.3, je třeba ještě provést úpravy odkazů a uživatelského rozhraní na .aspx stránkách. Zejména se jedná o opravy cest ke stránkám po jejich přesunu do nových složek, smázání odkazů na neexistující stránky a přeložení šablony do českého jazyka.

4.9 Custom Error Handling

Dobrou praxí je vytvářet vlastní stránku pro zobrazování chyb, které mohou nastat při běhu aplikace. Taková stránka by měla zapadat svým designem do kontextu celé aplikace a může zobrazovat podrobnosti o jednotlivých chybách nebo obecné hlášení o problému, které uživateli neřekne nic bližšího, ale také nepůsobí tak rušivým dojmem, jako běžný výpis. Druhotnou výhodou je i to, že potenciálnímu útočníkovi neodhalí nic, co si vývojář vysloveně nepřeje. Zranitelnostem, kdy jsou chyby aplikace a webového serveru zobrazovány v plném rozsahu, se říká *Application Error Disclosure*.

Novou stránku lze vytvořit opět kliknutím pravým tlačítkem na projekt *Bp* a výběrem možnosti *Add* a *New Item*. Zde je třeba vybrat možnost *Web Form* a jako jméno vložit *Error*. Nově vytvořená stránka se musí upravit dle následujícího vzoru:

```
<%@ Page Title="Chyba" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Error.aspx.cs"
Inherits="BpRoles.Error" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent"
runat="server">

    <div class="jumbotron" style="color: red;">
        <h4>Nastala neočekávaná chyba!</h4>
    </div>

</asp:Content>
```

Oproti původně vygenerované stránce je zde implementována *MasterPage* s názvem *Site.Master* a komponenta *Content*, jejíž ID musí být shodné s ID komponenty *ContentPlaceHolder* na stránce *Site.Master*. Na tomto místě se totiž, ve výsledné stránce zobrazované prohlížečem, vykreslí obsah komponenty *Content*.

Nyní je třeba v hlavním souboru Web.config nastavit aplikaci i webový server, aby při chybě přesměrovali stránku na Error.aspx. Toho lze docílit přidáním

```
<customErrors mode="On" redirectMode="ResponseRewrite"
defaultRedirect="~/Error.aspx">
</customErrors>
```

do podelementu *system.web*, čímž se přesměrují chyby aplikace, a

```
<httpErrors errorMode="Custom">
  <remove statusCode="401" subStatusCode="-1" />
  <remove statusCode="403" subStatusCode="-1" />
  <remove statusCode="404" subStatusCode="-1" />
  <remove statusCode="500" subStatusCode="-1" />

  <error statusCode="401" subStatusCode="-1" prefixLanguageFilePath=""
        path="/Error.aspx" responseMode="ExecuteURL" />
  <error statusCode="403" subStatusCode="-1" prefixLanguageFilePath=""
         path="/Error.aspx" responseMode="ExecuteURL" />
  <error statusCode="404" subStatusCode="-1" prefixLanguageFilePath=""
         path="/Error.aspx" responseMode="ExecuteURL" />
  <error statusCode="500" subStatusCode="-1" prefixLanguageFilePath=""
         path="/Error.aspx" responseMode="ExecuteURL" />
</httpErrors>
```

do podelementu *system.webServer*, což přesměruje HTTP chyby hlášené webovým serverem. Do stejného podelementu je také třeba přidat

```
<httpProtocol>
  <customHeaders>
    <add name="X-Frame-Options" value="SAMEORIGIN" />
  </customHeaders>
</httpProtocol>
```

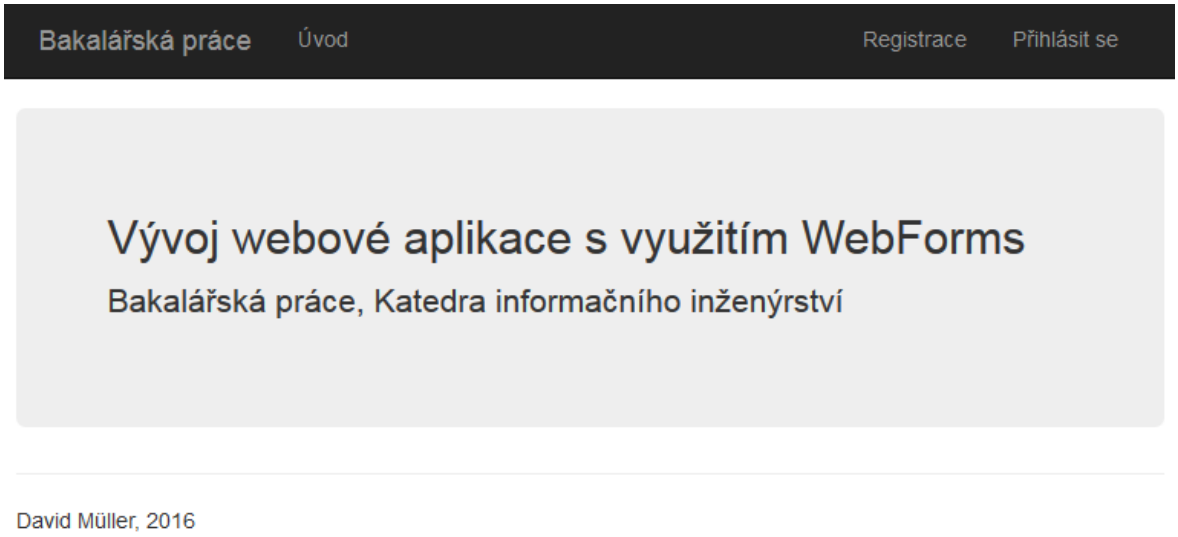
Což zamezí útokům typu *Clickjacking*. Jako poslední je třeba upravit soubor *Global.asax*, do kterého se přidá implementace události *OnApplicationError*:

```
void Application_Error(object sender, EventArgs e)
{
    Exception exc = Server.GetLastError();

    if (exc.GetType() == typeof(HttpException))
    {
        Response.Redirect("/Error");
    }

}
```

4.10 Testování aplikace

Závěrem proběhne otestování celé aplikaci, a to ručně i za pomoci penetračního nástroje. Cíle testování budou:

- Otestovat autorizační systém pokusy o oprávněný i neoprávněný vstup na stránky či složky s omezeným přístupem
- Vyzkoušet registraci a následné přihlášení uživatele
- Vyzkoušet zámek účtu při opakovaném chybném pokusu o přihlášení
- Zkontrolovat, zda se data skutečně správně ukládají a načítají z databáze
- Otestovat odolnost aplikace vůči závažným útokům z kapitoly 3.2.2



Obrázek 10: *Úvodní stránka aplikace (vlastní zpracování)*

4.10.1 Instalace a nastavení nástroje ZAP

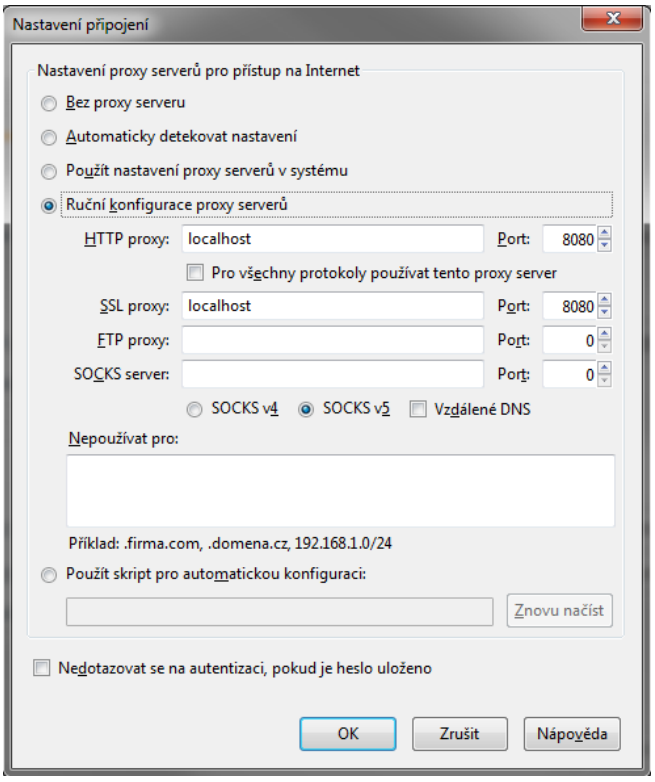
Penetrační nástroj ZAP lze stáhnout na oficiálních stránkách komunity OWASP. Jedná se o jeden z hlavních projektů, kterými se OWASP zabývá. V rámci této práce bude používán ZAP 2.4.3 ve verzi pro Windows. Nutná je také instalace Java Runtime Environment ve verzi 7 a vyšší. Hlavní stránka nástroje je k nahlednutí v příloze A.

Po nainstalování a spuštění tohoto nástroje, je prvním krokem uložení ZAP Root CA SSL certifikátu a jeho následné importování do prohlížeče Firefox, aby mohl ZAP číst šifrovanou komunikaci. Toto lze provést rozbalením menu *Tools* a výběrem položky *Options*. V nově otevřeném okně je pak třeba v levém panelu vybrat možnost *Dynamic SSL*

Certificates. Pokud certifkát nebyl z nějakého důvodu vygenerován, lze tak učinit pomocí tlačítka *Generate*. Poté je třeba jej uložit tlačítkem *Save*. Například na plochu.

V prohlížeči Firefox se certifikát importuje kliknutím na *Možnosti v* rozbalovacím seznamu v pravém horním rohu lišty. Na nově otevřeném panelu je třeba vybrat položku *Rozšířené* a poté *Certifikáty*. Zde stačí kliknout na tlačítko *Certifikáty*, čímž se otevře okno, kde lze ZAP certifikát importovat. Celá operace se potvrdí tlačítkem *OK*.

Nyní je třeba přejít na záložku *Síť* a klepnout na tlačítko *Nastavení připojení…*, což vyvolá okno, které je třeba nastavit podle obrázku č. 11. Uložit změny lze opět pomocí *OK*.



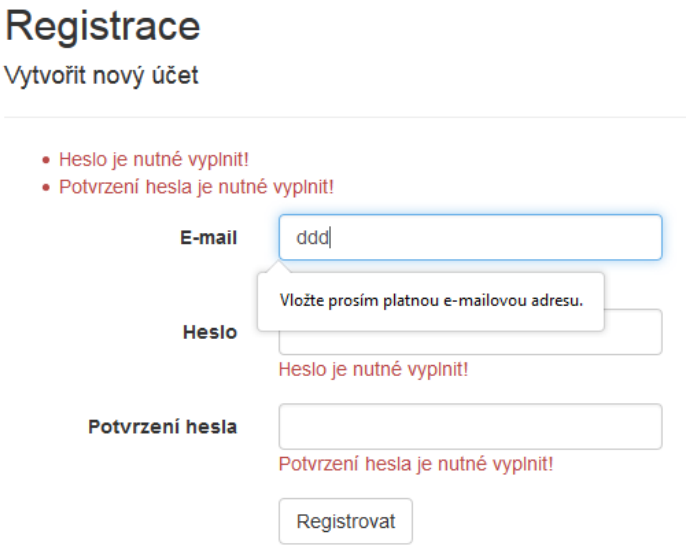
Obrázek 11: *Nastavení proxy připojení pro ZAP (vlastní zpracování)*

Port i adresu jsou uvedeny v nastavení nástroje ZAP kliknutím na položku *Local Proxy*. Zde je možné je i změnit podle vlastní potřeby.

Tímto je vše potřebné nakonfigurováno a ZAP bude od této chvíle logovat veškerou aktivitu v prohlížeči Firefox. Je dobré mít otevřenou pouze testovanou stránku, aby se zbytečně nezahlcoval výpis. Po ukončení testování je také nutné v nastavení připojení prohlížeče opět vypnout *Ruční konfigurace proxy serverů*.

4.10.2 Manuální testování

Aplikaci je třeba spustit v prohlížeči Firefox jeho výběrem v horní liště Visual Studia. Hned po vykreslení úvodní stránky je možné přejít k registraci nového testovacího účtu kliknutím na odkaz *Registrace* (viz obrázek č. 10). K registrování je vyžadováno vyplnění platné e-mailové adresy i hesla, jak je vidět na obrázku č. 12.



Obrázek 12: *Registrace (vlastní zpracování)*

Podrobnosti o požadavcích na zaregistrování nového uživatele se nastavují ve třídě *ApplicationUserManager* v souboru *IdentityConfig.cs*. Nastavení, použité v projektu obsaženém v příloze B, vyžaduje například unikátní e-mail a heslo obsahující alespoň 6 znaků, velká a malá písmena, čísla a speciální znaky.

V rámci této práce bude zaregistrován a nadále používán účet *tester@test.cz* s heslem *Testing12_**. Po úspěšné registraci bude účet rovnou přihlášen do aplikace, což se projeví v pravém horním rohu menu, kde se zviditelní tlačítka *Správa účtu* a *Odhlásit se*. Účet se také uloží do databáze:

	Id	UserName	Email	EmailConfirmed	PasswordHash
1	2011	tester@test.cz	tester@test.cz	0	AGJLVvYJhOaMO+4M1XPm5cefHXqHmEIRENm30p0i7o8RHakx...

Obrázek 13: *Výpis tabulky app_users (vlastní zpracování)*

Z obrázku č. 13 je patrné, že heslo není uloženo v prostém textu, což byla jedna z chyb zmiňovaných v podkapitole 3.2.2.6. Nyní je možné se pokusit o proniknutí do míst s omezeným přístupem.

Jak již zmíňovala podkapitola 3.2.2.7, řada webových aplikací trpí chybami v autorizačním systému, který je často možné zneužít například úpravou URL. Omezený přístup by měl být do složky *Restricted* v adresáři *Modules*, a samozřejmě také i do řídích souborů, jako je *Web.config*. Prvním testem bude tedy úprava URL podle vzoru:

```
https://localhost:44301/Modules/Restricted/Manage.aspx
```

Po potvrzení výběru se stránka skutečně nepřesměruje na stránku *Manage.aspx,* ale naopak na stránku *Login.aspx*, která žádá o přihlášení, aby bylo možné ověřit oprávněnost předchozího požadavku. Přestože se jednalo o neautorizovaný pokus o přístup, aplikace nevyvolala žádnou výjimku, neboť přístup k tomuto zdroji není kategoricky zakázán a tudíž je umožněno uživateli svůj omyl napravit a přihlásit se autorizovaným účtem, čímž by mu přístup již zamítnut nebyl a uživatel by na příslušnou stránku byl přesměrován. Přístup na stránku *Manage.aspx* je možné i kliknutím na tlačítko *Správa účtu*, které nese uživatelské jméno přihlášeného uživatele. Tato akce ale přinese stejný výsledek, jako úprava URL, takže i přístup skrze uživatelské rozhraní je ošetřený. Právě uživatelská rozhraní jsou často slabým místem, viz kapitola 3.2.2.7.

Skrze změnu v URL lze otestovat i přístup k souborům, jako je *Web.config* či *packages.config*. V těchto případech ale webový server vyvolá výjimku a bude zobrazena stránka *Error.aspx*, jejíž tvorbou a napojením se zabývala kapitola 4.9. Stejného výsledku dosáhne i pokus o zobrazení jakéhokoliv adresáře bez specifikování konkrétního souboru či stránky.

Dalším krokem bude otestování části aplikace s omezeným přístupem, nyní již však s autorizovaným účtem. Jelikož uživatelské rozhraní nemá možnost přiřadit uživateli roli *AccountAdministrator*, je pro potřeby testování nutné tuto operaci vykonat manuálně v databázi. Například pomocí příkazu insert nad tabulkou *app_users2Roles*.

Aby se změny promítly, je třeba se odhlásit a znovu přihlásit přes stránku *Login.aspx*. Nyní již přístup na stránku *Manage.aspx* nebude zamítán, ať už k ní bude přistupováno úpravou adresy či přes menu.

Uživatelské rozhraní v příloze B nabízí ve *Správě účtu* pouze možnost změnit heslo, což se provádí na stránce *ManagePassword.aspx*, která je taktéž přístupná pouze roli *AccountAdministrator*. Zde je nutné vyplnit aktuální heslo a dvakrát heslo nové. Aktuálním heslem je *Testing12_**. Jako nové heslo lze zvolit například *Testing2016_**. Po vyplnění

všech polí se celá operace potvrdí tlačítkem *Změnit heslo*, což bude potvrzeno hlášením *Vaše heslo bylo úspěšně změněno*. Přihlásit se pomocí starého heslo již tedy nebude možné a při takovém pokusu bude přihlášení zamítnuto s hlášením *Nepodařilo se přihlásit*.

Na závěr je třeba vyzkoušet zamykání účtu při několikanásobném chybném pokusu o příhlášení. Maximální počet těchto pokusů je 5, což lze opět změnit v souboru *IdentityConfig.cs*. Pro potřeby tohoto testu je na stránce *Login.aspx* tedy nutné vyplnit správně e-mail (*tester@test.cz)* a poté opakovaně chybně vyplňovat heslo. V databázové tabulce *app_users* je v průběhu tohoto procesu vidět, jak se postupně navyšuje hodnota sloupce *AccessFailedCount*. Pokud se zadá správné heslo ještě před dosažením maximálního počtu chybných pokusů o přihlášení, toto počítadlo se vrátí zpět na 0. V opačném případě dojde k zablokování účtu na dobu 5 minut, což lze nastavit v souboru *IdentityConfig.cs*.

Po uzamčení účtu dojde vždy k přesměrování na stránku *Lockout.aspx* a to až do doby, než uplyne nastavená doba čekání. Poté je možné se opět bez problémů přihlásit a nebo znovu zamknout účet 5 chybnými pokusy o přihlášení. Systém zamykání také chrání uživatelské účty před útoky hrubou silou, neboť útočník pokoušející se o uhodnutí hesla by tak měl vždy pouze určitý počet pokusů a pak by musel nějakou dobu čekat, což tuto metodu útoku dělá velmi neefektivní.

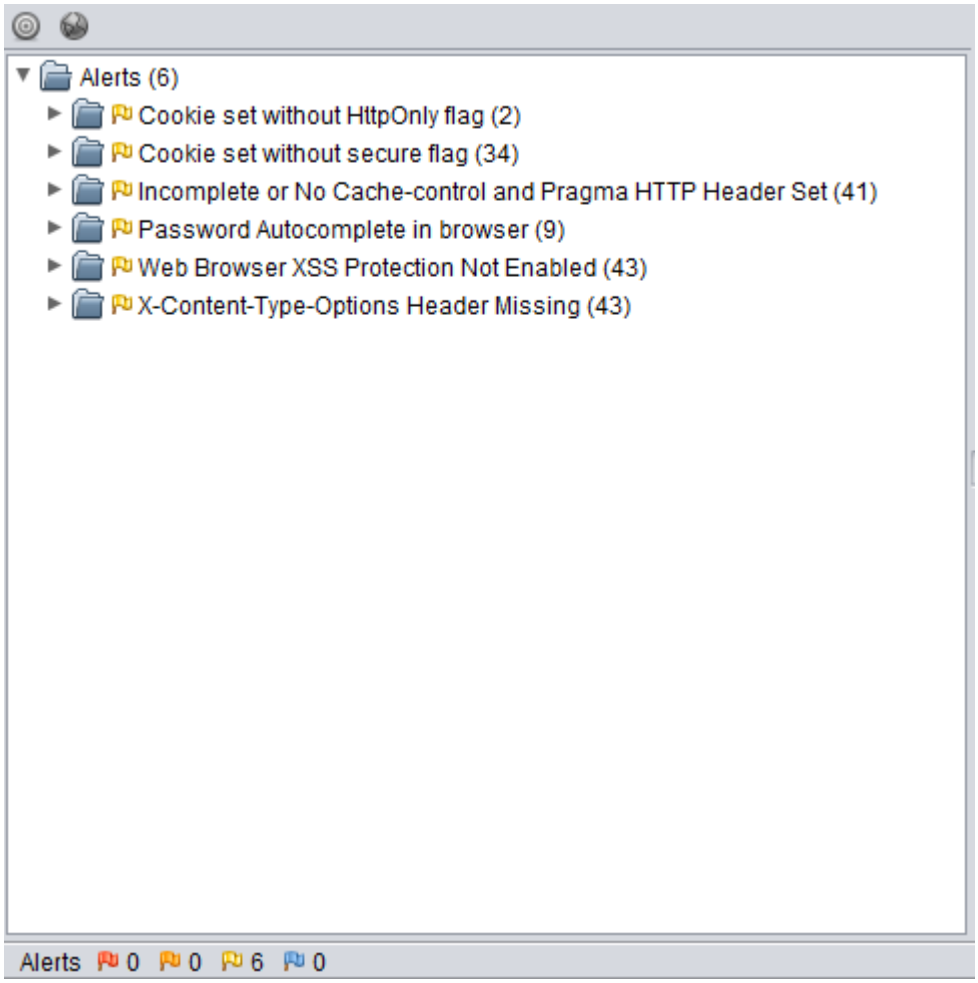
4.10.3 Skenování nástrojem ZAP

Po celou dobu manuálního testování ZAP logoval veškeré dění. To je zobrazené v záložce *History* ve spodní části jeho okna, jak je vidět v příloze A. Samotné skenování lze započít v panelu *Quick Start* výběrem adresy, kterou je v rámci této práce *https://localhost:44301*. Je třeba připomenout, že toto je skutečný útok na danou aplikaci a tak jej, z právního hlediska, lze provést pouze se svolením majitele stránky.

Po stisknutí tlačítka *Attack* spustí ZAP robota typu spider, aby se tak pokusil najít všechny stránky skenované aplikace. Jakmile spider dokončí prohledávání aplikace, započne aktivní skenování, což je v podstatě provádění různých útoků na nalezené stránky. Mezi těmito útoky je například injektování, XSS a další, jimiž se zabývala kapitola 3.2.2 a jejichž znalost je důležitá pro pochopení penetračního testování. Řada útočných metod také zneužívá zranitelností View statu, skrytých polí a dalších součástí State managementu

v ASP.NET, kterým se věnovala kapitola 3.1.1.1. Celý proces skenování může trvat i několik minut.

V příloze A je vidět, že v průběhu aktivního skenování bylo provedeno 5160 požadavků a nalezeno bylo 6 varování s nízkou důležitostí. Blíže je toto vidět na záložce *Alerts*.



Obrázek 14: *Záložka Alerts (vlastní zpracování)*

Varování této úrovně nejsou příliš naléhavá a tudíž je není třeba akutně řešit. Bližší informace jsou dostupné po rozkliknutí jednotlivých položek tak, jak jsou vidět na obrázku č. 14. V pravé části okna nabídne ZAP k nahlednutí HTTP odpovědi jednotlivých varování a především URL, která je s nimi spjatá. Zde je také k dispozici popis chyb v části *Description* a také popis možných řešení v části *Solution*.

5 Závěr

Hlavním cílem práce bylo vytvoření zabezpečeného základního jádra webové aplikace ve frameworku Web Forms, které mělo disponovat integrovaným systémem řízení a správy uživatelských účtů. Dále měla být funkčnost a flexibilita tohoto systému názorně demonstrována. Dílčím cílem pak byl popis Web Forms v obecném kontextu tvorby webových aplikací a jeho návazností na související části .NET frameworku.

Dílčí cíl se podařilo naplnit již na začátku teoretické části této práce, kde byly rozepsány části .NET frameworku, a to včetně Web Forms. Takto získané znalosti se prokázaly, jako velmi užitečné, v následující praktické části, kde bylo dosaženo i hlavního cíle. Zde proběhl jak návrh celého jádra, tak i jeho implementace. Závěrem práce bylo celkové otestování aplikace jak manuálně, tak i za pomoci automatizovaného nástroje, čímž se prokázala jistá míra bezpečnosti daného řešení a jeho odolnost proti útokům uvedeným v kapitole 3.2.2.

Tuto práci by bylo bezprostředně možné rozvinout několika směry. Zaprvé by bylo vhodné zavést logování výjimek (jež aplikace a webový server mohou vyvolat) do databáze. Prvním krokem byla kapitola 4.9, kde byly položeny základy vstříc tomuto řešení. Začít by se dalo rozšířením události *Application_Error* v souboru Global.asax.

Druhým krokem k vylepšení celé aplikace by bylo načítání často používaných dat do paměti serveru. Jednat by se mohlo například o seznam uživatelů či uživatelských rolí. Takto by se zrychlila odezva serveru, neboť by nebylo třeba neustále spouštět dotazy nad databází.

Třetím vhodným zlepšením se jeví zdokonalení registračního a především autentizačního procesu. V tomto směru by stačilo pouze zprovoznit další součásti ASP.NET Identity, jako je dvoufaktorová autentizace či přihlašování skrze externí poskytovatele autentizace, jako jsou Facebook, Twitter či Google.

Posledním krokem ve vývoji vyšší verze aplikace by mělo být provedení hloubkového code review a analýzy potenciálních zranitelností celé aplikace. K tomu by měly napomoct znalosti získané v kapitole 3.2.1, která se touto problematikou zabývá.

Přínosem celé práce je obecná analýza rizik vývoje webových aplikací z hlediska bezpečnosti a především pak definování postupu pro vývoj zabezpečené webové aplikace na platformě ASP.NET.

6 Seznam použitých zdrojů

Knižní publikace

DORANS, Barry. 2010. *Beginning ASP.NET Security*. Wrox. ISBN 978-0-470-74365-2.

GAYLORD, Jason, Christian WENZ, Pranav RASTOGI a Scott HANSELMAN. 2013. *Professional ASP.NET 4.5 in C# and VB*. 1. vydání. Wrox. ISBN 978-1-118-31182-0.

HOWARD, Michael, David LEBLANC a John VIEGA. 2009. *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. 1. vydání. McGraw-Hill Education. ISBN 978-0-071-62675-0.

MEUCCI, Matteo a Andrew MULLER. 2015. *Testing Guide 4.0 - Release*. The OWASP Foundation. 4. edice. Dostupné také z: https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf

Webové stránky a příspěvky

ASP.NET Overview. b.r. *Microsoft Developer Network* [online]. [cit. 2016-01-03].

ASP.NET State Management Overview. b.r. *Microsoft Developer Network* [online]. [cit. 2016-01-04].

ASP.NET State Management Recommendations. b.r. *Microsoft Developer Network* [online]. [cit. 2016-01-04].

Claims-Based Identity for Windows. 2011. *Microsoft* [online]. [cit. 2016-02-27]. Dostupné z: http://download.microsoft.com/download/7/D/0/7D0B5166-6A8A-418A-ADDD-95EE9B046994/Claims-Based%20Identity%20for%20Windows.pdf

Common Language Runtime. b.r. *Microsoft Developer Network* [online]. [cit. 2016-01-05]. Dostupné z: https://msdn.microsoft.com/en-us/library/8bs2ecf4

Cutting Edge : A First Look at ASP.NET Identity. 2015. *Microsoft Developer Network* [online]. [cit. 2015-12-15]. Dostupné z: https://msdn.microsoft.com/en-us/magazine/dn605872.aspx

Introduction to ASP.NET Identity. 2013. *ASP.NET* [online]. [cit. 2015-12-05]. Dostupné z: http://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity

Introduction to ASP.NET Web Forms. b.r. *ASP.NET* [online]. [cit. 2016-01-07]. Dostupné z: http://www.asp.net/web-forms/what-is-web-forms

Open Web Interface for .NET. b.r. *Open Web Interface for .NET* [online]. [cit. 2016-02-27]. Dostupné z: http://owin.org/#about

Overview of Custom Storage Providers for ASP.NET Identity. 2014. *ASP.NET* [online]. [cit. 2016-02-29]. Dostupné z: http://www.asp.net/identity/overview/extensibility/overview-of-custom-storage-providers-for-aspnet-identity

OWASP Risk Rating Methodology. b.r. *OWASP* [online]. [cit. 2016-01-29]. Dostupné z: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

OWASP Top 10 - 2013. 2013. *OWASP* [online]. [cit. 2016-01-27]. Dostupné z: https://www.owasp.org/images/f/f3/OWASP_Top_10_-_2013_Final_-_Czech_V1.1.pdf

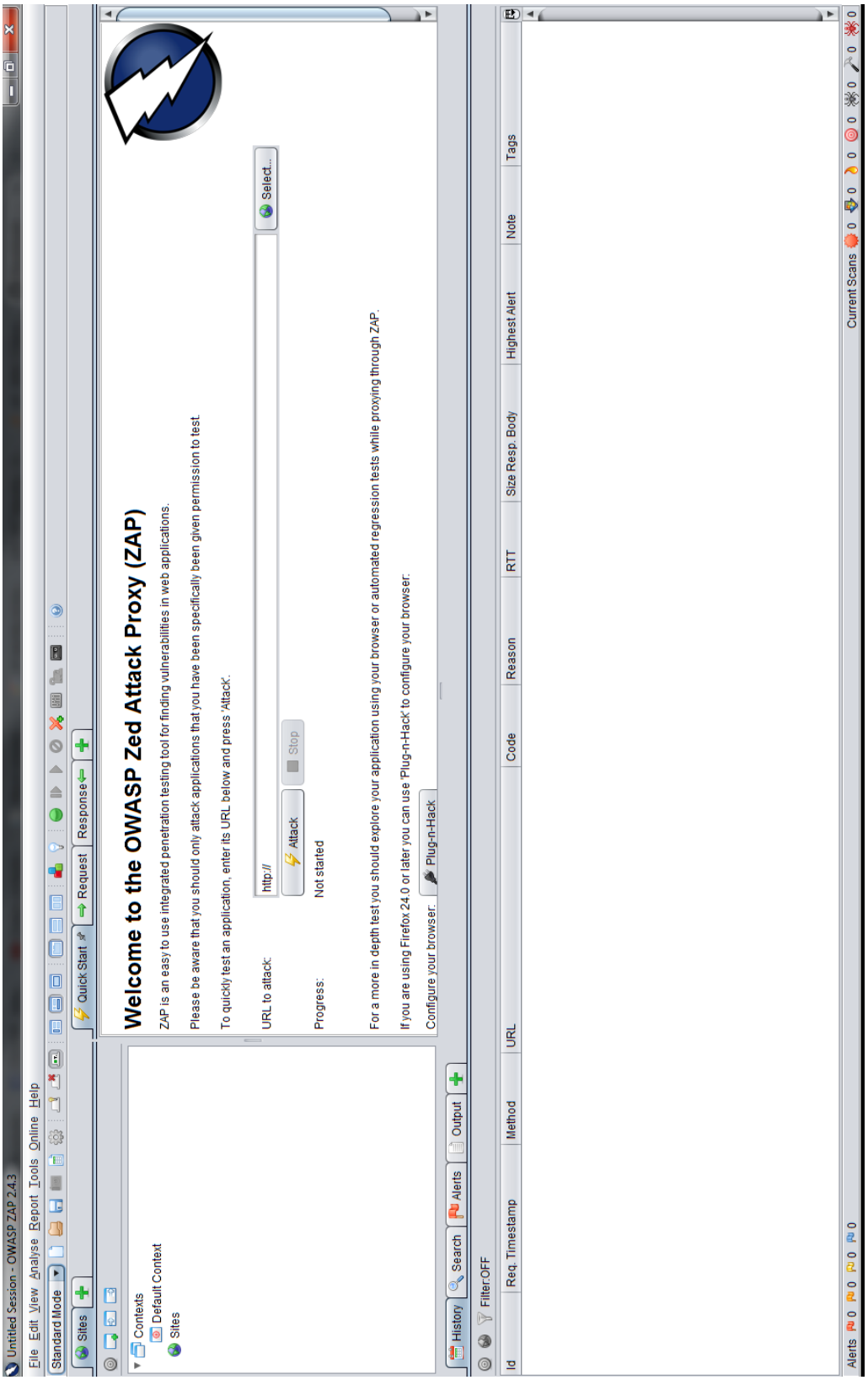
OWASP ZAP 2.4. b.r. *OWASP* [online]. [cit. 2016-03-08]. Dostupné z: https://github.com/zaproxy/zaproxy/releases/download/2.4.0/ZAPGettingStartedGuide-2.4.pdf

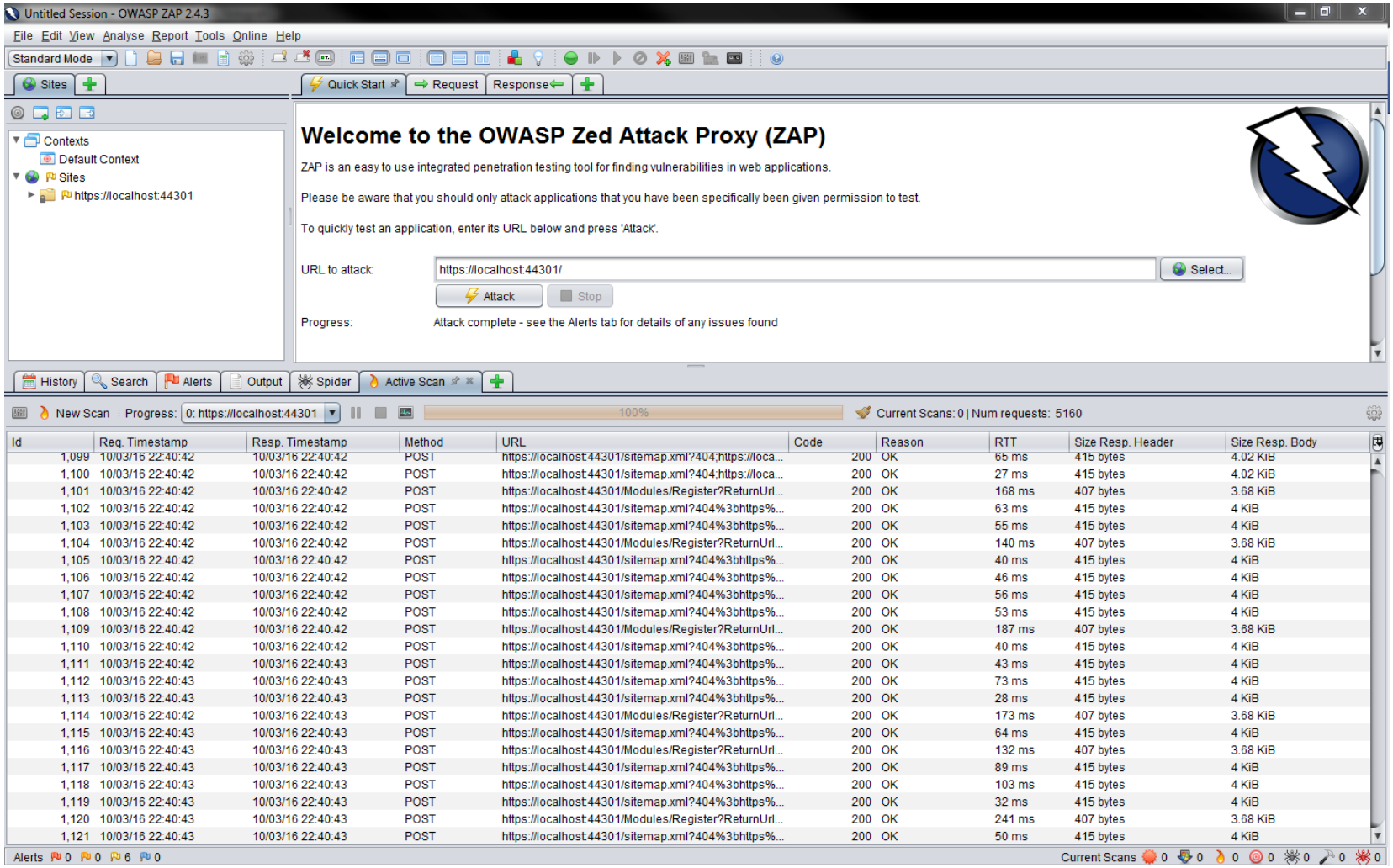
Preventing Cross-Site Request Forgery (CSRF) Attacks in ASP.NET Web API. 2012. *ASP.NET* [online]. [cit. 2016-02-15]. Dostupné z: http://www.asp.net/web-api/overview/security/preventing-cross-site-request-forgery-csrf-attacks

SQL Server 2014 Express LocalDB. b.r. *Microsoft Developer Network* [online]. [cit. 2016-02-29]. Dostupné z: https://msdn.microsoft.com/en-us/library/hh510202%28v=sql.120%29.aspx

7 Přílohy

A. Penetrační nástroj ZAP




Untitled Session - OWASP ZAP 2.4.3
File Edit View Analyse Report Tools Online Help
Standard Mode
Sites
Quick Start
Request
Response
Contexts
Default Context
Sites
https://localhost:44301
Welcome to the OWASP Zed Attack Proxy (ZAP)
ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.
Please be aware that you should only attack applications that you have been specifically been given permission to test.
To quickly test an application, enter its URL below and press 'Attack'.
URL to attack:
https://localhost:44301/
Select...
Attack
Stop
Progress:
Attack complete - see the Alerts tab for details of any issues found
History
Search
Alerts
Output
Spider
Active Scan
New Scan
Progress:
0: https://localhost:44301
100%
Current Scans: 0 | Num requests: 5160
Id
Req. Timestamp
Resp. Timestamp
Method
URL
Code
Reason
RTT
Size Resp. Header
Size Resp. Body
1,099 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404;https://loca... 200 OK 65 ms 415 bytes 4.02 KiB
1,100 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404;https://loca... 200 OK 27 ms 415 bytes 4.02 KiB
1,101 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/Modules/Register?ReturnUrl... 200 OK 168 ms 407 bytes 3.68 KiB
1,102 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 63 ms 415 bytes 4 KiB
1,103 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 55 ms 415 bytes 4 KiB
1,104 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/Modules/Register?ReturnUrl... 200 OK 140 ms 407 bytes 3.68 KiB
1,105 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 40 ms 415 bytes 4 KiB
1,106 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 46 ms 415 bytes 4 KiB
1,107 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 56 ms 415 bytes 4 KiB
1,108 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 53 ms 415 bytes 4 KiB
1,109 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/Modules/Register?ReturnUrl... 200 OK 187 ms 407 bytes 3.68 KiB
1,110 10/03/16 22:40:42 10/03/16 22:40:42 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 40 ms 415 bytes 4 KiB
1,111 10/03/16 22:40:42 10/03/16 22:40:43 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 43 ms 415 bytes 4 KiB
1,112 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 73 ms 415 bytes 4 KiB
1,113 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 28 ms 415 bytes 4 KiB
1,114 10/03/16 22:40:42 10/03/16 22:40:43 POST https://localhost:44301/Modules/Register?ReturnUrl... 200 OK 173 ms 407 bytes 3.68 KiB
1,115 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 64 ms 415 bytes 4 KiB
1,116 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/Modules/Register?ReturnUrl... 200 OK 132 ms 407 bytes 3.68 KiB
1,117 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 89 ms 415 bytes 4 KiB
1,118 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 103 ms 415 bytes 4 KiB
1,119 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 32 ms 415 bytes 4 KiB
1,120 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/Modules/Register?ReturnUrl... 200 OK 241 ms 407 bytes 3.68 KiB
1,121 10/03/16 22:40:43 10/03/16 22:40:43 POST https://localhost:44301/sitemap.xml?404%3bhttps%... 200 OK 50 ms 415 bytes 4 KiB
Alerts 0 0 6 0
Current Scans 0 0 0 0 0 0 0

B. Obsah CD

- /bp_core.sql
  - SQL skript na založení databáze.
- /BpRoles
  - Složka obsahující kompletní projekt zpracovávaný v rámci praktické části bakalářské práce.