Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky

# Diplomová práce

2015 Marek Šmolík

# Vysoká škola ekonomická v Praze

**Fakulta informatiky a statistiky**

**Katedra informačních technologií**


Studijní program: Aplikovaná informatika
Obor: Podniková informatika

# Bezpečnost webových aplikací

**DIPLOMOVÁ PRÁCE**


Student : **Bc. Marek Šmolík**
Vedoucí diplomové práce : **doc. Ing. Alena Buchalcevová, Ph.D.**
Oponent diplomové práce : **Ing. Jan Mészáros**


**ROK : 2015**

**Prohlášení**

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne .....

podpis

**Poděkování**

Rád bych poděkoval paní doc. Ing. Aleně Buchalcevové, Ph.D. za trpělivost a pomoc při formování mých představ při vedení této práce.

# Abstrakt

Práce se zaměřuje na oblast bezpečnosti webových aplikací s důrazem na platformu ASP.NET MVC. Primárním cílem je vytvoření ucelené metodiky pro systematické ověření bezpečnosti webové aplikace, která popisuje konkrétní implementační detaily určené vývojářům a testerům. Metodika čerpá kategorizaci, technické detaily i jednotlivé zranitelnosti z více zdrojů a doplňuje je o konkrétní příklady. Práce je přínosná pro vývojáře a testery zmíněné platformy především proto, že přináší podrobné informace, které dostupné metodiky nezahrnují.

Klíčová slova: bezpečnost, web, ASP.NET MVC, metodika

# Abstract

This thesis is focused on a web application security subject with an emphasis on ASP.NET MVC platform. Its primary objective is to create a comprehensive methodology for systematic verification of web application security, which contains specific implementation details intended for developers and testers. The methodology takes categorization, technical details and individual vulnerabilities from multiple sources and enhances them with specific examples. Work is valuable for developers and testers of mentioned platform mainly because it provides detailed information not included in available methodologies.



Keywords: security, web, ASP.NET MVC, methodology

# Obsah

# 1 . Úvod

Webové aplikace za dobu své existence změnily celý dnešní svět, způsob jakým lidé komunikují, pracují, vzdělávají se nebo spravují své finance. Nesporný dopad této technologie je důkazem obrovského pokroku. Pokroku, který napříč všemi technologickými i ostatními vrstvami této platformy způsobuje také velké problémy. Jeho tempo jen obtížně sledují standardy, protokoly, technologické komponenty, ale také lidé za webem stojící. Jak uvádí [4], mnoho z těchto klíčových částí systému je užíváno daleko za jejich původně zamýšleným účelem, což s sebou přináší také velmi rozsáhlý prostor pro bezpečnostní rizika.

Tato kapitola vymezuje základní téma práce, zužuje sledovanou oblast a nastiňuje cíle práce.

## 1.1 Vymezení tématu, důvod výběru tématu

Práce se zaměřuje na bezpečnost moderních webových aplikací s akcentem na technologickou platformu .NET, především aktuálně velmi prosazovanou technologii ASP.NET MVC (10/2014). Hlavním záměrem je ucelený popis sady bezpečnostních problémů dnešních webových aplikací. Jednotlivé problémy jsou rozděleny do kategorií, zařazeny do úrovní podle kritičnosti jejich ošetření (míry rizika), popsány jejich principy a vlastnosti ekonomické i technické povahy.

Oblast bezpečnosti aplikací obecně je velmi široká a konkrétních problémů je rozsáhlé spektrum. Omezení na webové aplikace zahrnuje mj. oblast webových služeb, integračních řešení nad službami (tzv. SOA, „Service oriented architecture“, „Architektura orientovaná na služby“, zdroj [3]), mashupové aplikace a mnoho dalších. Práce se věnuje zúžené oblasti zranitelností, které se týkají běžných webových aplikací – blíže je specifikuje kapitola 3.

Ačkoliv jsou vybrány bezpečnostní problémy, které jsou typicky aplikovatelné na webové aplikace vytvořené na výše uvedené technologické platformě, výstupy práce je možné využít prakticky na libovolnou webovou aplikaci.

Důvodem tohoto zaměření je fakt, že kromě standardů vycházejících z projektů OWASP („Open Web Application security project“, „otevřený projekt bezpečnosti webových aplikací“, autor, zdroj [29]) autor nenalezl dostupnou moderní obecnou metodiku v této oblasti. Navíc samotný OWASP ASVS ustoupil ve verzi 2013/2014 („Application Security Verification Standard 2014“, „standard ověření bezpečnosti aplikací“, autor, zdroj [17]) od konkrétních postupů, jak zranitelnosti ověřit nebo ošetřit a věnuje pozornost převážně principům a výčtu problémů k ověření. Pro testery a vývojáře (resp. společnosti) je následně velmi nákladné vyhledávat jednotlivé postupy ověření bezpečnosti. Více se problematice dostupných metodik věnuje kapitola (3.1.2).

## 1.2 Charakteristika současného stavu oblasti

Hlavním problémem bezpečnosti současných webových aplikací je podle [4], [17] nebo [37] mj. rozšířená sada relativně snadno použitelných nástrojů, platforem a knihoven pro jejich vývoj. To má za následek, že vývojáři jsou často odstiňováni od nižších vrstev webu a nejsou tedy zcela seznámeni s jeho fungováním. Tato neznalost následně může vést k vytváření funkčních, nicméně ne bezpečných aplikací.

V oblasti bezpečnosti webových aplikací se pohybuje velké množství profesionálů, institucí i organizací zaměřených na vzdělávání. Povědomí o základních rizicích postupem času stoupá, jak

dokládá vývoj nejčastějších bezpečnostních chyb aplikací, zdroje [37], [9], [4] a další. Stejné zdroje však zároveň naznačují, že část dobře známých chyb se stále opakuje i v nejnovějších webových aplikacích.

V jiných odvětvích, jako například stavebnictví, jsou pro tyto případy vydávány metodiky a standardy sjednocující pohled na problematiku, zvyšující povědomí a umožňující vyžadování kvality dle standardů zákazníkem. V oblasti IS/IT („informační systémy / informační technologie“, autor) je metodik a standardů velké množství, zaměřují se na řízení IT projektů, IT Governance a další. Metodiky pro řízení, hodnocení a zvyšování **bezpečnosti** těchto systémů se dají shrnout do tří skupin: obecné odborné, specializované oborové a ostatní. Obecné odborné se zaměřují na odborníky, vývojáře a testery, stejně jako tato práce. Specializované oborové metodiky se věnují úzkému, typicky státem regulovanému oboru. Ostatní metodiky nejsou pro použití vývojáři a testery ani vhodné, ani určené.

Hlavním nedostatkem dostupných metodik je nízká použitelnost pro vývojáře, testery a specialisty. V případě první skupiny metodik je to pak přílišná nekonkrétnost (ASVS 2014). Jedinou metodikou, která v těchto kritériích obstojí je projekt OWASP TOP 10 ([37]). Podrobněji se dostupným metodikám a jejich popisu věnuje kapitola 3.1.2.

## 1.3 Cíle a způsob jejich dosažení

Primárním cílem práce je návrh metodiky pro ověření bezpečnosti webové aplikace. Metodika je vhodná především pro penetrační testery a vývojáře a je obecně použitelná pro webové aplikace jakékoliv platformy (například ASP.NET, PHP („PHP Hypertext Preprocessor“), Java). Příklady jsou uváděny na technologii ASP.NET MVC, pro kterou jsou také zranitelnosti vybírány. Druhotným cílem práce je souhrn maximálního možného počtu zranitelností z více zdrojů, jejich kategorizace a rozdělení do úrovní metodiky.

Metodika je sestavena z několika primárních zdrojů, ze kterých přebírá nebo odvozuje kategorizaci bezpečnostních rizik, popis principů a způsobů provedení útoků, vybírá jednotlivá rizika relevantní pro cílenou platformu a na základě kritérií je pak řadí do úrovní. Výčtem primárních zdrojů a popisem postupu jejich slučování se zabývá kapitola 3.2.

## 1.4 Přínosy práce

Práce je koncipována jako metodika ověření bezpečnosti se specifikací úrovní a konkrétních zranitelností jim odpovídajících. Metodika poskytuje konkrétní kroky, jak každý popsaný útok provést (resp. způsob jak otestovat aplikaci), jak se proti němu bránit, ukázky zdrojového kódu, popis rolí v útoku (popis útočníka, nutných dovedností), popis severity a dopadů úspěšného útoku, ekonomické aspekty útoku aj.

Metodika je kompatibilní se standardy v oblasti, jako je OWASP ASVS, TOP 10 a dalšími a to díky uvedeným mapováním jednotlivých úrovní/zranitelností mezi těmito dokumenty (kapitola 8.1).

Mezi výstupy práce patří také popis teoretických východisek webové bezpečnosti, přehled aktuálních bezpečnostních rizik, praktická sada příkladů (kolekce zranitelných webových aplikací a jejich oprav), metodický postup, jak ověřit úroveň bezpečnosti aplikace s důrazem na ekonomické aspekty zranitelností nebo očekávané formální způsoby zpracování testů (například očekávané výstupy), případně jejich využitelnost při auditu IS.

Hlavním přínosem práce je tedy popsaná metodika k ověření aplikace a konkrétní popis zranitelností.

## 1.5 Předpoklady a určení práce

Jak bylo uvedeno, metodika je vhodná především pro penetrační testery a vývojáře, dále také jako referenční materiál pro ekonomicky zaměřené subjekty ve společnosti (manažery, obchodníky), které pomocí ní mohou popsat úroveň zabezpečení aplikací.

## 1.6 Rešerše zdrojů

Problematice bezpečnosti se věnuje široké spektrum zdrojů. Vymezené oblasti bezpečnosti webových aplikací je pak věnována jejich značná část. Tato kapitola jmenuje vybrané z nich.

### 1.6.1 Knihy, periodika a odborná literatura

Kniha *„The web application hacker's handbook: finding and exploiting security flaws“* autorů D. Stuttarda a M. Pinta (zdroj [4]) se věnuje celému rozsahu webové bezpečnosti, od zranitelností zaměřených na jednotlivé slabé body platformy, až po metodologii útoků a nástroje vhodné pro penetrační testery. Je zaměřena obecně na webové aplikace libovolné platformy.

Kniha *„XSS: Cross-Site Scripting v praxi : o reálných zranitelnostech ve virtuálním světě“* R. Kümmela (zdroj [2]) je specificky zaměřena na konkrétní oblast webové bezpečnosti (popsané v kapitole 5.1). Jejím hlavním přínosem je velmi komplexní a hluboká analýza dané oblasti a velké portfolio ukázkových i reálných příkladů z praxe.

Kniha *„Web security testing cookbook: systematic techniques to find problems fast“* autorů P. Hope a B. Walthera je věnována konkrétnímu a ucelenému popisu testování webových aplikací, nástrojům usnadňujícím vyhledání zranitelností a také metodice provádění testů.

### 1.6.2 Akademické práce

Bakalářská práce *„Bezpečnost ASP.NET aplikací“* autora M. Pavelka (zdroj [40]) se věnuje principům a architektuře autentizačních a autorizačních mechanizmů platformy a vybraným bezpečnostním aspektům a zranitelnostem. Zranitelnosti jsou popsány včetně příkladů a popsané principy jsou podloženy ukázkovou aplikací. Práce je zaměřena spíše obecně, na širší publikum dané platformy a pokrývá obecné základy její bezpečnosti.

Bakalářská práce *„Aplikace e-learningu a bezpečnost dat“* autora J. Menčíka (zdroj [39]) se věnuje bezpečnosti specifického typu aplikací a také konkrétním vybraným implementacím tohoto typu softwaru. Práce popisuje sadu zranitelností podle metodiky OWASP TOP 10 (zdroj [37]). Jednotlivě je popisuje včetně příkladů a základního odůvodnění, a následně je aplikuje při testování vybraných produktů. Práce je orientována především na testování a nikoliv vývoj nebo ochranu aplikací.

Diplomová práce *„Testování bezpečnosti webových aplikací“* autora M. Eliáše (zdroj [38]) se podrobně věnuje webové architektuře a platformě, popisuje teoreticky nejčastější zranitelnosti, jejich klasifikaci a metody testování.

Bakalářská práce *„Hodnocení bezpečnosti PHP aplikace podle standardu OWASP ASVS“* autora J. Sůvy (zdroj [41]) se věnuje metodikám a postupům penetračního testování a podrobně popisuje obsah vybraného standardu bezpečnosti (zdroj [17]). Autor provádí také praktické

prověření a vyhodnocení bezpečnosti vybrané aplikace podle standardu.

### 1.6.3 Taxonomie, metodiky, seznamy a ostatní

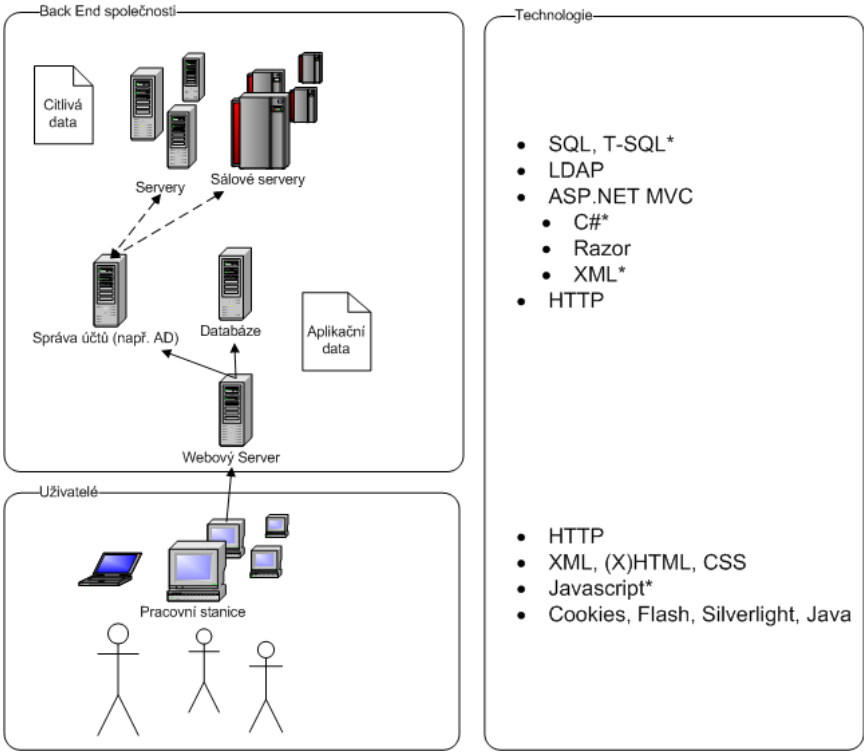
Primárním zdrojem informací v oblasti jsou taxonomie a seznamy zranitelností. Podrobně se popisu těchto zdrojů věnují kapitoly 3.1.2 a 3.2.

# 2 . Teoretická východiska webové bezpečnosti

Tato kapitola se věnuje teoretickým východiskům webové bezpečnosti, popisu hlavních úskalí webové architektury, jejich odůvodnění, popisu hlavních zdrojů bezpečnostních rizik a základních způsobů obrany.

## 2.1 Komplexnost platformy

Platforma současných webových aplikací je velmi komplexní – tvorba aplikací na ní postavených vyžaduje znalost velkého množství technologií a standardů. Tento složitý systém poskytuje velmi mnoho prostoru nejen pro škodlivé chování útočníků, ale především pro chyby tvůrců aplikací. Níže uvedený obrázek 2.1 ilustruje zjednodušenou hardwarovou strukturu běžné webové aplikace a jí odpovídající technologické vrstvy – typicky se může jednat o vnitropodnikový systém například pro správu úkolů nebo docházky, intranet.




*Ilustrace 2.1: Zjednodušené schéma typické webové aplikace ASP.NET MVC, zdroj: autor*

Obrázek obsahuje několik zásadních informací:

- Na straně společnosti (provozovatele aplikace) je aplikace „uvnitř“ infrastruktury (zdroj [4]).
  - Webová aplikace je provozována až za typickou „linií síťové obrany“ společnosti.
  - Pokud útočník bez webové aplikace hodlal proniknout k vyobrazeným citlivým datům, musel napadat přímo síťovou infrastrukturu, servery, síťové prvky (routery, firewally) a chyby v jejich implementaci. Tento velmi náročný úkol mohli správci sítě prakticky zcela znemožnit.
  - Pokud však společnost vystaví webovou aplikaci v síti Internet a aplikace nebude

zabezpečena například proti útoku typu SQL („structured query language“, autor) Injection (viz kapitola 4.1), stačí útočníkovi pozměnit odeslaná data v textovém poli aplikace a nežádoucí akce může bez problémů cestovat napříč zabezpečenými vrstvami sítí až k vnitřnímu serveru a provést téměř libovolnou akci.

- Infrastrukturní zabezpečení nemá prakticky možnost takovému útoku zabránit.

- Na straně klienta je aplikace spouštěna ve webovém prohlížeči.
  - Od dob počátků webu, kdy webový prohlížeč pouze zobrazil textovou informaci obdrženou ze serveru uživateli, se prohlížeče staly samostatnými platformami pro provoz nejrůznějších aplikací.
  - Prohlížeče umožňují zobrazovat graficky propracovaný obsah, odesílat formulářová data nebo soubory na server, spouštět doplňky třetích stran pro rozšíření funkcí a především spouštět klientské skripty v jazyce Javascript.
  - Tato sada nástrojů umožňuje na klientském stroji provádět téměř libovolné operace. Příkladem mohou být nové aplikace využívající možností standardu HTML 5 („Hypertext markup language“, [14]), jako technologie WebGL („web graphics library“, zdroj [14]) pro vykreslování 3D grafického obsahu.
- Technologicky se jedná o velmi rozmanitý systém, zahrnující mimo jiné:
  - Velmi širokou škálu technologií jak na straně serveru tak klienta snižující možnost snadno a spolehlivě zabezpečit celou aplikaci, především z důvodů kladení vysokých nároků na znalosti vývojářů. Technologie uvedené v ilustraci s hvězdičkou jsou typicky doprovázeny množstvím knihoven a frameworků, případně dalšími rozšiřujícími technologiemi. Všechna tato dodaná abstrakce přináší až na výjimky větší komplexitu.
  - Serverová strana obrázku naznačuje použití databázových dotazovacích jazyků (SQL, T-SQL), typicky doplněných o nástroje ORM („Objektově Relačního Mapování“, autor). LDAP („Lightweight Directory Access Protocol“, „protokol pro přístup na adresářový server“, autor) a HTTP („HyperText Transfer Protocol“, autor) zastupují kategorii protokolů, kterých je od nejnižších síťových vrstev, až po nejvyšší integrační vrstvy mnoho. Nad těmito vrstvami jsou typicky využity technologie z frameworku .NET (ASP.NET MVC, Razor), jazyk C# a další. Opět s výrazným zastoupením knihoven a nástrojů, které rozšiřují pole působnosti potencionálního útočníka.
  - Výše zmíněná přeměna prohlížečů v plnohodnotná prostředí je doprovázena rozmachem jazyka Javascript. Stále širší možnosti otevřené tomuto jazyku na klientském stroji a množství dostupných knihoven a nástrojů vytváří řadu dalších faktorů bezpečnosti webu.

### 2.1.1 Popis hlavních platforem a směrů

Platforem a programovacích jazyků pro tvorbu webových aplikací je velké množství, jejich úplný výčet přesahuje potřeby této práce. Z pohledu bezpečnosti lze předpokládat, že aplikace na všech běžných platformách může být napsána bezpečně. Krom ekonomických faktorů, jako jsou náklady rostoucí s bezpečností, čas nebo znalosti, které společnost musí nakoupit v podobě pracovníků, jsou důležité faktory zavedené používaným programovacím jazykem a dodanými

knihovnami – tzv. „frameworkem“. Typicky se jedná o vlastnosti jazyka z pohledu typové kontroly a předcházení „překlepům“ (silně typové a slabě typové jazyky z angl. „strongly“ a „weakly typed“), možnosti předcházení problematickým strukturám, jako je stavba SQL dotazu textově (například za použití silně typových výrazů namísto textu) a používání ukazatelů („pointerů“), nebo vlastnosti prostředí a knihoven, jako je možnost využít vestavěné a prověřené mechanizmy pro autorizaci, přístup k databázi, bezpečnou práci s pamětí apod.

Níže uvedená tabulka na vybraných příkladech hodnotí platformy z pohledu bezpečnosti. Jedná se o souhrnné subjektivní hodnocení jazyka, prostředí, knihoven, architektury a dalších vlivů autorem práce.

| **Platforma** | **Míra abstrakce** | **Potenciální chybovost v kódu** | **Potenciální chybovost v konceptu** | **Potenciál „syndromu černé skříňky“** |
|---|---|---|---|---|
| PHP (základní) | Nízká | Vysoká | Vysoká | Nízký |
| PHP (s frameworkem) | Nízká – střední | Vysoká | Střední | Nízký – střední |
| Java (základní JSP, servlet) | Nízká – střední | Nízká – střední | Střední | Nízký – střední |
| Java (JEE, EJB, Spring) | Střední - vysoká | Nízká | Nízká – střední | Vysoký |
| ASP.NET MVC | Střední | Nízká – střední | Nízká – střední | Nízký – střední |
| ASP.NET WebForms | Vysoká | Nízká | Střední | Vysoký |

*Tabulka 2.1: Platformy z pohledu bezpečnosti, zdroj autor*

Platforma PHP je výše uvedena ve své základní podobě a také v podobě s použitím frameworku (například „Nette“). Obdobně je rozdělena platforma Java na základní minimalistické použití JSP („java server pages“) a využití frameworků a plnohodnotných možností JEE („java enterprise edition“). ASP.NET je rozdělen na hlavní dva proudy, které jsou obdobné využití a nevyužití frameworku – MVC větev s nižší mírou abstrakce a WebForms.

Následující seznam popisuje jednotlivá kritéria hodnocení a jejich význam z pohledu bezpečnosti:

- Míra abstrakce,
  - hodnotí míru odstínění programátora od základních webových vrstev, jako protokol HTTP nebo jazyk HTML,
  - vyšší míra abstrakce může snižovat nároky na znalosti a náklady na implementaci, naopak nižší míra abstrakce může usnadnit řešení komplexních problémů skrytých pod abstrakčními vrstvami,
  - z pohledu bezpečnosti se jedná o přesun části zodpovědnosti za bezpečnou činnost aplikace z vývojáře na framework (v pozitivním i negativním smyslu),
- potenciální chybovost v kódu,
  - hodnotí snadnost vytvoření bezpečnostní chyby vývojářem v aplikaci,

  - typicky vybírá nejsnazší cestu vývojáře k získání vstupu od uživatele a jeho použití v rámci databázového dotazu (potenciální podsunutí SQL a další),
  - z pohledu bezpečnosti vysoký potenciál značí, že vývojář může (a často bude) využívat nebezpečných konstrukcí nebo nezabezpečených vstupů a platforma (jazyk, prostředí, framework) jej koncepcí, architekturou ani kontrolou nesměruje k bezpečnější variantě,
  - nízký potenciál naopak značí, že je vývojář směřován k použití „best-practice“ a většinou není nucen k vymýšlení vlastních „cest“ a řešení,
- potenciální chybovost v konceptu,
  - hodnotí snadnost vytvoření konceptu, designu nebo architektury, která není bezpečná,
  - například mixování vrstev v aplikaci a tím porušování bezpečnostních zásad, jako práce s databází na úrovni prezentační logiky,
  - z bezpečnostního hlediska chybné koncepty vedou obvykle k nepřehlednosti, obtížné správě a kontrole kódu a roztříštění bezpečnostních a validačních pravidel, což může mít za následek celkové snížení bezpečnosti,
- potenciál „syndromu černé skříňky“,
  - hodnotí, jak snadno daná platforma umožňuje vývojáři porozumět nižším vrstvám implementace,
  - vývojář je do jisté míry motivován poznat nižší úroveň tak, aby dokázal v případě nutnosti na takové úrovni uvažovat (při ladění chyby, problémech s kompatibilitou),
  - klade-li platforma v tomto směru „velký odpor“, vývojář může začít na implementaci slepě spoléhat, z implementace se tedy stane tzv. „černá skříňka“ (z angl. „black box“),
  - pokud vývojář nerozumí chování takové komponenty, může ji chybně využívat, konfigurovat a zavádět tak další bezpečnostní problémy.

Vzhledem k faktu, že webové aplikace jsou stále více závislé na skriptovacích technologiích na straně klienta, je důležité uvést, že obdobná situace je pozorovatelná na klientském stroji. Ať už se jedná o aplikace Flash, Java Applety, HTML 5, nebo javascriptové frameworky, v závislosti na míře abstrakce přinášejí obdobné problémy, jako na straně serveru popsané výše.

Typickým příkladem je jazyk „Typescript“ (zdroj [23]), který má za cíl odstínit vývojáře komplexních aplikacích na straně klienta od nízkoúrovňových problémů javascriptu. Navíc přináší do tohoto jazyka silně-typovou kontrolu, definice tříd a další. Výhodou tohoto přístupu může být zvýšená produktivita, možnost odstínění od odlišných implementací v různých prohlížečích nebo v některých případech zpřehlednění architektury a návrhu. Negativním dopadem na bezpečnost může být přílišná abstrakce a s tím spojená neznalost nižší vrstvy vývojáři a zmiňované slepé spoléhání na platformu. Jelikož se „Typescript“ kompiluje do Javascriptu, je nutné si uvědomovat některé důležité konsekvence – například vývojář spoléhající na abstraktní „Typescript“ může ukládat na klientském stroji citlivé informace v „privátních“ proměnných, aniž by věděl, že v přeloženém javascriptu není prakticky možné jejich čtení zabránit.

## 2.2 Hlavní bezpečnostní problémy platformy

V kontrastu s v úvodu popisovanou složitostí webové platformy, její základy jsou vystavěny na jednoduchých principech. To také podle [4] umožnilo webu a Internetu jejich prudký rozvoj. Tyto jednoduché základy jsou však zásadně nadužívány a v současném stavu je otázkou, zda limitují možný vývoj webových aplikací, nebo je dosahováno hranice, kdy jejich použití i v zájmu bezpečnosti aplikací ještě dává smysl.

### 2.2.1 Bezstavovost

Jedním z klíčových konceptů umožňujícím snadnou tvorbu a dodání **obsahu** v počátcích WWW („World Wide Web“) je bezstavovost. Jedná se o vlastnost primárně HTTP protokolu, kdy strany (klient a server) naváží spojení, dotazující se klient požaduje určitý zdroj, server odpoví a spojení je ukončeno. Mezi jednotlivými dotazy na server není žádná vazba.

Tato vlastnost je zásadní překážkou při tvorbě **aplikací**, jelikož základem většiny z nich je činnost nezávislých klientů pod vlastními pracovními „účty“ a s vlastními daty. Oddělení práce klientů je typicky řešeno uchováváním stavu v aplikaci – v případě architektury aplikace na jednom stroji („tlustého klienta“) je udržován stav ve vnitřní paměti na stroji, u klient-server architektury je používán typicky stavový protokol umožňující otevřít spojení a udržovat jej po celou dobu práce s aplikací. Už první víceuživatelské webové aplikace musely tento princip porušovat. Následně se standard (protokol HTTP) společně s prohlížeči přizpůsobil novým požadavkům. Prohlížeče umožnily ukládání cookies, zapamatování uživatelského hesla a autentizace, automatické síťové přihlášení a v neposlední řadě umožňují ukládat komplexní data „offline“ (HTML5).

Řešení bezstavovosti lze rozdělit do dvou skupin: opakovaně přihlašující a persistentní. Opakované typy přihlášení fungují typicky tak, že prohlížeč požaduje obsah po serveru, ten vrací odpověď s kódem 401 (neautorizováno), prohlížeč pak vyžádá po uživateli jméno a heslo (případně certifikát nebo jiné). Při dalších požadavcích na server je následně heslo (nebo jeho obdoba) odesíláno na server opakovaně (je udržován stav). Dochází tedy k odesílání citlivé informace při každém požadavku. K tomuto typu řešení patří např. „Basic authentification“ („základní autentizace“, autor) - autentizace na úrovni protokolu HTTP (zdroj [4], pojmenování převzato ze serveru Microsoft IIS, „Internet Information services“). Kapitoly věnované autentizaci popisují útoky, které jsou zaměřeny na napadení takových mechanizmů.

Persistentní řešení bezstavovosti využívá jednoduchého principu – při požadavku na server je vytvořen unikátní klíč, dostatečně dlouhý a náhodný, aby nebylo snadné jej uhodnout hrubou silou, a ten je odeslán na klienta. Klient jej v nějaké podobě uchovává a odesílá zpět na server s dalšími požadavky. Server pak citlivé informace (jako, jestli je uživatel přihlášen, jeho heslo apod.) uchovává na své straně. Je-li jednou klient přihlášen (jménem a heslem, certifikátem nebo jinak), odesílá už „jen“ tento náhodný klíč (tzv. „session id“, „identifikátor sezení“, autor), nikoliv samotné citlivé údaje. Typicky jsou identifikátory uchovány v cookies (viz níže), případně ve formulářových polích („input type hidden“) nebo v URL („uniform resource identifier“) parametru. Uchování v jiném úložišti, než v cookies je jedním z široké škály bezpečnostních rizik využívaných útoky zaměřenými na tzv. „správu relací“ (z angl. „session management“) - kapitoly 4.2, 5.1, 6.2 a další se věnují útokům napadajícím toto řešení.

### 2.2.2 Klient, prohlížeč

Jakákoliv informace odeslaná z klienta může být (a je dobré předpokládat, že bude) podvrh. Po odeslání dat ze serveru není možné očekávat prakticky nic s jistotou. Od nejnižších hardwarových vrstev, jejích ovladačů, síťových vrstev, protokolů, fyzické sítě mezi klientem a serverem, až po klientský stroj, jeho identifikaci, operační systém, prohlížeč a kód v něm běžící, všechny tyto prvky komunikace mohou být podvrženy. Útočník může změnit obsah veškerých skrytých polí, cookies, technologií flash odeslaných dat, kompletně ovládat klientský javascript, měnit a odeslat libovolné soubory, podvrhnout certifikáty, kompromitovat SSL („Secure sockets layer“), falšovat IP adresu („Internet protocol adress“), hlavičky HTTP (např. referer, cookies, user-agent), URL adresy a další. Veškerý vstup z klienta je tedy kriticky důležité ověřovat.

Jak ukazují příklady některých zranitelností poslední doby (2013 – 2015), podvržené a dobře připravené vstupy z klienta jsou hlavním zdrojem bezpečnostních rizik:

- Heartbleed (zdroj [10]) je bezpečnostní chyba, při které útočník pomocí připraveného **nevalidního** požadavku na server získá fragment paměti serveru. Principiálně je serveru odesílán řetězec a jeho délka. Server by měl řetězec vrátit. Chybná implementace v některých verzích kryptografické knihovny Open SSL (zdroj [28]) však vrací úsek paměti serveru odpovídající požadované délce – **bez validace**. Pokud je zaslán korektní vstup (zpráva „test“ a číslo 4) je vše v pořádku. Je-li odesláno číslo 65535, server vrací nezabezpečenou paměť. Ta může (a bude) obsahovat citlivá data nebo například kritické kryptografické klíče. Útok je podobný zranitelnostem pracujícím s přetékáním paměti (tzv. „buffer overflow“).
- Akustická kryptoanalýza (zdroj [33]) je bezpečnostní chyba, která umožňuje útočníkovi využít tzv. „postranních kanálů“ (zvuku vydávaném součástkami počítače – nikoliv mechanického původu) k odposlouchávání kryptografických klíčů při šifrování RSA („Rivest Shamir Adleman“). Klíčem k úspěchu této metody je precizně **připravený vstupní řetězec** odesílaný na cílový stroj. Při zpracování vstupu algoritmem dochází k časově náročným výpočtům, které je možné vysledovat a zneužít.
- Apple iCloud – únik citlivých dat (zdroj [34]) ze serverové služby společnosti Apple byl podle zdroje způsobený otevřenou zranitelností služby, při které je možné hrubou silou hádat uživatelská hesla. Klíčovým faktorem úspěchu útoku je **podvržený vstupní řetězec**, díky kterému je možné opakovaně zadávat hesla, aniž by došlo k automatickému zablokování účtu.

Dalším, neposledním faktorem bezpečnosti klientské strany webové platformy je samotné běhové prostředí, tedy prohlížeče. Webové prohlížeče procházejí už od prvotního rozmachu webu velmi prudkým vývojem. Je-li na prohlížeč nahlédnuto jako na běžnou, i když velmi složitou aplikaci, je zřejmé, že jejich implementace přináší velké množství vlastních chyb a bezpečnostních rizik. Široké spektrum zranitelností spoléhá na chybnou, nestandardní nebo experimentální implementaci v prohlížeči. Často velmi uživatelsky přívětivá funkčnost zavádí mnoho rizik.

Příkladem je funkce ukládání přihlašovacích údajů nebo obsahu formulářů a jejich předvyplňování. Uživatelsky velmi oblíbená funkce, kterou průběžně napadá mnoho útoků. Typickým zástupcem je útok, který umisťuje na formulář webové aplikace pole pro vyplnění jména uživatele. Pod tímto polem jsou však vizuálně skryta další pole, typicky s adresou a dalšími údaji. Uživatel zvolí položku jména, prohlížeč nabídne předvyplnění položky a společně s ní vyplní

i adresu a další data.

Na prohlížeče je tedy v současnosti vhodné nahlížet obdobně, jako na operační systém. Projekt společnosti Google vytvářející operační systém prakticky pouze z prohlížeče je tohoto trendu důkazem (zdroj [13]). Představa, že by v operačním systému uživatele mohl jakýkoliv program spouštět, instalovat a nahrávat libovolný kód je nemyslitelná. O to více alarmující je fakt, že v prohlížeči drtivá většina uživatelů běžně povoluje běh libovolného kódu jazyka javascript.

### 2.2.3 Cookies

Částečně popsaná kritická součást webové platformy jsou tzv. „cookies“ (česky „sušenky“). Cookies jsou textové soubory umožňující serveru (resp. javascriptu) ukládat na klientském stroji krátkou textovou informaci. Byly navrženy (RFC, „Request For Comment“, 2109, 2965 a 6265 z let 1997, 2000 a 2011) jako řešení stavových informací nad nestavovým protokolem – původně pro ukládání „nákupního košíku“. Staly se zároveň řešením managementu relací uživatelů. Jak popisuje kapitola Bezstavovost, toto kriticky důležité úložiště postupem času převzalo roli důvěryhodného úložiště identifikátorů relací na klientském stroji. Mimo to jsou využívány k ukládání javascriptových dat, uživatelských nastavení a přizpůsobení stránek.

Cookies samotné nejsou běžně využitelné pro přímý útok na klientský stroj. Až na velmi vzácné příklady vycházející z chyb v prohlížečích, není možné nainstalovat s jejich pomocí vir, útočný kód, ani nahrát spustitelná data. Cookies jsou ale hojně využívány pro ukládání tzv. „tracking“ („sledovacích“, autor) dat. Ta pak slouží ke sledování pohybu uživatelů po síti Internet a k potencionálnímu narušení jejich soukromí.

Extrémním případem nadužívání cookies jsou pak tzv. „zombie cookies“ (nebo také „perzistentní cookies“). Jedná se o techniku, kdy je cíleně vytvořena informace na klientském stroji tak, aby nebylo snadné (příp. možné) ji odstranit. Otevřenou knihovnu (javascript) „evercookie“ (zdroj [16]) je možné využít k automatickému uchování a obnově dat mj. v „cookies“, LSO („local shared objects“ pro flash), fiktivně cachovaném obrázku vytvořeném pomocí HTML 5 canvasu, HTML 5 storage a dalších umístěních. Takovýto perzistentní a uživatelem běžně nesmazatelný identifikátor lze využít ke sledování a dalším aktivitám narušujícím soukromí.

Mimo výše uvedená rizika jsou cookies často terčem útoků, které se snaží kompromitovat mechanizmy udržování uživatelských relací. Se zvyšováním důrazu na bezpečnost webu také stoupají opatření pro zabezpečení cookies, jako znemožnění čtení javascriptem (znak „http only“), povolení přenosu cookies pouze na bezpečné přenosové lince (znak „secure“) a další.

## 2.3 Problematika standardů webu

Již od počátku WWW postupně gradoval problém s udržením tempa velmi prudkého rozvoje webových technologií, prohlížečů obsahu a tvůrců obsahu pod kontrolou tak, aby zkušenost uživatelů byla v mezích možností stabilně kvalitní a v době webových aplikací také především bezpečná. Vzhledem ke způsobu vzniku a rozmachu Internetu a jeho technologií trvalo relativně delší dobu, než se na poli ustálilo několik institucí, které udržují standardy webu.

Následující seznam jmenuje některé vybrané standardy:

- W3C standardy konzorcia W3C (tzv. „recommendations“),

- RFC dokumenty společenství Internet Engineering Task Force - IETF (z angl. „request for comments“; dokumenty také tvořící tzv. „Internet standards“),
- standardy ISO („International Organization for Standardization“).

Už z faktu, že standardizačních organizací je více, standardizace je decentralizovaná a neexistuje autoritativní přístup k tvorbě a vyžadováním plnění standardů je zřejmé, že se jedná o velmi komplikovaný soubor pravidel a velmi komplexní proces jejich údržby a vzájemného sladění.

Výše popsané problémy vedou dodnes k rozdílům v implementaci standardů webovými prohlížeči (dodavateli, tzv. „vendory“) nebo servery. Tyto rozdíly jsou mnohdy také záměrné – v době tzv. „browser wars“ („válek prohlížečů“) se jednotlivé klientské aplikace snažily získat větší podíl na trhu díky experimentálním, nestandardním nebo uživatelsky přitažlivým funkcím. Některé funkce uživatelé následně vyžadovali od tvůrců aplikací, což vedlo do spirály – standard se snažil vytvořit systematické a univerzální nástroje (jako např. kaskádové styly – „css“), prohlížeče obsahovaly standardu neodpovídající prvky (jako např. tag „`<marquee>`“), uživatelé požadovali nové funkčnosti a tvůrci aplikací hledali rovnováhu mezi standardností a přívětivostí.

Tyto rozdílnosti, nestandardnosti a prudký vývoj dodnes dopadají na web ve formě doznívajících platforem (jako prohlížeč Internet explorer veze 8.0 a nižších), zpětně kompatibilních standardů komplikujících tvorbu a především pak rozsáhlou komunitu vývojářů, kteří nepovažují znalost standardů za důležitou a běžnou.

Z bezpečnostního hlediska jsou primárním problémem:

- doznívající platformy – prohlížeče, servery a technologie, které nepodporují moderní bezpečnostní standardy,
- rychlá adaptace nových nedostatečně prověřených standardů – vývojáři tlačeni poptávkou používají abstraktní knihovny využívající nové nebo experimentální funkce HTML 5 apod.,
- omezená standardnost aplikací – vývojáři pracují primárně na základě svých předchozích zkušeností (viz popis „zlozvyků“ z minulosti výše), nebo metodou „pokus-omyl“, což vede k řešením, která zdaleka nemusí být bezpečná.

Neposledním bezpečnostním problémem je pak nově příchozí standard HTML 5. Tento standard zavádí do webové platformy velké množství nových konceptů, které se snaží najít odpovědi na otázky řešené současnými webovými aplikacemi. Základním společným rysem většiny z těchto změn je rozšiřování možností webu – tedy především prohlížečů a jazyka javascript. Nový standard například přináší možnost pracovat s lokální databází (např. pro „offline“ režim), zpracovávat a vykreslovat 3D grafický obsah s možností pracovat přímo s grafickým hardwarem nebo možnost pracovat lokálně se soubory. Tato rozšíření s sebou krom mnohdy pozitivních důsledků na vývoj webových aplikací přináší nová bezpečnostní rizika a dotahují přeexponované technologie webu do extrému.

# 3 . Kategorizace, východiska a struktura metodiky

Tato kapitola popisuje koncepci tvorby metodiky, východiska a zdroje, ze kterých čerpá, specifikuje základní kategorie a pojmy a definuje úrovně bezpečnosti.

## 3.1 Východiska metodiky

Jelikož je metodika primárně zaměřena na vývojáře a testery, jejím hlavním obsahem je popis principů a technologických postupů zneužití zranitelností a obrany proti nim. Z tohoto důvodu nejsou popisovány nástroje použitelné pro automatickou (resp. poloautomatickou) detekci zranitelností, statickou a dynamickou analýzu kódu ani obdobné. Práce se zaměřuje na manuální provedení útoku a analýzu jeho principu. Zároveň doporučované techniky obrany jsou omezeny primárně na principy a manuální ošetření bez využití rozšířených frameworků.

Na rozdíl od dostupných obdobných metodik se práce věnuje konkrétnímu popisu problému na dané platformě s kompletní ukázkou kódu útoku i aplikace, přičemž všechny příklady jsou podloženy kompletní přiloženou ukázkovou aplikací (na CD).

### 3.1.1 Vymezení výběru zranitelností

Téma bezpečnosti webových aplikací se prolíná s obecnou bezpečností aplikací (příp. bezpečností obecně). Z tohoto důvodu je nutné zúžit okruh vybíraných bezpečnostních problémů. Bezpečnost obecně zahrnuje například oblast řízení rizik („risk management“), kterému jsou věnovány celé samostatné metodiky. Obecná bezpečnost aplikací se pak věnuje tématům, jako správa paměti, práce s ukazateli, řízením oprávnění na úrovni operačního systému aj.

Ani tak rozsáhlé zdroje, jako ty uváděné v kapitole 3.2, nepokrývají celé výše popsané spektrum. Navíc jejich praktické využitelnosti pak tento velký rozsah může bránit. Z těchto důvodů se tato práce omezuje na problémy vybírané orientačně podle následujících pravidel:

- Týkají se běžně webových aplikací,
- jsou běžně aplikovatelné na cílené platformě ASP.NET MVC,
- jsou obecně v kompetenci vývojáře webové aplikace,
- riziko, které představují není zanedbatelné,
- autor práce je na základě svých zkušeností považuje za reálné riziko v praxi.

Z výše uvedených bodů vyplývá, že se metodika nevěnuje například přímé správě paměti a ukazatelům (tzv. „unmanaged code“) v jazyce C#, jelikož běžně není nutné pro webovou aplikaci na dané platformě takový kód vytvářet. Obdobně se metodika až na výjimky nevěnuje problémům bezpečnosti mobilních zařízení.

Bod vyznačující, že vybraná zranitelnost je v kompetenci vývojáře webové aplikace vymezuje, že je běžně vytvořena na úrovni webové aplikace nebo v blízkém okolí. To znamená, že například chybná architektura uživatelských skupin na adresářovém serveru do této kategorie nespadá.

Poslední bod naznačuje jistou subjektivitu výběru zranitelností vloženou autorem. Jedná se o výběr útoků, které je s přihlédnutím k ekonomickým faktorům ještě praktické provádět. Je

zvažována motivace případných útočníků a obránců (resp. firem). Vzhledem k časovým i rozsahovým limitům je autor nucen omezit výběr zranitelností na základě svého úsudku. Jak popisují následující kapitoly, při tvorbě metodiky je primárně použit přesně specifikovaný postup. Stejným postupem je tedy možné v budoucnu metodiku rozšířit o autorem vyřazené zranitelnosti.

### 3.1.2 Dostupné metodiky

Jak je uvedeno v úvodní kapitole 1.2, dostupné metodiky bezpečnosti aplikací lze rozdělit do tří skupin: obecné odborné, specializované oborové a ostatní. Každá z těchto skupin má rozdílný obsah a cílovou skupinu. Text níže se věnuje popisu vybraných metodik.

Do první skupiny spadají například projekty rodiny OWASP ([29]), jako je standard ASVS 2014 ([17]). Jsou to odborné obecné metodiky, které mohou využívat profesionálové, testeři a vývojáři a které nejsou vytvořeny institucemi pro regulaci specifických oborů. Jsou tedy vhodné pro obecné aplikace i webové aplikace, stejně, jako tato práce.

Druhá skupina jsou standardy odvětví, typicky vytvářené státními institucemi a regulátory. Zástupci jsou standardy PCI DSS („The Payment Card Industry Data Security Standard“, „Standard bezpečnosti dat odvětví platebních karet“, zdroj autor) a HIPAA („Health Insurance Portability and Accountability Act“, „Zákon o přenosnosti a odpovědnosti ve zdravotním pojištění“, zdroj autor). Jedná se o vysoce specializované metodiky určené k uzákonění bezpečnosti v některých odvětvích, případně státem sledovaných oborech. Tyto standardy jsou vhodné pro bezpečnostní audity a také jako základní stavební kámen bezpečnosti aplikací, které jim musí odpovídat. Svým rozsahem, určením i obsahem kladou vysoké nároky na tvůrce aplikací a přináší relativně malý „zisk“ v podobě kvality (resp. bezpečnosti) aplikace.

Třetí skupinu tvoří ostatní metodiky, které se zabývají bezpečností jen okrajově, nebo z jiného hlediska. Mezi tyto metodiky se řadí rodina ISO („International Organization for Standardization“, „Mezinárodní organizace pro normalizaci“), např. ISO 27 000, nebo standard TEMPEST, vzdáleně také rodina ISMS („Information security management system“, „systém řízení bezpečnosti informací“, autor), ISO 20000, ISO 9001 a ITIL („Information Technology Infrastructure Library“, „Knihovna infrastruktury informačních technologií“, autor) a COBiT („Control Objectives for Information and related Technology“, [1], „Kontrolní cíle pro informace a související technologie“, autor), nebo spíše jim přidružený Risk IT. Dále také skupina metodik řízení a měření rizik, jako DREAD nebo STRIDE. Všechny tyto standardy mají význam pro specifickou oblast. Z pohledu bezpečnosti se věnují jejímu procesnímu zařazení, managementu, měření a řízení, procesům prevence, nebo také specifické bezpečnosti, jako fyzickému zabezpečení hardwaru a ochraně dat. Jsou tedy vhodné zejména pro management bezpečnosti, popis procesů a úzké specifické technické oblasti.

Ze jmenovaných skupin jsou pro využití vývojáři a testery aplikací vhodné zejména metodiky první skupiny, nicméně ani tato skupina většinou neposkytuje konkrétní kroky, ukázky a postupy pro ověření a zabezpečení webových aplikací.

#### *3.1.2.1 PCI DSS*

„The Payment Card Industry Data Security Standard“ je standard věnující se bezpečnosti použití platebních karet, uchování jejich dat, identifikace, přenosu apod. Standard se věnuje podrobné specifikaci 12 vysokoúrovňových požadavků na zabezpečení platebních karet, popisu postupu verifikace, testovacím procedurám, popisu formálních výstupů testování a dalším

tématům. Standard má přímý vztah s PA-DSS („Payment Application Data Security Standard“), který se věnuje bezpečnosti samotných aplikací, nikoliv platformy. Jak uvádí metodika, *„použití PA-DSS aplikace ještě neznačí, že organizace plní PCI DSS“* (volně přeloženo, zdroj [30]).

Požadavky PCI DSS se týkají témat, jako budování bezpečné sítě a systémů, ochrana dat držitelů karet, management zranitelností, implementace silných kontrol přístupu, monitorování a testování sítí a údržba bezpečnostní politiky informací.

Standard je dostupný ve zdroji [30].

#### *3.1.2.2 HIPAA*

„Health Insurance Portability and Accountability Act“ je zákon věnující se zabezpečení osobních citlivých zdravotních údajů (z angl. „Personal Health Information“, PHI), jejich uchování, zpracování, přenosu nebo omezení oprávnění k jejich zpracování. Tento velmi rozsáhlý komplex dokumentů vytvořený a uzákoněný v roce 1996 vymezuje pravidla, politiky, postihy, metody ověření, správu identifikátorů nebo rozsah samotných osobních informací. Z pohledu bezpečnosti webových aplikací je standard psán v době, kdy nebylo možné předvídat vývoj aplikací k dnešní podobě. Z tohoto důvodu je relativně obecný a jeho ověření a plnění vyžaduje důkladný audit.

Pravidla popisují primárně tři oblasti: administrativní ochranu, fyzickou ochranu a technickou ochranu. V rámci těchto oblastí jsou definována pravidla, která aplikace spadající pod tento zákon musí plnit. Z pohledu vývojářů a testerů je standard příliš rozsáhlý a obecný.

Standardu se věnuje zdroj [35].

### 3.1.3 Formální způsob zpracování testů

Při využití metodiky je vhodné zpracovávat ověřování a testy jednotně tak, aby byly výstupy srovnatelné a případně využitelné při auditu.

#### *3.1.3.1 Testování aplikace metodikou*

V případě využití metodiky při testování aplikací je vhodné dodržet následující doporučení:

- Uvádět detailní postup reprodukce. Jelikož je metodika koncipována tak, že jednotlivé zranitelnosti v úrovních jsou relativně obecné, je vhodné vývojáři doplnit odkaz na nalezenou zranitelnost přesným postupem.
- V reportu chyby uvést kategorizaci zranitelnosti, která navede vývojáře na možné umístění primárního zdroje problému.
- V reportu chyby uvést severitu a pokusit se reprodukovat typický dopad zranitelnosti popsaný v metodice.

#### *3.1.3.2 Audit bezpečnosti*

V případě využití metodiky a výstupů z testování při auditu bezpečnosti informačního systému je vhodné dodržet následující doporučení:

- V rámci reportu uvést specifikaci úrovně, do které aplikace podle tvůrce spadá, pro určení plnění metodiky.
- V rámci reportu jmenovat klíčové zranitelnosti, které vzhledem k aplikaci představují

nejvyšší riziko podle zadavatele auditu.

- Pro jednotlivé zranitelnosti uvést, v jakém rozsahu byly testovány a do jaké míry požadavky splňují. Vzhledem k obecnější povaze popisu zranitelností je obtížné určit, zda je aplikace zabezpečena proti celé popsané problematice.
- Využít jednotlivé zranitelnosti a úrovně jako základní check-list pro audit.

Následující tabulka uvádí příklad formálního výstupu auditu:

| | | | |
|---|---|---|---|
| **Očekávaná úroveň auditované aplikace:** | | Náležitá | |
| **Vyhodnocení splnění úrovně:** | | **Nesplňuje** | |
| **Splňovaná úroveň (volitelně):** | | - | |
| **Klíčové zranitelnosti:** | | 1.1 | |
| | | | |
| *Požadavky:* | *Míra splnění:* | *Dostupné provedené testy:* | *Poznámka:* |
| 1.1 | Částečně splňuje | T1_SQLInjection, T2_SQLInjection | Využívá protokol LDAP, který je náchylný LDAP Injection |
| 1.2 | Nesplňuje | T3_Session | |
| 1.3 | Splňuje | T4_Authorization_1, T5_Authorization_2 | |
| 1.4 | Nesplňuje | T4_Authorization_1, T5_Authorization_2 | Administrační rozhraní zabezpečeno jménem a heslem umístěným v konfiguraci v podobě čistého textu |
| 1.5 | Splňuje | T6_Database | |

*Tabulka 3.1: Příklad formálního výstupu auditu, zdroj autor*

## 3.2 Metoda získání informací

Metodika je sestavena z níže uvedených primárních zdrojů. Každý tento zdroj poskytuje jiné a do jiné podrobnosti popsané informace – každý je tedy zpracován odlišným způsobem. Všechny primární zdroje jsou ale zpracovány stejným postupem a se stejným výstupem. Postup je popsán v následující podkapitole. Primární zdroje jsou:

- OWASP TOP 10 ([37]),
- OWASP ASVS ([17]),
- CWE („Common Weakness Enumeration“, společnosti Mitre) ([24]),
- CWE/SANS TOP 25 ([9]).

Ostatní zdroje využité pro některé typy útoku jsou uvedeny v kapitole Zdroje.

### 3.2.1 Obecný postup zpracování primárních zdrojů

Každý primární zdroj je nejprve analyzován a zpracován do stromové struktury kategorií, skupin útoků a případně konkrétních útoků. Kategorie a zranitelnosti jsou následně zařazeny do kategorií využívaných v této metodice. Kapitola 3.3 popisuje kategorie a způsob jejich vzniku.

Následně jsou z primárních zdrojů vybrány jednotlivé zranitelnosti. Postup zapracování zranitelnosti do metodiky je následující:

1. Poslední stav metodiky je vyhodnocen a je doplněno mapování s vybraným zdrojem (kapitola 8.1).
2. Zdroje mohou být zpracovány v libovolném pořadí i opakovaně.
3. Pro každou úroveň metodiky a každou její zranitelnost je vyhodnocena vazba na každou zranitelnost (resp. úroveň, případně kategorii) ve vybraném zdroji.
4. Vazba je hodnocena body vyjádřenými znakem podle tabulky níže.
5. Zranitelnosti (resp. úrovně, případně kategorie) s hodnocením nižším než 20 bodů jsou zařazeny ke zpracování do metodiky nebo k vyřazení. Zranitelnosti s hodnocením mezi 20 a 50 body jsou zvažovány k širšímu zpracování. Zranitelnosti s hodnocením mezi 50 a 100 body jsou pouze prověřeny (jejich dostatečnost). Zranitelnosti s hodnocením nad 80 bodů mohou být považovány za zpracované.
6. Zranitelnosti, které dosahují vysokého hodnocení především díky průnikům a neurčitým vazbám mohou být dále zpracovány.
7. K nové zranitelnosti k zařazení je provedeno vyhledání relevantních zranitelností, které mohou být spojeny do společné kapitoly.
8. Po doplnění metodiky o nové zranitelnosti se pokračuje od kroku 1.

Tabulka hodnocení vazeb metodiky a zdrojů:

| Kód | Hodnota | Popis |
|---|---|---|
| ? | 1 | '?' - neurčená vazba |
| + | 10 | '+' - průnik, doplňuje |
| < | 40 | '<' - je podmnožinou, zužuje |
| = | 100 | '=' - odpovídá |
| > | 110 | '>' - pokrývá, rozšiřuje |

*Tabulka 3.2: Hodnocení vazeb mezi metodikou a zdroji, zdroj autor*

### 3.2.2 Postup zpracování zdroje OWASP TOP 10

„Open web application security project - TOP 10“ („otevřený projekt bezpečnosti webových aplikací – TOP 10“, zdroj autor) je otevřený komunitní projekt zaměřený na bezpečnost webových aplikací. Primárním výstupem projektu je dokument obsahující popis deseti zranitelností (spíše však kategorií), které považuje za nejdůležitější bezpečnostní slabiny webových aplikací. Mezi

verzemi projektu se mění jak způsob určení „váhy“ dané zranitelnosti (například přechod od četnosti výskytu k řízení rizik), tak úroveň specifikace konkrétní zranitelnosti – od konkrétních ukázek kódu, až po principiální výčty a odkazy na další zdroje.

Kompletní dokument je možné získat ve formátu PDF ve zdroji [37]. Pro vytvoření metodiky v této práci je použit tento soubor ve verzi „2013 (release)“.

Zdroj je zpracován standardním způsobem uvedeným výše s tím rozdílem, že neobsahuje konkrétní výčty zranitelností, ale převážně jejich kategorie. V každé kategorii TOP 10 je uveden seznam typických zranitelných míst aplikace napadaných danou kategorií, způsob obrany většinou realizovaný odkazem na další projekty OWASP, popis principu zranitelnosti a odkazy na další zdroje. U každé kategorie je provedena analýza dalších zdrojů a v případě potřeby rozpad kategorie na konkrétní zranitelnosti. Například kategorie „A2 – Broken Authentication and Session Management“ je rozdělena na základní variace (jako kapitoly 4.2, 5.3), pokročilé 6.2 a také mezi další kapitoly týkající se konfigurace.

Ze zdroje jsou také využity informace o dodatečných často se vyskytujících zranitelnostech v kapitole „Additional risks to consider“ („další rizika ke zvážení“, autor).

### 3.2.3 Postup zpracování zdroje OWASP ASVS 2014

„OWASP – Application security verification standard 2014“ („OWASP – Standard ověření bezpečnosti aplikací 2014“, zdroj autor) je otevřený komunitní projekt zaměřený na bezpečnost webových aplikací z rodiny projektů „OWASP“. Hlavním výstupem projektu je dokument obsahující výčet celkem 140 pravidel (vynechána pravidla pro mobilní aplikace) ve 12 kategoriích. Tato pravidla jsou zařazena do tří úrovní zabezpečení (z celkových čtyř) – přičemž první úroveň nazvaná „Level 0: Cursory“ je ponechána na specifikaci samotným uživatelem ASVS.

Kompletní dokument je možné získat ve formátu PDF ve zdroji [17]. Pro vytvoření metodiky v práci je použit tento soubor ve verzi 2014 (s přihlédnutím k verzi 2013 beta).

Zdroj neobsahuje ani běžné kategorie ani konkrétní popsané zranitelnosti aplikací. Namísto toho obsahuje soubor slovně popsaných pravidel, která mají být splněna. Některá pravidla jsou popsána např. na úrovni použití jednoho konkrétního elementu v HTML dokumentu (V2.2), naopak některá jsou velmi obecná a abstraktní (V16.9). Do této práce jsou jednotlivá pravidla slučována do větších celků, které většinou odpovídají přibližně polovině obsahu každé ze 12 kategorií ASVS.

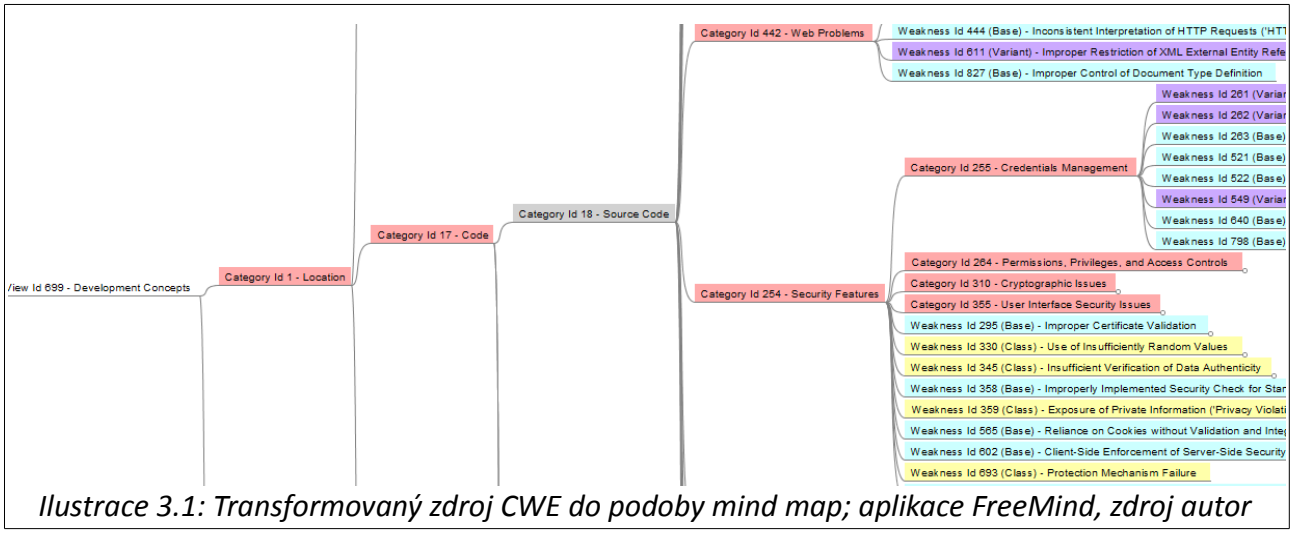
### 3.2.4 Postup zpracování zdroje CWE

Common Weakness Enumeration (CWE, „Výčet běžných zranitelností“, zdroj autor) je sponzorovaný projekt shromažďující informace o zranitelnostech v aplikacích vytvářený mj. řadou světových organizací. Výstupů tohoto projektu je velmi velké množství – primárně se však jedná o „pohledy“ („View“) na jednu sadu dat. Data obsahují více než 500 popisů zranitelností, jejich konsekvencí, s příklady, odkazy, vazbami, desítky kategorií a podkategorií a několik „pohledů“ a „řezů“ napříč těmito daty.

Kompletní strukturu/data projektu ve formátu XML („eXtensible Markup Language“) je možné získat ve zdroji [24]. Pro vytvoření metodiky v této práci je použit tento soubor ve verzi 2.8 a schéma (XSD, „XML Schema Definition“) verze 5.4.2.

Jelikož se jedná o rozsáhlý zdroj, jeho zpracování předcházela analýza jeho struktury, rozbor

dostupných vizualizací a extrahovaných pohledů (jako je pohled *„Seven Pernicious Kingdoms Taxonomy“*, zdroj [25]). Pro lepší orientaci v datech zdroje je vytvořena transformační šablona (XSLT, „eXtensible Stylesheet Language Transformations“), pomocí níž je zdroj převeden do podoby tzv. „myšlenkové mapy“ ve formátu aplikace FreeMind (zdroj [26]). Šablona je uvedena v příloze v kapitole 13.1. Následující obrázek ilustruje extrahovaný výstup:




*Ilustrace 3.1: Transformovaný zdroj CWE do podoby mind map; aplikace FreeMind, zdroj autor*

Takto zpracovaný zdroj je využit k sestavení kategorizací, hlavního obsahu úrovní metodiky i k výběru hlavních zranitelností. Detailní informace zdroje pak slouží jako podklad pro ukázky kódu a principy útoků.

### 3.2.5 Postup zpracování zdroje CWE/SANS TOP 25

„Common Weakness Enumeration / SysAdmin, Audit, Networking, and Security TOP 25“ („TOP 25 z výčtu běžných zranitelností a Administrace, auditu, sítí a bezpečnosti“, zdroj autor) je společný projekt tvůrců CWE a společnosti SANS, která se specializuje na standardizaci, audit a certifikaci v oblasti bezpečnosti. Primárním výstupem projektu je dokument podrobně popisující vybraných 25 bezpečnostních chyb ve třech kategoriích, jejich principů a ukázek zdrojového kódu. Dokument odkazuje na široké portfolio dalších zdrojů, které popisují odstranění těchto zranitelností nebo také související architektonické a návrhové chyby.

Kompletní dokument je možné získat ve formátu PDF nebo HTML ve zdroji [9]. Pro vytvoření metodiky v práci je použit tento soubor ve verzi 2011.

Zdroj obsahuje obdobnou úroveň popisu zranitelností jako tato metodika. Je využit jako podklad pro část zranitelností, kategorizaci a hodnocení ekonomických i technických aspektů útoků.

## 3.3 Kategorizace zranitelností

Jelikož je tato metodika primárně zaměřena na vývojáře, testery a převážně technickou stranu realizace bezpečné aplikace, jsou kategorie v metodice odvozeny od jednotlivých vrstev nacházejících se v běžných aplikacích tohoto typu. Jedná se tedy o kategorie ve smyslu umístění dané zranitelnosti. Základní sadu kategorií tvoří:

[A] Prezentační vrstva,

[A.1] uživatelské rozhraní,

[A.2] prezentační model,

[A.3] webová vrstva,

[B] logická vrstva,

[B.1] aplikační logika,

[B.2] byznys logika,

[C] datová vrstva,

[C.1] databáze,

[C.2] ostatní,

[D] bezpečnostní vrstva*,

[E] integrační vrstva*,

[F] konfigurační vrstva*.

* bezpečnostní, integrační a konfigurační vrstvy jsou ve skutečnosti spíše oblastmi, než samostatnými vrstvami, protože jsou typicky provázány napříč aplikací.

Do jednotlivých kategorií jsou pak umisťovány zranitelnosti podle nejpravděpodobnějšího místa výskytu. Jedna zranitelnost může být umístěna ve více kategoriích. Níže uvedená tabulka popisuje jednotlivé kategorie a jejich přibližný rozsah.

| **Kategorie** | **Popis** |
|---|---|
| [A] Prezentační vrstva | Obsahuje vrstvy rozhraní pro komunikaci uživatele s aplikací, včetně uživatelem běžně nepoužívaných komunikačních kanálů (např. HTTP hlavičky) |
| [A1] Uživatelské rozhraní | Uživatelské rozhraní od klientského prohlížeče, přes javascript, HTML a další výstupy, až po jednoduchou logiku zobrazení |
| [A2] Prezentační model | Vrstva zpracovávající požadavky HTTP (controllery, rozhraní), webové komponenty, helpery a veškerá základní logika práce s webovými požadavky (např. zpracování nahrání souboru) |
| [A3] Webová vrstva | Obsluha HTTP protokolu, hlaviček, expirací, cache, přesměrování, správu parametrů URL, směrování, cookies |
| [B] Logická vrstva | Veškerá logika aplikace, až na základní formátování a jednoduché rozhodování |
| [B1] Aplikační logika | Logika zpracovávající operace pro webové vrstvy, integrační vrstvy, podpůrné činnosti (např. vyhodnocování používaných implementací), zpracování operací pro byznys logickou vrstvu |
| [B2] Byznys logika | Pravidla práce s byznys objekty, zpracování jejich interakcí, vyhodnocení všech datově, výpočetně nebo obsahově důležitých operací |
| [C] Datová vrstva | Zahrnuje veškerý přístup k datům, souborům, podpůrným zdrojům |

| Kategorie | Popis |
|---|---|
| | Neobsahuje čistě webové přístupy a zdroje |
| [C1] Databáze | Přístup k databázovým systémům, úložištím, datovým serverům, reportům, analytickým nástrojům |
| [C2] Ostatní | Veškerý další přístup k datově významným zdrojům |
| [D] Bezpečnostní vrstva | Zabezpečení, autorizace a autentizace primárně na logické úrovni (nikoliv konkrétní HTTP metody apod.), integrační zabezpečení |
| [E] Integrační vrstva | Správa přístupu k externím systémům, propojení se vzdálenými zdroji, poskytování služeb |
| [F] Konfigurační vrstva | Veškerá konfigurace potřebná pro všechny ostatní vrstvy |

*Tabulka 3.3: Popis kategorií metodiky, zdroj autor*

## 3.4 Specifikace severity a dopadů

Jednotlivé zranitelnosti v metodice jsou krom výše popsané kategorizace označeny také tzv. „severitou“ (závažností) a analyzovány z pohledu technologických a ekonomických dopadů.

### 3.4.1 Severita

Severita a technický dopad jsou vyhodnoceny ve čtyřech možných scénářích útoku, přičemž celková závažnost zranitelnosti je z nich maximální dosažitelnou. Níže uvedená tabulka popisuje jednotlivá hodnocení severity.

| Kód | Název | Popis |
|---|---|---|
| A+ | Kritická | Zranitelnost umožňuje kompletní převzetí systému, infrastruktury nebo aplikace, její dopady jsou zcela v rukách útočníka a mohou být nevratné, postihuje všechny uživatele, systém, nebo správce a infrastrukturu bez možnosti ji obejít, její odstranění vyžaduje nové zavedení systému nebo dokonce úplné zrušení infrastruktury, může být aplikována zcela beze stop |
| A | Vysoká | Zranitelnost zasahuje do větší části aplikace nebo systému, její dopady jsou nevratné, postihuje všechny uživatele bez možnosti ji obejít a následným útokům je možné zabránit jen novým zavedením systému a opravou, je aplikována s minimálním množstvím stop |
| B | Střední | Zranitelnosti není možné se vyhnout, její dopady lze neutralizovat pouze zásahem/opravou, ale je možné se jí uživatelsky vyvarovat a následným útokům většinově zabránit, její aplikace zanechává stopy |
| C | Nízká | Zranitelnosti je možné se vyhnout, její dopady snadno zcela neutralizovat nebo obejít a následným útokům zabránit, její aplikace zanechává uživatelské stopy |

*Tabulka 3.4: Popis hodnocení severity, zdroj autor*

Zranitelnost nemusí plnit všechny popsané vlastnosti severity, aby tak mohla být označena. Následující tabulka dále popisuje jednotlivé scénáře útoků.

| Úroveň útoku | Popis |
|---|---|
| Základní/amatérský | Útok prováděný náhodným nebo částečně cíleným útočníkem amatérem. Není prováděn s konkrétním a přesným cílem. Není cíleně maskován. |
| Odborný náhodný | Útok prováděný náhodným odborníkem nebo skupinou. Je prováděn s obecnějším cílem. Je maskován za běžnou činnost uživatele. |
| Odborný cílený | Útok prováděný cíleným odborníkem nebo skupinou. Je prováděn s konkrétním cílem. Je maskován se snahou o odstranění stop. |
| Profesionální cílený | Útok prováděný profesionálním cíleným týmem. Je prováděn s přesným cílem, pravidly nebo zadáním. Je maskován všemi dostupnými prostředky s využitím napadených uživatelských stanic, serverů a sítí se snahou o odstranění stop a vzbuzení minimálního podezření. |

*Tabulka 3.5: Popis scénářů a úrovní útoků, zdroj autor*

### 3.4.2 Technické dopady úspěšného útoku

Technické dopady útoků jsou stejně jako Severita hodnoceny v rámci čtyř scénářů útoku. Dopadem se v tomto kontextu rozumí stav, kterého docílí útočník použitím vybraného útoku.

| Dopad | Popis | Příklad |
|---|---|---|
| Data | Ztráta, kompromitace, zveřejnění, odcizení, poškození nebo jiné napadení dat uživatelů nebo aplikací | Vymazání dat z tabulky SQL úložiště |
| Systém | Ovlivnění, zneužití až částečné nebo úplně převzetí kontroly nad systémem, infrastrukturou nebo platformou | Spuštění příkazů v příkazové řádce operačního systému pod privilegovaným účtem |
| Služba | Zpomalení, poškození, znepřístupnění nebo kompletní vyřazení služby poskytované aplikací nebo systémem | Distribuované zahlcení systému požadavky |
| Účet | Zobrazení, napodobení, odcizení nebo znepřístupnění uživatelských účtů nebo relací, administračních účtů a správy | Odcizení relace uživatele na veřejné síti |
| Osoba | Zmanipulování osob, zneužití důvěryhodnosti, napadení jiných systémů a aplikací a poškození třetích stran | Zmanipulování uživatele k účasti na zahlcení serveru třetí strany |

*Tabulka 3.6: Popis možných technických dopadů útoku, zdroj autor*

### 3.4.3 Ekonomické aspekty úspěšného útoku

Ekonomické aspekty útoku jsou slovním zhodnocením dané zranitelnosti z pohledu správců (resp. vlastníků) aplikace a z pohledu útočníků. Primárně se zabývají dopady na: existenci subjektů systému, finanční výnosy (resp. ztráty) subjektů, zainteresovanost třetích stran a další faktory.

## 3.5 Základní pojmy

Metodika ve svém popisu využívá spíše volné pojmenování objektů a účastníků v systému. Následující základní definice jsou volně inspirovány normou ISO/IEC 27000:2012 (zdroj [15]).

Uvedený výčet jejich alternativ užívaných v textu není úplný. Tabulka níže obsahuje pouze vybrané pojmy, které při užití metodiky mohou být důležité pro korektní pochopení kontextu.

| Pojem | Alternativy | Popis |
|---|---|---|
| Útok | Napadení, zneužití, podvržení | pokus o zničení, odhalení, změnu, poškození, odcizení nebo získání neoprávněného přístupu k nebo neoprávněné užití aktiva |
| Aktivum | Zdroj, informace, data, údaj, služba, hodnota | jakákoliv položka (hmotná i nehmotná) mající hodnotu pro organizaci, uživatele nebo osoby |
| Autorizace | Oprávnění, kontrola přístupu | omezení přístupu k aktivům podle požadavků bezpečnosti a firemních podnikatelských procesů |
| Autentizace | Oprávnění, identifikace | stanovení ujištění o důvěryhodnosti udávané hodnoty vlastnosti subjektu (resp. pravost subjektu) |
| Odpovědnost | Dopad, výsledek | přiřazení rozhodnutí a akcí subjektu |
| Vlastnost | Atribut, údaj, hodnota | popisná hodnota objektu nebo aktiva, kterou lze kvantitativně nebo kvalitativně rozlišit |
| Ekonomická činnost | Provoz, funkce organizace | procesy zachování a údržby činnosti organizace a činnost samotná |
| Dopad | Výstup, výsledek | vyústění události ovlivňující aktiva a objekty |
| Severita | Závažnost, důležitost | celková míra rizika události, nebo změny aktiva (objektu) ve smyslu kombinace okolností a pravděpodobnosti |
| Pravděpodobnost | Šance, možnost | možnost, že něco nastane (událost, útok) |
| Data | Zdroj, informace, data, údaj, záznam | kolekce hodnot přiřazených objektům a aktivům |
| Událost | Akce, operace | výskyt nebo změna sady okolností |
| Bezpečnost | Zabezpečení, ochrana, ověření | zachování důvěryhodnosti, integrity a dostupnosti informací |
| Obrana | Ochrana, náprava | opatření s cílem odstranění potenciálního nenaplnění bezpečnosti |
| Vlastník | Zainteresovaná strana, zúčastněný, subjekt | osoba nebo organizace, která ovlivňuje, je ovlivňována nebo sebe považuje za ovlivňovanou rozhodnutími, útoky, dopady, odpovědností, událostí aj. |
| Třetí strana | - | osoba nebo orgán identifikovaný, jako nezávislý na zúčastněných subjektech ve smyslu dané události |
| Zranitelnost | Slabina, chyba | slabina aktiva nebo obrany, kterou lze zneužít k útoku |

*Tabulka 3.7: Základní pojmy inspirované normou ISO/IEC 27000:2012, zdroj autor*

Metodika dále využívá sadu příkladů aplikací, které zastupují v rámci úrovní konkrétní ověřovanou aplikaci. Aplikace naznačují rozsah, míru složitosti, předpokládaný obsah jejich

implementace a také předpokládaný zásah vlastníků a aktiv v případě útoku.

- Aplikace vnitrofiremního a mezi-firemního prostředí,
  - aplikace typu **ERP** („enterprise resource planning“, „plánování podnikových zdrojů“, autor), **CRM** („customer relationship management“, „řízení vztahu se zákazníky“, autor),
  - jedná se o malé interní, střední až velké mezi-firemní aplikace,
- aplikace pro osobní účely,
  - např. aplikace pro správu úkolů - „todo list“,
  - osobní aplikace malého rozsahu obsahující minimum důležitých aktiv,
- veřejné aplikace pro sdílení obsahu,
  - jako CMS („content management system“, „systém pro správu obsahu“, autor), Wiki, Chat, Fórum,
  - aplikace pro malé i velké skupiny uživatelů obsahující převážně veřejný obsah,
- aplikace státní správy,
  - nekritické aplikace státní správy, jako podpůrné evidence úřadů, pomocné aplikace pro rozhodování, veřejné databáze aj.,
  - kritické aplikace státní správy – aplikace ohrožující zdraví a život osob, pracující s utajovanými skutečnostmi, vojenské, aplikace zasahující významnou část populace.

## 3.6 Specifikace úrovní bezpečnosti

Metodika obsahuje čtyři úrovně bezpečnosti aplikací.

1. Náležitá,
2. základní,
3. standardní,
4. elitní.

Každá z úrovní je vhodná pro jiný typ aplikací – i aplikace pro osobní účely může aplikovat vhodné metody z elitní úrovně. Úrovně neobsahují veškeré zranitelnosti, které jsou pro dané aplikace nebezpečné. Vzhledem k množství uvedených zdrojů je nutné, aby tvůrci aplikace pokračovali ve výzkumu zranitelností své úrovně a aplikaci dále zabezpečovali.

První tři úrovně se zabývají především konkrétními problémy webové platformy, kroky jejich reprodukce a ošetření. Elitní úroveň se pak dále zabývá principy, dobrým návrhem a architekturou aplikace a bezpečnostními politikami.

Jelikož typy útoků spolu do značné míry souvisí, v některých případech je u zranitelnosti uveden pokročilý způsob zabezpečení, který je nad rámec dané úrovně. V takovém případě je u mechanizmu uveden popis, například *„[Oprava pro Úroveň 3 – Standardní]“*, který tím vymezuje danou opravu pro vyšší úroveň. Aplikace nesplňující tento bod tedy ještě není vyřazena z vybrané úrovně zabezpečení.

# 4 . Úroveň 1 – Náležitá

Tato kapitola se věnuje popisu Náležité úrovně zabezpečení webových aplikací a zranitelností do ní spadajících.

Náležitá úroveň je intuitivně očekávána každým běžným zákazníkem a uživatelem aplikací. Zahrnuje zabezpečení proti nejběžnějším a zároveň nejjednodušším typům útoků, útokům, které mnohdy mohou spouštět útočníci bez předchozích zkušeností.

Tuto úroveň by měla splňovat každá (i triviální) aplikace obsahující neveřejná uživatelská data – mezi ty patří minimálně e-mailová adresa, jméno nebo příjmení. I v případě aplikací, které o uživatelích žádná obdobná data neukládají, aplikace uživatele minimálně autentizuje na základě hesla – je důležité si uvědomit, že velká část uživatelů používá jedno nezměněné heslo pro více systémů. Prolomení obrany zcela vedlejší nebo podpůrné aplikace a získání hesel uživatelů je běžný způsob útoku na dobře zabezpečenou primární aplikaci.

Aplikace nesplňující plnohodnotně tuto úroveň, s výjimkou zranitelností nerelevantních pro typ aplikace nebo platformu, je nutné považovat za nebezpečné pro použití v jakékoliv oblasti – vnitrofiremním prostředí (typu ERP, CRM), aplikaci pro osobní účely (např. aplikace pro správu úkolů - „todo list“), veřejnou aplikaci pro sdílení obsahu (CMS, Wiki, Chat, Fórum), aplikace státní správy (všech typů).

Mezi typické zranitelnosti na této úrovni patří základní varianty Injection (SQL, základní vkládání kódu), chyby správy přístupu (omezení oprávnění), chyby autentizace nebo základní ověřování vstupů.

## 4.1 Zranitelnost: Podsunutí SQL

Podsunutí SQL (z angl. „SQL Injection“) je variací injection zranitelností („podsunutí“ nebo „vstříknutí“), které cílí na interpret jazyka SQL typicky na databázovém serveru (MSSQL server, MySQL, Oracle aj.). Pro aplikace nepoužívající SQL databázi, ale pokládající dotazy do jiného úložiště je možné vyhledat variaci injection pro daný typ úložiště. Zranitelnost je irelevantní pro aplikace nepoužívající úložiště postavené na kladení skládaných dotazů (typicky textových).

### 4.1.1 Princip útoku a obrany

Základ zranitelnosti vychází ze zmiňovaného principu, kdy je datové úložiště dotazováno textovými dotazy jazyka SQL složenými aplikací. Tyto texty databázový stroj interpretuje a provádí požadované operace. Druhým pilířem zranitelnosti je nutnost aplikace využívat vstup od uživatelů – dotazy aplikace vychází z akcí uživatele. Může-li uživatel předat aplikaci vstup, který je následně odesílán interpretu SQL v rámci dotazu ke zpracování, může se pokusit o SQL injection. Jelikož interpret SQL bez upozornění (příznaku, označení) není schopen rozpoznat, jaká část dotazu je dodána vývojářem aplikace a jaká uživatelem, jde o spojený text, může vhodně připravený vstup spustit libovolnou akci.

Princip obrany vychází z výše uvedených nutných podmínek pro úspěch injection. Předejít SQL úložišti (pilíř 1) nebo parametrům od uživatele (pilíř 2) není většinou možné. Klíčem je rozpoznání uživatelského vstupu interpretem – je nutné jej upozornit, že tento vybraný **parametr** není prováděcí instrukcí, ale vstupem uživatele (parametrem). Druhou úrovní obrany (ať už jsou parametrizovány dotazy nebo ne) je validace vstupu. Útoku níže lze snadno zabránit tak, že

aplikace validuje, jestli je vstup číslo. V případě SQL injection je vždy vhodnější použití white-list validace, jelikož injection může být velmi sofistikovaná.

### 4.1.2 Kategorizace

Kategorie: [C.1], [B.1]

#### *4.1.2.1 Severita, dopad*

| **Severita** | **A+, Kritická** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | A | Data |
| Odborný náhodný | A | Data, Služba, Účet |
| Odborný cílený | A+ | Data, Systém, Služba, Účet |
| Profesionální cílený | A+ | Data, Systém, Služba, Účet, Osoba |

*Tabulka 4.1: Severita a dopad Podsunutí SQL, zdroj autor*

#### *4.1.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizená data, uživatelské účty, vyhrožují kompromitací aplikace, společnosti, znehodnocují celý ekonomicko-společenský účel aplikace za úplatu. Dále riskují finanční postihy, žaloby a trestní řízení.

Vlastníci: ztrácí postavení na trhu, platí smluvní pokuty a penalizace za ztrátu dat, kompromitaci klientů, přicházejí o klientelu v důsledku poškození účelu aplikace, ukončují svou činnost.

Třetí strany: platí za cílené poškození (reputace) konkurence, odcizení dat s cílem průmyslové špionáže, napadení a převzetí systému.

### 4.1.3 Způsob prověření (provedení)

Typickým příkladem je vyhledání dat podle uživatelského vstupu. Uživatel zadá do formulářového pole, URL parametru nebo jinam hodnotu (například Id výrobku), aplikace vytvoří dotaz na data s omezením podle parametru:

URL uživatele:

```
http://aplikace.com/product?id=5
```

SQL dotaz aplikace:

```
string query = "select * from Product where Id = " + id;
//id je parametr z URL výše
```

Běžné dotazy s číslem produktu fungují správně – aplikace je funkční. Následující URL provádí SQL injection odstraňující všechna data z tabulky „Product“:

```
http://aplikace.com/product?id=5;delete from Product--
```

Parametr je z URL přečten a na serverové straně vznikne dotaz:

```
select * from Product where Id = 5;delete from Product--
```

Oba dotazy jsou interpretem zpracovány a tabulka je promazána. Klíčem úspěchu je oddělení útočného dotazu od původního (znak středník „;“). Ve složitějších dotazech je pak klíčový konec útoku, dvě pomlky („--“), které označují zakomentování zbytku dotazu (ignorování interpretem).

Příklad využití ukončení komentářem:

```
"select * from User where (type = 1 and login = '" + login + "')"
```

Řetězec login pro injection (apostrof ['] je využit k ukončení uvození řetězce):

```
login' OR 1=1)--
```

Při identifikaci potenciálních míst pro SQL injection je vhodné hledat: formuláře (HTTP POST) a jejich skrytá pole, zejména pak přihlašovací formulář, odkazy s parametrem (id z databáze, název položky, filtr), javascriptové dotazy na server (typicky ajax) s parametrem. Je vhodné využít některý z mnoha dostupných nástrojů pro statickou analýzu kódu (v případě white-boxu), např. ze zdroje [29].

### 4.1.4 Způsob obrany

Řešení SQL injection je více. Následující seznam uvádí příklady od nejvhodnějších. Všechny přístupy je vhodné kombinovat s robustní validací vkládaných vstupů:

- Využití objektového rozhraní pro kladení dotazů do databáze s vestavěným využitím parametrických dotazů (jako Entity framework a LINQ, nebo obecně ORM). Dotaz je tvořen zcela bez použití textových řetězců. Vstupy uživatele jsou předány a explicitně vyznačeny interpretu SQL.
- Využití SQL dotazů s parametrickým rozhraní a pomocí „abstraktního SQL“ (jako „EQL - Entity query language“, nebo „HQL“ - „hibernate query language“). Dotaz je tvořen textovým řetězcem, který není přímo SQL – do něj je teprve překládán. Vstupy uživatele jsou předány a explicitně vyznačeny interpretu SQL.
- Využití SQL dotazů s parametrickým rozhraní a SQL. Dotaz je tvořen textovým řetězcem SQL. Vstupy uživatele jsou předány a explicitně vyznačeny interpretu SQL.
- Využití SQL dotazů a skládání řetězců. Dotaz je tvořen textovým řetězcem SQL a přímo vloženými vstupy uživatele. Vstupy uživatele je nutné ověřovat robustními mechanizmy validace, knihovnami (např. OWASP ESAPI [29]). I přesto není dotaz zcela bezpečný.

  Je vhodné prověřit aplikaci nástroji pro statické vyhledávání SQL injection v kódu.

### 4.1.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Repository.Implementation.Insecure.UserRepository<br>Metoda GetUserForLogin |
| **Oprava** | T1_SecuBank.Repository.Implementation.Secure.UserRepository<br>Metoda GetUserForLogin |

#### *4.1.5.1 Ukázkové použití: zranitelné*

Aplikace využívá ADO.NET pro získání dat z databáze, vyhledává uživatele podle zadaného jména a hesla pomocí následujícího dotazu:

```
myCommand.CommandText = string.Format("select * from [User] where (Login
= '{0}' and Password = '{1}')", login, password);
```

Útočník může do formulářového pole pro „login“ vložit uživatele „admin“ a do hesla následující injection:

```
' OR Login = 'admin')--
```

Vzniklý dotaz je umožní přihlášení pod loginem „admin“.

Dalším možným útokem je snaha zaútočit na data – variant se nabízí mnoho. Je možné postupně odhalit kompletní strukturu databáze, vyextrahovat data, smazat logy apod. Následující injection odstraní všechny uživatele.

```
'); delete from User--
```

#### *4.1.5.2 Ukázkové použití: bezpečné*

Pro zachování příkladu je ponechán přístup pomocí ADO.NET. Nejedná se tedy o výše popsané „nejlepší řešení“ pomocí objektového rozhraní. Příklady objektového rozhraní však obsahuje ukázková aplikace T1_SecuBank. Dotaz to databáze je zadán následovně:

```
myCommand.CommandText = "select * from [User] where (Login = @login and
Password = @password)";
myCommand.Parameters.Add(new SqlParameter("@login", login));
myCommand.Parameters.Add(new SqlParameter("@password", password));
```

Interpretu je tedy explicitně vyznačeno, která část dotazu je parametrem. Výsledný skript odchycený nástrojem profiler (MSSQL management studio profiler) je následovný*:

```
declare @login nvarchar(5),@password nvarchar(23)
set @login=N'admin'
set @password=N''' OR Login = ''admin'')--'

select * from [User] where (Login = @login and Password = @password)
```

** výstup je upraven po formální stránce pro zvýšení čitelnosti*

Z výsledku je patrné, že SQL injection je „zachycena“ v parametru a nemůže ovlivnit průběh dotazu.

### 4.1.6 Dodatečné informace

Pro SQL injection je důležitý fakt, že nelze „slepě“ důvěřovat použitým frameworkům (jako ORM). Například využití ORM NHibernate ([27]) a jeho „HQL“ ještě nezaručuje bezpečí proti SQL

injection. Dotaz NHibernate lze zadat parametricky (korektně), nebo také spojováním textových řetězců (chybně). Entity framework je podle zdroje [20] rozdělen stejně.

## 4.2 Zranitelnost: Únos relace

Únos relace (z angl. „Session Hijacking“) je jedním ze skupiny útoků zaměřených na autentizační a identifikační mechanizmy aplikace. Vychází především z bezstavovosti, hlavního bezpečnostního problému platformy popsaného v kapitole 2.2.1. Cílem útoku je získání identity jiného uživatele. Útok je (v popsané podobě) irelevantní pro aplikace identifikující/autentizující uživatele jiným způsobem (například certifikátem).

### 4.2.1 Princip útoku a obrany

Hlavním bodem zájmu všech variací tohoto útoku je získání identifikátoru uživatele (někdy také „tokenu“), podle kterého jej server rozpoznává. Server nedokáže jednoznačně identifikovat uživatele na úrovni fyzické vrstvy (sítě), IP adresy ani jiné prvky pro to nedostačují (viz 2.2). Z tohoto důvodu si vypomáhá „uložením“ unikátního klíče na klientovi – v prohlížeči. Ten jej pak s každým požadavkem odesílá na server. Útočník, který úspěšně získá tento, ještě aktivní, klíč, může pracovat v aplikaci stejně jako napadený uživatel. Typickým cílem jsou přitom privilegovaní uživatelé a administrátorské účty.

Optimální, avšak většinou pouze teoretickou možností obrany je úplné vynechání identifikátoru z procesu. Útočník by pak neměl, jak se za jiného uživatele vydávat. Hlavním principem obrany jinak zůstává zabezpečení klíče na všech úrovních, jako jeho častá revalidace, krátké expirace, silné kryptograficky bezpečné generování klíčů, zamezení čtení klíče jinde, než v bezpečném úložišti (typicky cookies), implicitní nedůvěřování zaslaným klíčům apod.

### 4.2.2 Kategorizace

Kategorie: [D], [A.3], [B.1], [F]

#### *4.2.2.1 Severita, dopad*

| **Severita** | **B, Střední** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Účet |
| Odborný náhodný | B | Data, Účet |
| Odborný cílený | B | Data, Účet, Osoba |
| Profesionální cílený | B | Data, Služba, Účet, Osoba |

*Tabulka 4.2: Severita a dopad únosu relace, zdroj autor*

#### *4.2.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují zneužívají uživatelské účty, vyhrožují kompromitací uživatelů, poškozují osoby uživatelů. Dále riskují finanční postihy, žaloby a v závislosti na rozsahu i trestní řízení.

Vlastníci: ztrácí důvěru uživatelů, platí smluvní pokuty a penalizace za nezabezpečení účtů, kompromitaci klientů, v případě napadení privilegovaných účtů ztrácí kontrolu nad aplikací.

Třetí strany: platí za cílené poškození (reputace) konkurence, napadení konkrétních osob a jejich účtů, kompromitaci jejich dat.

### 4.2.3 Způsob prověření (provedení)

V závislosti na konkrétní implementaci je možné provést následující variace útoku:

- vytvoření identifikátoru hrubou silou,
- získání identifikátoru z neexpirovaného přihlášení,
- získání identifikátoru zasílaného v URL,
- získání identifikátoru jiným kanálem (napadení javascriptem, virem apod.).

Pokud vývojář vytvoří **vlastní mechanizmus generování** identifikátoru, jde často o snadno napadnutelný článek zabezpečení. Vestavěné funkce pro generování frameworku .NET zohledňují kryptografickou bezpečnost tokenu (dostatečnou náhodnost, entropii), využívají časový seed, dále tzv. Machine key (v machine.config daného serveru uvedený unikátní identifikátor stroje, který často využívá server IIS) a další techniky.

Napadení generovaného identifikátoru hrubou silou spoléhá na možnost testování velkého množství identifikátorů za sebou. Jak uvádí zdroj [18], ověření velkého množství kombinací znaků a čísel není se současnou výpočetní silou problém. Dalším faktorem je, že útočník relativně snadno rozpozná technologii, kterou server používá (hlavičky HTTP) a může otestovat běžné způsoby generování tokenů na vlastním stroji se stejnou technologií – lze tedy přesně replikovat odhadovanou funkčnost serveru a to včetně časového seedu (útok může prověřit všechny hodnoty v blízkosti času provedení – například kryptograficky nebezpečný System.Random využívá systémových hodin, jako výchozí seed s omezeným rozlišením v řádu desítek milisekund).

K získání klíče je také možné využít **chybného nastavení jeho expirace**. Chybnou implementací je použití:

- klíčů, kterým nevyprší platnost po době nečinnosti (nejkratší doba akceptovaná uživateli),
- klíčů, kterým nevyprší platnost v rámci extrémně dlouhé činnosti (tzv. „absolutní expirace“, tedy čas, za který vyprší klíč i v případě, že je aktivní, zdroj [17]) *[Oprava pro Úroveň 2 – Základní]*,
- klíčů, které nejsou zneplatněny během odhlášení uživatele.

Delší životnost klíče přináší bezpečnostní rizika s jeho uchováním. Primárně se jedná o nezkušenost uživatelů, kteří se přihlašují na veřejných místech. Zároveň zvyšuje šance útočníka, jelikož při úspěšném útoku na klíč, například pomocí distribuované sítě, má delší čas na reakci a pokračování v útoku. Zároveň zvyšuje pravděpodobnost úspěchu útoku hrubou silou, jelikož je v jeden okamžik více aktivních identifikátorů (méně jich expiruje).

Identifikátor může být také **chybně ukládán/odesílán** klientovi. Typicky se jedná o umístění tokenu v URL (tzv. „url rewritting“), případně ve skrytých polích formulářů („input type hidden“). Identifikátor umístěný v URL je například odesílán prohlížečem v hlavičce „referer“. Útočník tak může umístit odkaz na svou útočnou stránku – ta sleduje hlavičku referer a sbírá identifikátory uživatelů, které pak zneužívá. Platforma ASP.NET je náchylná k tomuto útoku, vzhledem k výchozímu chování tokenu FormsAuthentication, popsaného v kapitole 4.2.6.

Další možností je získání identifikátoru javascriptem nebo jiným, i newebovým útokem. Této kategorii se věnují kapitoly zaměřené na útoky vloženým javascriptem (XSS) a další.

Při identifikaci potencionálních míst Session Hijackingu je vhodné vyhledat: způsob vytváření identifikátoru a sílu jeho náhodnosti (případně, jestli neobsahuje nějaká nenáhodná data, jako login), způsoby expirace tokenu, expirace při odhlášení, nečinnosti, způsobu uchování tokenu (url, hidden pole, cookies – s označením „http only“, „secure“ apod.), prověření, jestli aplikace umožňuje přihlášení bez cookies – tzv. „cookieless“ mód apod.), ověření, jestli aplikace kombinuje token s další obranou (druhý token, validace ip adresy a další).

### 4.2.4 Způsob obrany

Běžným útokům tohoto typu je náchylná většinou jen vlastní implementace správy session vytvořená vývojářem – z tohoto důvodu je doporučeno (např. [37]) používat pokud možno vestavěnou funkcionalitu pro správu session ve frameworku. To znamená:

- nevytvářet vlastní schéma zabezpečení založené na manuálně vytvářených cookies, případně využití session,
- nevytvářet vlastní tokeny pro identifikaci.

Pro ASP.NET je nutné rozlišit správu session, identifikace anonymního uživatele a autentizačního tokenu forms. Tyto tři klíče generuje IIS nezávisle a ukládá je také nezávisle – zatímco session je uložena a validována v paměti serveru, forms a anonymní uživatel se na serveru neuchovávají. Dále forms jsou ve výchozím nastavení pro „cookieless“ mód nastaveny na „UseDeviceProfile“, zatímco session na „UseCookies“ (viz 4.2.6).

Základní obranou je tedy:

- využití vestavěných mechanizmů (FormsAuthentication),
- nastavení expirace tokenů na nejmenší možnou,
- zakázání „cookieless“ módu, pokud to je možné (aplikace nepodporuje velmi stará nebo „exotická“ zařízení),
- zamezení získání hodnoty tokenu na klientovi (nastavením „HttpOnly“ aj.).

### 4.2.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Web.Config<br>Sekce System.Web/authentication<br>(Útok na referer v T1_SecuBank/Controllers/AttackerController<br>a odpovídajícím view; Zranitelná funkce<br>T1_SecuBank/Controllers/MessageController) |
| **Oprava** | T1_SecuBank.Web.Config<br>Sekce System.Web/authentication |

#### 4.2.5.1 Ukázkové použití: zranitelné

Aplikace využívá FormsAuthentication pro autentizaci uživatele. Forms je ponecháno prakticky ve výchozím nastavení a není doplněno dalším mechanizmem ověření tokenu. Aplikace umožňuje uživatelům odesílat mezi sebou textové zprávy. Jelikož není možné vkládat HTML ani jiné formátování, jsou pro pohodlí uživatelů odkazy ve formátu „HTTP://xxx.xxx.xxx“ automaticky převáděny na tag „anchor“.

Běžná zpráva obsahující text „Ahoj, podívej se na http://www.moje-stranka.cz“. Je převedena na:

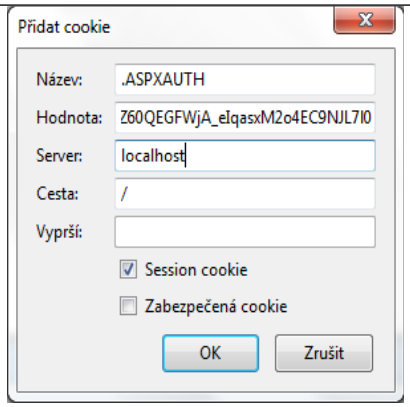
```
<p>Ahoj, podívej se na <a href="http://www.moje-stranka.cz">http://moje-stranka.cz</a></p>
```

Útočník může odeslat zprávu s odkazem na svou útočnou, na první pohled běžnou aplikaci.

Pokud se do aplikace SecuBank přihlásí uživatel ze zařízení, které IIS identifikuje, jako zařízení bez cookies (což může být chybně například tablet nebo mobilní telefon), je odesílán FormsAuthentication tiket v URL.


*Ilustrace 4.1: Hlavička referer odeslaná útočníkovi, která obsahuje Forms authentication tiket; prohlížeč: Google Chrome, zdroj autor*

Tato URL je pak odeslána na útočníkovu stránku v hlavičce „Referer“. Útočník pak může bez obtíží tento token zapsat do své cookie/URL a do aplikace se přihlásí.




*Ilustrace 4.2: Přidání cookie s hodnotou autentizačního tokenu Forms, prohlížeč: Mozilla FireFox, zdroj autor*

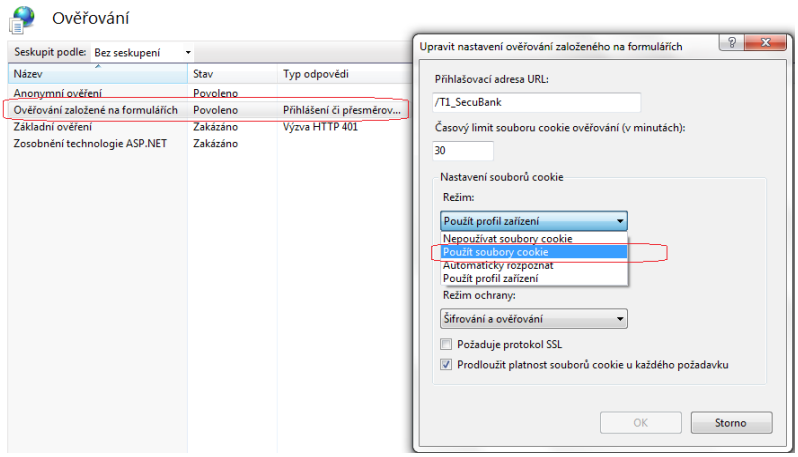
#### 4.2.5.2 Ukázkové použití: bezpečné

Aplikace využívá stále stejný mechanizmus, obrazovky a téměř stejnou konfiguraci. V konfiguraci je nastavena položka „cookieless“ na hodnotu „cookies“.

```
Konfigurace forms módu, umožňující popsaný session hijacking:
<authentication mode="Forms">
 <forms loginUrl="/T1_SecuBank" />
</authentication>

Konfigurace forms módu, neumožňující popsaný session hijacking:
<authentication mode="Forms">
 <forms loginUrl="/T1_SecuBank" cookieless="UseCookies" />
</authentication>
```

Stejnou konfiguraci je možné ovlivnit také ve správci IIS:




*Ilustrace 4.3: Konfigurace IIS a ověřování formuláři (FormsAuthentization) - nastavení cookieless módu na "UseCookies"*

Nyní není možné se do aplikace přihlásit bez cookies. Navíc hlavička referer útočníkovi neprozrazuje přihlašovací token.

### 4.2.6 Dodatečné informace

Server IIS standardně hostující ASP.NET aplikace podporuje automatické rozlišování klienta podle odeslané „User agent“ hlavičky. Pokud je zaslána hlavička, která prozrazuje, že klient nepodporuje ukládání cookies, ve výchozím nastavení přepíná do tzv. „cookieless“ módu, tedy odesílání identifikátorů (session, forms, anonymous) v URL. Pro ověření je možné změnit „User agent“ hlavičku na hodnotu „Generic Downlevel“ (zdroj [8]). Tato hlavička mj. způsobí odesílání tokenů v URL (ve výchozím nastavení). Výchozí nastavení pro FormsAuthentication tiket je „UseDeviceProfile“, tedy identifikace z hlavičky (zdroj [22]).

Podle [8] a dalších zdrojů ASP.NET MVC nepodporuje oficiálně „cookieless“ mód pro autentizaci (token v URL). Provedené testy autorem práce v ukázkové aplikaci však potvrzují, že je možné pomocí něj aplikaci napadnout.

## 4.3 Zranitelnost: Obcházení autorizace (parametry dotazu)

Obcházení autorizace (z angl. „Authorization Bypass“, jinak také „Direct object reference“), je jedním z mnoha útoků specializovaných na obcházení („bypass“) zabezpečení přístupu k vybraným zdrojům aplikace. Tato kapitola specificky pojednává o obcházení zabezpečení přístupu k datům pomocí parametrů requestu. Cílem útoku je získání přístupu k datům, která nejsou útočníkovi běžně v aplikaci přístupná. Zranitelnost je irelevantní pro aplikace, které neobsahují data omezená uživatelskými účty – tedy „všichni mohou vidět všechna data“.

### 4.3.1 Princip útoku a obrany

Primárním zaměřením útočníka je získání dat, která nemají být jeho účtu přístupná – jedná se tedy o přihlášeného útočníka (může se jednat např. o napadený účet běžného uživatele, viz 4.2).

Základním pilířem úspěchu je pro tento útok možnost získávat požadovaná data pomocí zadaných parametrů. Tvůrce aplikace umožní typicky uživateli výběr z několika datových položek (seznam), mezi kterými má zvolit jednu, jejíž detail chce zobrazit. Seznam těchto položek je omezen oprávněním uživatele. Uživatel zvolí položku identifikovanou jednoznačným identifikátorem (typicky číselným). Aplikace přijme identifikátor (parametr) a vrací požadovaný výsledek.

Druhým pilířem je pak možnost útočníka odeslat upravený požadavek na detail vybraných dat. Jelikož útočník může měnit prakticky jakýkoliv parametr dotazu na server (viz 2.2.2), může změnit identifikátor položky na takový, na který nemá oprávnění (nevidí jej v omezeném seznamu). Aplikace už neověří oprávnění uživatele na získání detailu dat, jelikož tvůrce předpokládá, že uživatel vždy přichází z omezeného seznamu, který je „bezpečný“.

Základním principem obrany je doplnění kontroly oprávnění na každý bod aplikace, který vrací jakákoliv data, která by měla být omezena uživatelským účtem, oprávněním. Rozšířenou variantou obrany je pak zamezení používání fyzických identifikátorů v uživatelském rozhraní – viz Způsob obrany.

### 4.3.2 Kategorizace

Kategorie: [D], [B], [A.2]

#### *4.3.2.1 Severita, dopad*

| **Severita** | **B, Střední** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Data |
| Odborný náhodný | C | Data |
| Odborný cílený | C | Data, Účet |
| Profesionální cílený | B | Data, Účet, Osoba |

*Tabulka 4.3: Severita a dopad obcházení autorizace, zdroj autor*

#### *4.3.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizená data, údaje z uživatelských účtů (jako e-mailové adresy), vyhrožují kompromitací aplikace, zveřejňují aplikační data (jako uživatelské transakce).

Vlastníci: ztrácí postavení na trhu, platí penalizace za kompromitaci klientů, ztrácí důvěru uživatelů, vykazují ztráty v případě zveřejnění uživatelských dat.

Třetí strany: platí za cílené poškození (reputace) konkurence, strojovou extrakci dat určitého typu, získávají konkurenční výhodu díky přístupu k neveřejným datům.

### 4.3.3 Způsob prověření (provedení)

Nejběžnějším příkladem tohoto útoku je změna parametru v URL. Aplikace nabídne uživateli seznam položek z jeho účtu. U každé položky je uveden odkaz na její detail – URL detailu vypadá typicky takto:

```
http://aplikace.com/product/detail/5
```

Číslo na konci adresy je typicky umělý identifikátor („Id“) položky v databázi, nejčastěji typu integer (INT, 32-bit) nebo long (BIGINT, 64-bit). Jelikož je otevření odkazu běžným GET („HTTP GET“) požadavkem, útočník může URL změnit na libovolné jiné „Id“. Náhodným výběrem nebo útokem hrubou silou může útočník získat data jiných uživatelů.

Přístup je možné získat i mimo běžné dotazy na detail. Například editační formulář se skrytým polem („input type hidden“) „Id“ umožňuje změnit toto pole a odeslat formulář se záměrnou chybou (nevyplněnou povinnou položkou). Server vyhodnotí chybu a pokusí se znovu načíst data k editaci – přičemž opět neprověří, jestli má uživatel na data právo.

Při identifikaci potencionálních míst pro Authorization bypass je možné hledat: libovolné seznamy položek s omezeným přístupem a zobrazování jejich podrobností (GET požadavky), formuláře pro filtrování nebo úpravu záznamů s možností zadání identifikátoru (POST požadavky), javascriptové dotazy na server (ajax) získávající například výběrová pole (číselníky) s omezením, identifikátory a klíče odesílané serverem ve skrytých polích, cookies apod. (je možné, že jejich změnou dojde k získání oprávnění, přístupu k datům apod.)

### 4.3.4 Způsob obrany

Řešení této zranitelnosti je značně závislé na architektuře aplikace. Především je nutné rozhodnutí, která vrstva (úroveň) je odpovědná za omezení oprávnění. Následující výčet popisuje řešení od základního až po pokročilé:

1. před vrácením dat (v MVC běžně controller) je ověřeno, zda na vybraný identifikátor má uživatel oprávnění (samostatný dotaz například do databáze),
2. bod 1 výše + na úrovni získání data z databáze je v rámci dotazu zakomponován i uživatelský účet (např. `"SELECT*FROM Account WHERE Id = 1 AND OwnerLogin = 'login'"`)
3. body 1, 2 výše + zamezení používání fyzických identifikátorů v uživatelském rozhraní na úrovni celé aplikace (použití umělých identifikátorů) *[Oprava pro Úroveň 2 – Základní]*
4. body 1, 2 výše + zamezení používání fyzických identifikátorů v uživatelském rozhraní na úrovni uživatelských session (použití umělých dočasných identifikátorů)

5. body 1, 2, 4 výše + cache omezení oprávnění pro optimalizaci výkonu

Zamezení používání fyzických identifikátorů znamená vytvoření umělé náhrady – například náhodné číselné řady, guid identifikátoru nebo hashe. Na úrovni aplikace jde pak o vytvoření jednoho identifikačního „seznamu“ a jeho použití k překladu identifikátorů v aplikaci. Zamezení použití klíčů na úrovni uživatelské session znamená vytvoření dočasného identifikačního „seznamu“ pro každého uživatele zvlášť.

Aplikace tedy v seznamu datových položek nevrací unikátní identifikátory datových skladů, ale umělý identifikátor vytvořený dočasně pro daný okamžik a daného uživatele. Tímto způsobem je pak obtížné nebo nemožné vyvolat vlastní dotaz na data pomocí změny parametru.

### 4.3.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Repository.Implementation.Insecure.BankAccountRepository Metoda GetAccountDetail<br>T1_SecuBank.Controllers.BankAccountController Metoda Detail |
| **Oprava** | T1_SecuBank.Repository.Implementation.Secure.BankAccountRepository Metoda GetAccountDetail<br>T1_SecuBank.Controllers.BankAccountController Metoda GetAccountDetail využívající Repository UserHasAccount |

#### *4.3.5.1 Ukázkové použití: zranitelné*

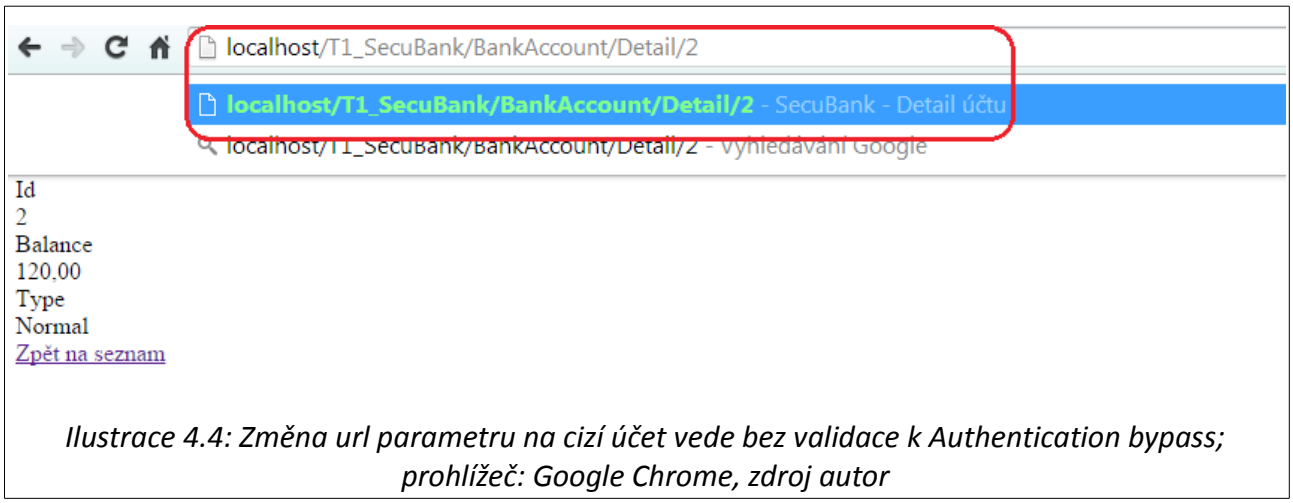
Aplikace zobrazuje seznam bankovních účtů uživatele. Tento seznam je omezen uživatelským účtem. Detail vybraného účtu však již nevaliduje, jestli se na něj dotazuje korektní uživatel, nebo ne. Je použit následující databázový dotaz:

```
return _context.Accounts.FirstOrDefault(a => a.Id == id);
```

Controller pak pouze volá tento dotaz:

```
...
  BankAccountModel account = logic.GetAccountDetail(id);
  return View(account);
...
```

Jelikož nedochází ke kontrole oprávnění na vybraný účet, je možné pomocí změny parametru v URL zobrazit i cizí účty:




*Ilustrace 4.4: Změna url parametru na cizí účet vede bez validace k Authentication bypass; prohlížeč: Google Chrome, zdroj autor*

#### 4.3.5.2 Ukázkové použití: bezpečné

Doplněním controlleru o validační logiku, jestli má uživatel oprávnění danou akci provést je základním krokem ověření – v tomto případě jde prakticky o „dvojité“ ověření téhož. V praxi je však v případě složitějších dotazů a komplexní logiky vhodné podobné základní ověření provést (typicky pro validaci mnohonásobné vazby 1:N nebo M:N):

```
if (!logic.CanUserDisplayAccount(id))
{
 //TODO: zpravu uzivateli o chybe a zalogovani podezrele akce
 return RedirectToAction("IndexSafe");
}

BankAccountModel account = logic.GetAccountDetail(id);
if (account == null)
{
 return RedirectToAction("IndexSafe");
}
return View(account);
```

V rámci samotného dotazu pak aplikace ověřuje, jestli má uživatelský účet na daný bankovní účet vazbu:

```
...//zkráceno pro stručnost
IQueryable<Account> q = from acc in accQ.Where(acc=>acc.Id == id)
            from ua in uaQ.Where(ua => ua.AccountId==acc.Id)
            from user in userQuery.Where(user => user.Id== ua.UserId)
            where user.Login == userName
            select acc;
return query.FirstOrDefault();
```

Optimální obranou pak zůstává vytvoření dočasných identifikátorů pro uživatelské rozhraní, ideálně pak unikátní pro každou relaci. Tímto způsobem pak krom tohoto zabezpečení útočník nemá možnost hádat hrubou silou databázové klíče ostatních uživatelů.

## 4.4 Zranitelnost: Přímý skok

Přímý skok (volně z angl. „Direct request“, jinak také zvaný „Forced browsing“, tj. „vynucené procházení“, autor) nebo „Function level access control“ („ověření přístupu na úrovni funkcí“, autor), je útok zaměřený na získání přístupu k funkčnosti, na kterou nemá uživatel oprávnění. Zranitelnost je irelevantní pro aplikace, které nemají žádné funkce nedostupné všem uživatelům (např. administrační funkce).

Zranitelnost je velmi obdobná zranitelnosti z kapitoly 4.3 (Authorization Bypass), jelikož se jedná primárně o úpravu parametru requestu na server s cílem získat přístup mimo své oprávnění. Primární rozdíl je krom způsobu obrany ve výsledku obou akcí, tedy v tom, co útočník získá. Zatímco získáním přístupu k **datům** se zabývá „Authorization Bypass“, tato kapitola popisuje získání přístupu k **funkci**. Zdroje porovnání [37], [5].

### 4.4.1 Princip útoku a obrany

Hlavním cílem útočníka je získání přístupu k funkci, která nemá být být jeho účtu přístupná. Nejedná se pouze o přihlášeného útočníka, jelikož k dané funkci se může pokusit přistoupit i anonymně.

Při útoku je snaha o odhadnutí cest ke skrytým, systémovým nebo administrátorským funkcím, k funkcím vyšších privilegií, systémovým souborům, skrytým logům, výpisům chyb apod. Tvůrce aplikace primárně spoléhá na fakt, že tyto funkce nejsou běžným uživatelům dostupné – viditelné. Při odhadování cest k těmto funkcím je útočníkovi nápomocen server odesílající svou identifikaci, verzi, platformu a další hlavičky. Jelikož aplikace často ověřují uživatelské oprávnění při zobrazování možných akcí, jsou tvůrci aplikací ve falešném pocitu bezpečí.

Primární obranou je vytvoření konzistentního modulu ověřování oprávnění napříč aplikací. Ověření mohou být umístěna i nad akcemi zdánlivě bez potřeby zabezpečení – robustnost takového řešení umožní autorizaci rozšiřovat, auditovat a snadno spravovat.

### 4.4.2 Kategorizace

Kategorie: [D], [A.2], [B.1]

#### *4.4.2.1 Severita, dopad*

| **Severita** | **B, Střední** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Data |
| Odborný náhodný | B | Data, Služba |
| Odborný cílený | B | Data, Služba |
| Profesionální cílený | B | Data, Systém, Služba |

*Tabulka 4.4: Severita a dopad přímého skoku, zdroj autor*

#### *4.4.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizená data, vyhrožují poškozením aplikace, brzdí nebo ukončují provoz aplikace pomocí administračních funkcí. Dále riskují finanční postihy a žaloby.

Vlastníci: ztrácí postavení na trhu, platí smluvní penalizace za nedostupnost aplikace, ztrácí důvěru uživatelů, ztrácí kontrolu nad aplikací.

Třetí strany: platí za cílené poškození (reputace) konkurence, poškození funkčnosti aplikace v době vrcholu konkurenčního boje.

### 4.4.3 Způsob prověření (provedení)

Základním ověřením je test, kdy je aplikace procházena s maximálními možnými oprávněními (typicky administrátorský účet) a zaznamenány všechny navštívené adresy (funkce). Následně je stejná cesta zopakována standardním neprivilegovaným účtem. Nakonec je stejná sada URL volána anonymním uživatelem.

Pokud aplikace reaguje konzistentně a bezpečně u běžných akcí, je vhodné ověřit všechny specificky administrační funkce – typicky má aplikace velmi dobře ověřená oprávnění na akcích pro běžné uživatele, ale nejprivilegovanější účty, administrační a pomocná rozhraní jsou zanedbána. Primárně proto, že je používají zkušení uživatelé, kteří taková „omezení nepotřebují“. Druhotně také proto, že tato rozhraní bývají zpravidla psána narychlo a jen s minimálním důrazem na uživatelské „pohodlí“.

Pro prověření aplikace je možné využít nástroje, jako například Apache JMeter ([6]), s jehož pomocí je možné vytvořit kolekci dotazů na vybrané funkce (URL). Tyto dotazy se neomezují pouze na typické „GET“ požadavky. Následně se pomocí jeho přihlašovacího modulu přihlásit při testu za různé uživatelské role.

### 4.4.4 Způsob obrany

Následující seznam popisuje hlavní body obrany:

- autorizační modul ověřující oprávnění uživatele na vybranou akci by měl kontrolovat veškeré akce aplikace (i zdánlivě veřejné),
- struktura oprávnění musí mít dostatečnou granularitu, zvyšující robustnost, snadnou správu a umožňující centrální mechanizmus autorizace,
- jednotlivé vrstvy aplikace mohou obsahovat svá vlastní autorizační pravidla (typicky autorizace na úrovni akce v aplikaci, byznys funkce v logické vrstvě, databázového schématu, databázové operace/uložené procedury)
- každá kolekce oprávnění (např. role) by měla být co nejmenší, tj. uživatel například na databázovém serveru dostane oprávnění na nejmenší možný rozsah činností (v kontrastu s přiřazením práv „do budoucna“).

### 4.4.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.AdminController |

| | |
|---|---|
| **Oprava** | T1_SecuBank.Controllers.AdminController (třída AdminSafeController)<br>T1_SecuBank.Logic.Secure.SecurityLogic Metoda GetUserPrincipal<br>T1_SecuBank.Global.asax.cs Metoda Application_AcquireRequestState |

#### *4.4.5.1 Ukázkové použití: zranitelné*

Aplikace obsahuje administrační rozhraní, které umožňuje provádět akce privilegovaným uživatelům. Na dané akci však není kontrolováno, kdo z uživatelů ji může provádět – je využit pouze základní atribut pro validaci přihlášení. Tvůrci aplikace spoléhají na fakt, že odkaz na administrační rozhraní není nikde uveden a neznalý uživatel jej nenalezne.

```
[Authorize]
public class AdminController : Controller
{
 public ActionResult Index()
 {
  ...
```

Libovolný přihlášený uživatel tedy může administrační rozhraní zobrazit. Nástroje pro prohledávání webových aplikací dokáží takto „neviditelná“ rozhraní odhalit například pomocí hledání hrubou silou.

#### *4.4.5.2 Ukázkové použití: bezpečné*

Vytvořením centralizované autority pro ověřování oprávnění je možné všechny akce aplikace opatřit určením požadovaných práv. Ukázková aplikace se omezuje jen na některé akce.

Aplikace obsahuje databázové tabulky `SecurityGroup`, `SecurityRole` a vazební `UserSecurityGroup`. V tomto schématu jsou uloženy uživatelské skupiny, které mají přiřazené role (různá oprávnění) a jsou navázány na uživatele. Každý uživatel má tak vybranou sadu „rolí“, jako například „Administrátor zobrazující úvodní administrační rozhraní“ (`Admin.Index`). Tato granularita umožňuje vytvářet podobné skupiny s nepatrně odlišnými oprávněními.

Při každém dotazu na aplikaci (tj. vyvolání libovolné akce) je ověřována identita uživatele (metoda `Application_AcquireRequestState`). V případě, že je uživatel přihlášen, dojde k získání jeho oprávnění z databáze a nastavení tzv. „`IPrincipal`“. Tento objekt může být ještě uchován v cache pro zlepšení výkonu. Následující úryvek obsahuje načtení oprávnění uživatele:

```
... roles = _repository.GetRolesForUser(user.Identity.Name);
... principal = new GenericPrincipal(user.Identity, roles.ToArray());
```

Oprávnění jsou získána následujícím dotazem:

```
...GetRolesForUser(string login)
//zkráceno pro stručnost
 var query = from u in userQuery.Where(u => u.Login==login)
  from ug in userGroupQuery.Where(ug => ug.UserId==u.Id)
  from r in roleQuery.Where(r => r.SecurityGroupId==ug.SecurityGroupId)
  select r.Code;
```

Každý uživatel je tedy ve chvíli vyvolání akce již ověřen, doplněn o oprávnění z databáze a je možné ověřit jeho autorizaci standardním atributem:

```
[Authorize(Roles = "Admin")]
public class AdminSafeController : Controller
{
 [Authorize(Roles = "Admin.Index")]
 public ActionResult Index()
 {
```

## 4.5 Zranitelnost: Odhalení uchovaných citlivých údajů

Odhalení citlivých údajů (z angl. „Sensitive data exposure“) je skupina zranitelností ohrožujících citlivá data aplikace, jako jsou hesla, čísla účtů a kreditních karet, emaily, zdravotní záznamy a další. Tato kapitola vymezuje užší skupinu těchto zranitelností, konkrétně odhalující **uchované** citlivé údaje – typicky v databázi. Zranitelnost je irelevantní pro aplikace neobsahující žádná citlivá data včetně hesel a e-mailů uživatelů.

### 4.5.1 Princip útoku a obrany

Citlivá data jsou zpravidla získána díky jejich nezabezpečenému uchování, například ve formě čistého textu. Útoky jsou primárně zaměřeny na data na serveru, nicméně může jít o data uchovaná v libovolné formě i v prohlížeči, zálohách dat a dalších. Je důležité zvažovat potenciálního útočníka uvnitř infrastruktury, správce apod., tedy nejen běžného uživatele aplikace samotné – typicky napadený účet administrátora.

Primárním zaměřením útoku je vyhledání a prověření síly/prolomení zabezpečení cílových dat. I zdánlivě zabezpečená data je možné napadnout – typicky použití známé nedostatečně složité kryptovací funkce. Je vhodné zvažovat fakt, že útočník má po získání zabezpečené sady dat téměř neomezené časové i výpočetní možnosti – nelze se spoléhat na omezené množství pokusů dekryptování, ani na omezené znalosti způsobu kryptování (typicky obskurní způsob kryptování/hashování). Má-li útočník k dispozici alespoň jeden vzorek známých dat (běžně své vlastní heslo do systému), je možné vysledovat obskurní způsob jeho zabezpečení. V tomto případě je tedy „bezpečnost obskurností“ (z angl. „Security by obscurity“) ještě méně efektivní. Navíc je možné během útoku využít techniky, jako „duhové tabulky“ (z angl. „rainbow tables“), tedy předpočítané všechny* kombinace možných řetězců (*ve vybraném rozsahu; kombinací je pochopitelně neomezené množství). „Rainbow tables“ jsou variací tzv. „slovníkových útoků“.

Prvním a nejefektivnějším způsobem obrany je neuchovávání citlivých dat – data, která aplikace nepotřebuje (resp. nemá) nejsou hrozbou. Hlavním způsobem je pak využití známých v době implementace bezpečných* hashovacích a kryptovacích algoritmů. Kritickými body pak zůstávají ty, kdy se tato data nacházejí mimo server – odeslaná na klienta, emailem, v tiscích, v integracích, webových službách apod.

*téma bezpečnosti těchto algoritmů je komplexní a je nad rámec této práce. Pro výběr vhodných algoritmů je možné využít zdroj [29] nebo [4].

### 4.5.2 Kategorizace

Kategorie: [C], [B.1], [D], [A]

#### 4.5.2.1 Severita, dopad

| **Severita** | **A, Vysoká** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Data |
| Odborný náhodný | B | Data, Účet |
| Odborný cílený | A | Data, Účet, Osoba |
| Profesionální cílený | A | Data, Účet, Osoba |

*Tabulka 4.5: Severita a dopad odhalení uchovaných citlivých údajů, zdroj autor*

#### 4.5.2.2 Ekonomické aspekty úspěšného útoku

Útočníci: obchodují odcizená citlivá data, uživatelské účty a hesla, vyhrožují kompromitací osob, zneužívají osobních citlivých dat. Dále riskují finanční postihy, žaloby a trestní řízení.

Vlastníci: ztrácí postavení na trhu, platí smluvní pokuty a penalizace za nedostatečnou ochranu dat, kompromitaci klientů, ztrácejí licence k provozu činnosti, jsou cílem trestních řízení ze strany státu za nedostatečnou ochranu osobních údajů, ukončují svou činnost.

Třetí strany: platí za cílené poškození (reputace) konkurence, odcizení dat s cílem jejich zneužití (například data pacientů, farmaceutický průmysl), vyhledávají osobní data konkrétních osob s cílem jejich kompromitace nebo poškození.

### 4.5.3 Způsob prověření (provedení)

Pro mnoho aplikací společný příkladem této zranitelnosti je uchovávání uživatelských hesel. Základním východiskem pro útočníka je přístup k databázi hesel (například díky zranitelnosti 4.1, nebo pouhému uhádnutím hesla administrátora). Pokud jsou hesla uchována v podobě čistého textu, útok úspěšně končí.

Pokud jsou hesla zabezpečena například pouze pomocí hashovacího algoritmu SHA1, je možné zaútočit na databázi hesel hrubou silou. Rychlejší variantou je pak napadení pomocí předpřipravených tabulek – tzv. „duhových tabulek“ (z angl. „rainbow tables“).

Pokud jsou v aplikaci uchovávána data vyžadující použití v „čisté podobě“, například čísla kreditních karet, účtů, zdravotní údaje apod., prověří útočník, zda jsou tato data šifrována a na jaké úrovni. Jsou-li ukládána v podobě čistého textu, útok úspěšně končí. Pokud jsou šifrována například na úrovni databáze, pak pouhý SQL dotaz vrací převedená čistá data. Pokud jsou data šifrována až na úrovni aplikace, pak je možné prověřit sílu použitého algoritmu.

**Útok hrubou silou a pomocí rainbow tables**

Typicky tedy útočník získá sadu hashovaných hesel a prověří je frekvenční analýzou a dalšími statistickými nástroji – nejčastěji se vyskytující hesla (jejich hash) pak může podrobit manuální zkoušce. Dále může provést vygenerování testovací sady hashů pomocí nejběžnějších algoritmů (případně známého algoritmu aplikace).

Ukázkovou základní funkci pro vytvoření všech kombinací pro hash SHA1 je možné najít ve zdroji: `T1_SecuBank.Infrastructure.Util.HashTableGenerator` metoda

`GenerateSimple` (je vhodné generovat omezené sady například maximální délky 5 z důvodů časové náročnosti). I takto jednoduchým a neoptimálním algoritmem může útočník snadno vygenerovat všechny kombinace SHA1 hashů (písmen a čísel) do délky vstupu 5 během minut na dnešním průměrném domácím počítači. Řada krátkých hesel je snadno prolomena. Dopočtení všech hashů číselných řad nebo nejběžnějších hesel je pak ještě snazší.

**Ostatní kanály**

Zabezpečená data je možné očekávat také v jiných zdrojích, například e-mailech (potvrzení přihlášení uživatele, změna hesla), tiscích (PDF výstup uživatelského profilu) nebo integracích (interní webová služba mezi aplikacemi na serveru poptává detail uživatele a získá i heslo). Heslo zasílané v kterémkoliv z těchto kanálů může být nezabezpečené (čistý text) nebo zabezpečené méně, než v primárním zdroji – aplikaci.

Dalším možným kanálem je automatické vyplňování položek v prohlížeči, cache a další – typickým útokem je pak javascript (viz 5.1), který v prohlížeči doplní neviditelná pole pro přihlášení (jméno, heslo) a vyvolá akci pro automatické doplnění.

Při identifikaci potencionálních míst pro odhalení uchovaných citlivých údajů je možné hledat: formuláře pracující s hesly (resp. citlivými daty), obnovu zapomenutých hesel (resp. dat), skrytá formulářová pole pro ověření hesla na klientovi, databázové sloupce uchovávající tato data a způsob jejich zpracování, využití citlivých dat jinde než v primárním místě užití, jako webové služby, e-mailová korespondence, zálohy databází, extrakční/vytěžovací procesy databází, analytické nástroje, integrační platformy a middleware a další. Dále také položky dočasně nahrazující funkci těchto dat – typicky způsob uchování dočasného klíče pro obnovu účtu (jde v podstatě o heslo), uchování odpovědí na otázky pro reset hesla a další. V neposlední řadě způsob předání těchto dat mezi klientem a serverem, případně jejich použití v prohlížeči – cache, automatické vyplňování, ukládání hesel.

### 4.5.4 Způsob obrany

Dokonalou obranu v tomto případě prakticky není možné zajistit – už z principu fungování hashovacích algoritmů. Za dostatečnou obranu je považována taková, kdy prolomení hesel zabere dostatečně dlouhou dobu – je tedy dostatečně nákladné. Primárním cílem obrany je zabránění použití známých chyb algoritmů, (snadnému) útoku hrubou silou a použití „rainbow tables“. Sekundárním cílem obrany je maximalizace nutných použitých prostředků pro odkrytí dat.

Základním prvkem takové obrany je použití tzv. „soli“ (z angl. „salt“), tedy použití dodatečného vstupu pro hash. Typicky lze využít identifikátory v databázi (id, login), datum (vytvoření, hashe) aj. Optimální je použití kryptograficky bezpečného a náhodného řetězce (jako RNGCryptoServiceProvider).

Následující seznam popisuje úrovně možných obranných mechanizmů zabezpečení hesel od nejjednodušších. Jsou vynechány varianty považované za zcela nedostatečné, jako je textová podoba dat, kryptování na úrovni databáze, jednoduché zakódování (base64 apod.) nebo vlastní obskurní mechanizmy:

- Použití hashovacího algoritmu společně se solí. Běžně rychlé algoritmy jako SHA1, MD5. Jejich rychlost v kontrastu umožňuje snazší útoky hrubou silou.
- Použití „pomalého“ hashovacího algoritmu společně se solí. Algoritmy jako PBKDF2 (v .NET

System.Security.Cryptography.Rfc2898DeriveBytes). Tyto algoritmy opakují vybraný hash se solí v mnohonásobných iteracích (typicky tisících). Díky tomu je útok hrubou silou výrazně zpomalován.

- Použití „pomalého“ a výpočetně náročného hashovacího algoritmu společně s integrovanou solí. Algoritmus bcrypt doporučovaný zdrojem [37] je pro .NET dostupný ve formě knihovny (nuget balíčku). Tento algoritmus vrací hash s integrovanou solí. Navíc obsahuje také generátor soli, u kterého je možné určit výpočetní složitost – je možné jej tedy škálovat v čase.

Dalším bodem zabezpečení dat jsou všechna místa, kde se mohou vyskytovat mimo běžně bezpečné úložiště. Hesla (ani jiná citlivá data) nesmí být nikdy zasílána žádným kanálem v podobě čistého textu. Optimálně nejsou zasílána nikdy žádným kanálem. Například namísto e-mailů pro obnovu hesla obsahujícím text hesla je možné zasílat dočasné identifikátory, kterými se uživatel přihlásí a heslo bezpečně změní. Důležité je, že takový token musí být stejně bezpečný, jako heslo samotné a navíc by měl podléhat expiraci.

V neposlední řadě je nutné zabezpečit heslo na klientském stroji – například zasíláním HTTP hlaviček pro zákaz cache (vč. proxy cache), zákaz automatického vyplňování polí (atribut „autocomplete“ standardu HTML5, [14]) a optimálně zákaz ukládání hesla prohlížečem (například generováním náhodného názvu polí jména a hesla).

### 4.5.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank, databázová tabulka User, sloupec Password<br>T1_SecuBank.Logic.Insecure.HomeLogic Metoda LogonUser<br>T1_SecuBank.Repository.Implementation.Insecure Metoda GetUserForLogin |
| **Oprava** | T1_SecuBank, databázová tabulka User, sloupec HashedPassword a PasswordSalt<br>T1_SecuBank.Logic.Secure.HomeLogic Metoda LogonUser<br>T1_SecuBank.Repository.Implementation.Secure Metoda GetUserForLoginWithHash<br>T1_SecuBank.Infrastructure.Util.CryptoProvider |

#### *4.5.5.1 Ukázkové použití: zranitelné*

Aplikace využívá běžné ověření jménem a heslem. Hesla jsou v databázi uchována v podobě čistého textu. Zadané uživatelské heslo je tedy zasláno přímo v rámci dotazu do databáze pro ověření shody:

```
User user = repository.GetUserForLogin(login, password);
```

Heslo je pak ověřováno dotazem uvedeným v kapitole 4.1.5.1. Hesla tedy nejsou nijak chráněna.

#### *4.5.5.2 Ukázkové použití: bezpečné*

Je využíváno ověření jménem a heslem. V databázi je uchováno heslo, jako `Base64` řetězec vytvořený hashovacím algoritmem `PBKDF2` nativně podporovaným ve frameworku .NET (jako třída `Rfc2898DeriveBytes`). Nejedná se tedy o nejvyšší doporučovanou úroveň z kapitoly Způsob obrany, nicméně jde o řešení přinášené samotnou platformou. Implementace tohoto algoritmu v .NET využívá jako podkladový algoritmus SHA1, což může být v některých případech nedostatečné nebo nedostatečně obecné. Níže je uveden způsob generování hashe.

```
public string GetHash(string password, string loginSalt)
{
 byte[] salt = Convert.FromBase64String(loginSalt);
 ... rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, salt);
 rfc2898DeriveBytes.IterationCount = 10000;
 byte[] hash = rfc2898DeriveBytes.GetBytes(20);
 return Convert.ToBase64String(hash);
}
```

Počet iterací s hodnotou „10 000“ je primárním způsobem zpomalení útočníka v případě útoku hrubou silou. Do budoucna je možné tedy sílu daného hashe zvyšovat změnou právě této cifry.

Dále je v databázi uchována „sůl“ (sloupec `PasswordSalt`), která je vytvářena jako `Guid (Guid.NewGuid())` a uložena jako `Base64` textový řetězec. Uživatelem zadané heslo společně s uchovanou „solí“ je použito k ověření přihlášení:

```
string loginSalt = repository.GetSaltForLogin(login);
...
CryptoProvider crypto = new CryptoProvider();
hash = crypto.GetHash(password, loginSalt);
user = repository.GetUserForLoginWithHash(login, hash);
```

### 4.5.6 Dodatečné informace

Dopad úspěšného útoku na aplikaci a získání databáze hesel je jen částí vzniklého problému – získané heslo často uživatel používá u svého e-mailového účtu (často uvedeného u hesla) i jiných aplikací. Snadno napadnutelná jednoduchá webová aplikace může otevřít přístup k jiným, kritickým a dobře zabezpečeným.

# 5 . Úroveň 2 – Základní

Tato kapitola se věnuje popisu Základní úrovně zabezpečení webových aplikací a zranitelností do ní spadajících.

Základní úroveň předpokládá splnění Náležité úrovně zabezpečení. Měla by být vyžadována, jako minimum každým zákazníkem, který aplikaci hodlá využívat pro svou ekonomickou činnost, uchování osobních údajů jiných osob *(podle zákona č. 101/2001Sb., o ochraně osobních údajů a o změně některých zákonů)*, nebo pro práci s jakýmikoliv údaji, které mohou způsobit finanční újmu nebo poškození reputace. Zahrnuje zabezpečení proti často se vyskytujícím typům útoků, které jsou sofistikovanější, než v náležité úrovni a vyžadují znalosti a čas útočníka. Druhou skupinou útoků této úrovně jsou velmi často se vyskytující a velmi běžné útoky. Jelikož jsou tyto aplikace často zdrojem obchodovatelných informací, zneužitelných v konkurenčním boji, ke kompromitaci firem nebo osob apod., mohou útoky být cílené, vycházející přímo či nepřímo od zainteresovaných iniciátorů útoku. Útoky základní úrovně mohou ale být také prováděny náhodnými amatéry, případně v rámci hromadných plošných útoků na Internetu.

Tuto úroveň by měla splňovat každá netriviální aplikace obsahující neveřejná uživatelská nebo firemní data s aktivními uživateli – typicky intranetové aplikace pro správu lidských zdrojů, základní účetní a finanční aplikace, výkaznictví.

Aplikace nesplňující plnohodnotně tuto úroveň (a všechny předcházející), s výjimkou zranitelností nerelevantních pro typ aplikace nebo platformu, je nutné považovat za nebezpečné pro použití ve: vnitrofiremním prostředí (typu ERP, CRM), veřejnou aplikaci pro sdílení obsahu (CMS, Wiki, Chat, Fórum), aplikace státní správy (všech typů).

Mezi typické zranitelnosti na této úrovni patří různé variace podsunutí skriptů, zneužití nekonzistentní práce s parametry, pokročilá správa přístupu aj.

## 5.1 Zranitelnost: Podsunutí skriptů

Podsunutí skriptů (volně z angl. „Cross-Site Scripting“, zkráceně „XSS“), nebo také „skriptování napříč sítí“ (zdroj [2]) je skupina technik, jejichž cílem je interpret skriptovacího jazyka v prohlížeči uživatele. Tato kapitola se věnuje podsunutí skriptů jazyka javascript. Zranitelnost je irelevantní pro aplikace, které žádný ze vstupů uživatelů nevrací zpět na výstupu v žádné formě.

### 5.1.1 Princip útoku a obrany

Obdobně jako v případě vložení SQL (4.1.1), první podmínkou pro úspěšný útok tohoto typu je možnost útočníka vložit vstupní řetězec do vstupu tak, aby byl následně na výstupu interpretován jako kód skriptovacího jazyka. Pro splnění této podmínky je obvykle nutná možnost vkládání textových řetězců a speciálních znaků. Není-li explicitně prohlížeč upozorněn, že daný úsek výstupu je uživatelský vstup (typicky zakódováním speciálních znaků), může jej interpretovat jako součást zdrojového kódu.

I velmi striktní validace na serveru může být nedostatečnou obranou, ačkoliv je-li vstup validován, například jako číslo (regulárním výrazem, parsováním nebo podobným), útočníkovi obvykle nezbývá mnoho prostoru. Hlavním principem obrany je zakódování nedůvěryhodného výstupu tak, aby nebyl interpretován jako skript, ani jako součást DOM („Document Object Model“, HTML).

### 5.1.2 Kategorizace

Kategorie: [A], [C], [D], [F]

#### *5.1.2.1 Severita, dopad*

| **Severita** | **A, Vysoká** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | B | Data, Služba |
| Odborný náhodný | B | Data, Služba, Účet |
| Odborný cílený | A | Data, Služba, Účet |
| Profesionální cílený | A | Data, Služba, Účet, Osoba |

*Tabulka 5.1: Severita a dopad Podsunutí skriptů, zdroj autor*

#### *5.1.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizená data, uživatelské účty, vyhrožují kompromitací aplikace, znehodnocují celý ekonomicko-společenský účel aplikace za úplatu, zneužívají uživatelů k napadání třetích stran, šíření virů, obsahu. Dále riskují finanční postihy, žaloby a trestní řízení.

Vlastníci: ztrácí postavení na trhu, platí smluvní pokuty a penalizace za ztrátu dat, kompromitaci klientů, přicházejí o klientelu v důsledku poškození účelu aplikace, jsou spojováni s dopady útoku (z pohledu uživatele útok provádí aplikace) ukončují svou činnost.

Třetí strany: platí za cílené poškození (reputace) konkurence, odcizení dat s cílem průmyslové špionáže, napadení uživatelů, zneužití prostředků prohlížečů, platí za šíření virů, dat.

### 5.1.3 Způsob prověření (provedení)

**Rozdělení „XSS“**

„XSS“ lze obecně rozdělit podle zdroje, který útok „vrací“ na (zdroje [2], [29]):

- Uložené (z angl. „stored“),
- odražené (z angl. „reflected“).

Obě tyto varianty jsou pak kombinovány se způsoby jejich využití, kterými jsou:

- Tradiční využití (serverové),
- založené na DOM (z angl. „DOM-based“),
- „čisté“ založené na DOM (z angl. „pure DOM-based“).

**Uložené „XSS“** využívá úložiště dat na straně serveru k uchování útočného řetězce, který je vracen prohlížeči. Tam je typicky vložen přes odeslání formulářových dat, parametr URL nebo ajax. **Odražené „XSS“** pak využívá chyby aplikace, kdy je zadaný vstupní řetězec vracen zpět uživateli (není uložen) – ať už jde o URL parametr, formulářová data nebo cookie.

**Tradiční serverové využití** pak spočívá v odeslání plného požadavku na server a obdržení plnohodnotné stránky včetně vloženého řetězce (uloženého nebo odraženého). **Využití založené**

**na DOM** spoléhá na chybu, kdy je vstup uživatele zpracován javascriptem – typicky tak, že vstup odešle na server a očekává návratové HTML zpracované serverem (uložené nebo odražené), které pak vykreslí do stránky. **Využití „čistě" založené na DOM** je pak modifikací předchozího s tím rozdílem, že javascript nekomunikuje se serverem, ale pracuje s lokálními prostředky – typicky tak, že data z URL přímo zpracuje javascriptem do HTML zdroje (odražené). S příchodem HTML 5 pak „čistá" varianta této techniky nabírá na váze díky využití „HTML 5 Local storage" („lokální úložiště"), lokálních databází, tedy v podobě „pure DOM-based stored XSS".

**Příklad základního „XSS"**

Základní příklad velmi běžného tradičního uloženého vložení skriptu je následující. Aplikace obsahuje formulář, kam uživatel napíše textovou zprávu, kterou server uloží a vypisuje v seznamu nových zpráv – aplikace typu „fórum". Běžná textová zpráva se vypíše například takto:

```
<div><p>Ale kdo bude hlídat hlídače?</p></div>
```

Útočník se může pokusit vložit kód, který se následně bude interpretovat, například:

```
Já jsem hlídač. <script type='text/javascript'>alert(„XSS");</script>
```

Takový vstupní řetězec by byl v prohlížeči interpretován a javascript by byl spuštěn.

**Výchozí problémy při útoku na platformu ASP.NET**

Na platformě ASP.NET by tento útok už při výchozím nastavení aplikace **nebyl úspěšný** díky výchozím filtrům – více informací v popisu obrany, 5.1.4. Tato obrana však není samostatně zcela dostačující a opomíjí například problémy způsobené skládáním/spojováním několika vstupů od uživatele do jednoho výstupu, vykreslování chybných vstupů prohlížeči aj. Navíc v předcházejících verzích byla mnohokrát nalezena metoda, jak filtr obejít.

Například následující vstup uživatele prochází filtrem a v běžných prohlížečích je považován za chybný HTML vstup (je tedy ignorován). Prohlížeč **Internet Explorer 9** jej však interpretuje, jako běžný element (inline podobně jako *„span"*), zdroj [11]:

```
<%tag onmouseover="alert('xss')">hover here
```

Takovýto útok úspěšně obchází první bariéru ASP.NET – výchozí filtr. Druhou implicitní vrstvou obrany ASP.NET (MVC) proti XSS je **automatické kódování výstupu**. Takovýto výstup je nebezpečný v případě, že byl vypsán jako důvěryhodný (tzv. „raw"), vykreslen pomocí `new System.Web.HtmlString` (obdoba „raw"), je chybně využit při vykreslování tzv. „helperem" nebo je vykreslen přímo do výstupu (např. „`__razor_helper_writer`"). Záleží tedy na důslednosti použití bezpečných možností frameworku. Druhým nebezpečným použitím je vykreslení vstupu javascriptem, typicky pomocí metod `innerHtml`, nebo knihoven (jako jQuery).

Poslední běžnou překážkou úspěchu „XSS" jsou **filtry prohlížečů**. Tyto filtry jsou převážně účinné, jako obrana proti „odraženým" („reflected") typům skriptů, jelikož sledují odeslaný požadavek prohlížeče (GET/POST) a návratový výstup serveru. Pokud se některá část „odrazila" (například parametr v URL) a je vykreslena v HTML, přičemž interpret javascriptu se snaží daný úsek kódu interpretovat, je zablokován. Jelikož kontrola probíhá „sémanticky", tj. prohlížeč se nesnaží pomocí pravidel „odhadnout", jestli se kód omylem spustí nebo ne, ale přesně ví, jestli daný kus kódu hodlá interpret spustit nebo ne, není možné zde využít techniky pro obcházení

regulárních výrazů apod.

Běžné techniky obrany musí aplikace odebrat ve chvíli, kdy **je nutné umožnit uživatelům vkládání formátovaného textu** – v danou chvíli je ASP.NET filtr vypnut (typicky pomocí atributu „AllowHTML“ na vybrané vlastnosti třídy nebo ještě méně bezpečně pomocí „ValidateInput(false)“ atributu na dané akci MVC controlleru). V tu chvíli do aplikace lze vkládat prakticky libovolný vstup a cílem je primárně obcházení enkódovacích technik a využití chyb.

**Techniky obcházení zabezpečení**

Technik pro obcházení zabezpečení vytvořeného vývojářem je široké spektrum. Seznam níže uvádí pouze příklady obcházení běžných filtrů. Pro podrobnější informace o možných technikách, variantách útoků a příkladech z praxe je možné využít zdroje [2], [29]:

- Pokud je blokován výraz „<script>...</script>“ a je odstraňován, je možné jej obejít násobným vložením:
  - Neúspěšný útočný řetězec: `<script>alert("XSS")</script>`
  - Úspěšný řetězec: `<scr<script>ipt>alert("XSS")</scr</script>ipt>`
  - Obdobně je možné obcházet jiné regulární výrazy apod., pokud nejsou aplikovány rekurzivně/cyklicky.
- Pokud lze vkládat vlastní elementy, ale skripty jsou úspěšně zakázány, je možné využít např. iframe:
  - Neúspěšný útočný řetězec: `<script>alert("XSS")</script>`
  - Úspěšný řetězec:
    `<IFRAME SRC="javascript:alert('XSS');"></IFRAME>`
- Pokud jsou blokovány vkládané elementy, aplikace často vkládá předvybranou hodnotu do polí „input“ do atributu „value“.
  - Element, na který je snaha zaútočit:
    `<input type="text" value="@Hodnota" />`
  - Úspěšný řetězec: `" onMouseOver="alert('XSS')`
  - Tento útok předpokládá chybu vložení atributu javascriptem (DOM, viz níže) nebo chybu na serverové straně (HTML.Raw, string.Format apod.)
- Pokud je blokováno vložení skriptu do atributů HTML elementu pomocí tzv. „javascript encoding“ (tj. zakódování spustitelného kódu javascriptu), je možné využít například kódovaný vstup unicode (jedná se o „XSS založené na DOM“)
  - Element, na který je snaha zaútočit: `<a id="a" href="#">Test Me</a>`
  - Neúspěšný útočný řetězec:
    `document.getElementById("bb").setAttribute("onclick", "alert(1)")`
  - Úspěšný řetězec:
    `document.getElementById("bb").setAttribute("onclick", "\u0061\u006c\u0065\u0072\u0074\u0028\u0031\u0029")`

- Metoda setAttribute pro atributy, ve kterých je očekáván exekuční kód automaticky provádí implicitní konverzi z mnoha formátů a mimo jiné i „unicode encoding“ (např. onClick, onMouseOver, onBlur, onError – obrázků, aj.).

**Možná využití XSS a scénáře útoku**

Výše uváděné příklady se zabývají pouze způsobem vložení skriptu do výstupu a obcházení nativní obrany platformy. Výsledný útok „`alert('XSS')`“ je pouze demonstrační. Pokud je útočník schopen tento skript vložit do výstupu jiným uživatelům, je velmi pravděpodobně schopen provést některé z následujících scénářů:

- Únos relace („session hijack“), získání účtů, hesel a citlivých údajů (viz Zranitelnost: Únos relace),
- obcházení obrany proti padělání požadavků (viz 6.1), tedy odesílání požadavků za uživatele,
- kompletní převzetí kontroly nad uživatelským prohlížečem, nahrávání trojských koní a podobných přímo do systému uživatelů a další,
- zaútočit na intranetovou aplikaci státní správy zabezpečené pomocí tzv. „air gap“ (tj. fyzicky oddělené od Internetu), propašovat skript do Internetové aplikace, která pomocí CSRF („Cross Site Request Forgery“) nakazí intranetové méně zabezpečené aplikace, ukrývá data v HTML5 storage a ve chvíli připojení na Internet data odesílá na C&C server.

Při identifikaci potenciálních míst pro „XSS“ je vhodné hledat: primárně jakýkoliv vstup aplikace umožňující vkládání formátovaných textů (WYSIWYG editory, „textarea“ elementy, textová pole), formuláře se vstupem umožňujícím vkládat jakékoliv speciální znaky, zástupné symboly apod., možnosti vkládání/výběru URL nahraných obrázků do obsahu (např. náhledový obrázek), `ajaxové` dotazy na server a práci s přijatým asynchronním HTML ze serveru, práci javascriptu s DOM na základě parametrů a hodnot od uživatelů (vykreslení pomocí `innerHtml`, nebo `jQuery.html`), jakoukoliv činnost javascriptu s URL, parametry a objekty jako „`window.location`“. V kódu aplikace ASP.NET MVC pak lze typicky hledat místa použití `@Html.Raw`, `new System.Web.HtmlString`, `string.Format` v rámci vykreslování `View`, vlastní `helper`, případně využívání přímého zápisu do výstupu (pomocí `__razor_helper_writer`, `helper context writer` apod.).

### 5.1.4 Způsob obrany

Prvním bodem obrany je využití veškerých možných integrovaných obranných mechanizmů frameworku – pro ASP.NET MVC primárně vstupní validátor zabraňující vložení skriptu a výchozí zakódování jakéhokoliv výstupu vkládaného pomocí vykreslovacího jádra „Razor“. Druhým bodem je pak ošetření veškerých vstupů a výstupů – optimálně tak, aby nebylo možné využít žádné speciální znaky. Text vkládaný do prvků HTML musí být kódován jak pomocí HTML enkódování (speciální znaky HTML), tak pomocí javascript-enkódování (speciální znaky javascriptu). Na výchozí ochranné prvky se nedá spolehnout primárně proto, že se snaží obecně ošetřit vstupy, jejichž použití v aplikaci následně nemohou znát – chybně naprogramovanou aplikaci pak nelze ochránit. Druhotně také proto, že v minulosti byly odhaleny postupy, jak obranu obejít a pravděpodobně takové slabiny existují i dále.

Následující seznam popisuje obranné mechanizmy od základních k pokročilým, které by měly být optimálně všechny kombinovány:

- Integrovaná obrana frameworku (validace vstupu, kódování HTML výstupu, atributů a javascriptu),
- nepoužívání nebezpečných rozhraní pro framework (`Html.Raw`, `new HtmlString`, `string.Format`, `__writer` apod.),
- zákaz veškerého HTML vstupu, je-li to možné (tj. explicitní validace a zákaz speciálních znaků, např. regulárním výrazem „`^[a-zA-Z0-9 ]*$`“, případně méně striktního, parsováním vstupních čísel),
- omezení HTML formátování pomocí explicitního „white-listu“, případně použití speciálního náhradního formátu, jako např. „BB code“ (tj. umožnění např. jen elementu `<b>`),
- snížení šance pro „DOM-based“ a „pure DOM-based“ skriptování minimalizací asynchronních operací javascriptu s DOM, prací s URL a parametry, lokální databází HTML5 a omezení použití rizikových rozhraní javascriptu, jako jsou funkce `eval`, `setAttribute`, `window.location`
- využití nových obranných mechanizmů HTML a HTTP, jako je „CSP“ („Content Security Policy“, „bezpečnostní politika obsahu“, autor):
  - „CSP“ umožňuje prohlížeči zaslat HTTP hlavičku (případně `<meta>` tag), která definuje politiky spouštění javascriptu ve stránce,
  - je možné například říci, že se nesmí spouštět žádné skripty vložené přímo do HTML (tj. lze spouštět pouze načtené skripty pomocí atributu „`src`“, nebo dokonce jen načtené pomocí „`src`“ a umístěné v sekci `<head>`),
  - dále je možné definovat bezpečné zdroje skriptů (odkazy na složky serveru), nebo URL adresu akce (ajax POST akce controlleru MVC), na kterou prohlížeč zasílá identifikované reporty o prolomení XSS obrany,
- povýšení klientů na nejnovější verze moderních prohlížečů, které obsahují integrovanou ochranu proti mnoha variacím „odražených XSS“ a „XSS založených na DOM“ (jako Internet Explorer 9+, Mozilla Firefox, Google Chrome).

### 5.1.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.MessageController Metody SendMessage, GetMessage<br>T1_SecuBank.Views.Message.SendMessage |
| **Oprava** | T1_SecuBank.Controllers.MessageController Metody SendMessageSafe, GetMessageSafe<br>T1_SecuBank.Views.Message.SendMessageSafe |

#### *5.1.5.1 Ukázkové použití: zranitelné*

Aplikace umožňuje odeslat zprávu uživateli (v ukázce se jedná pouze o fiktivní zprávu fiktivnímu uživateli). Akce `Controlleru` (`MessageController`) přijímá parametr „message“ obsahující text zadaný uživatelem. Tato akce **není** „odkryta“ (otevřena libovolnému uživatelskému

vstupu) atributem `[ValidateInput(false)]`. Zároveň není použit na první pohled nebezpečný výstup pomocí `Html.Raw` nebo obdobný (viz níže). Zdánlivě je tedy aplikace zabezpečena nativním filtrem vstupu a výchozím zakódováním výstupu.

```
//[ValidateInput(false)]
public ActionResult SendMessage(string message)
{...
```

Po přijetí požadavku akce „uloží“ vstup a vrací stránku, na které javascriptová funkce volá `ajaxovou` akci na serveru (`GetMessage`). Tato akce vrací „uložený“ text. Javascript jej pak vykreslí standardní cestou do DOM. Následující úryvek obsahuje část této akce:

```
[HttpPost]
public JsonResult GetMessage()
{
 return new JsonResult
 {
  Data = ...data,//vraci ulozeny retezec
```

Úryvek níže obsahuje javascritový kód volající na server (za použití knihovny jQuery):

```
... $.ajax({
        type: "POST",...
        success: function (data) {...
            container.html(data);...
```

V předpřipraveném formuláři je uveden útočný vstup, který obchází nativní ochranu a pro prohlížeč Internet Explorer 9 (a po úpravě i nižší) způsobí spuštění podvrženého skriptu. Následující úryvek obsahuje text, který obchází filtr vstupu:

```
Toto je naprosto zásadní sdělení! <%tag onMouseOver='[XSS]'
style='position:absolute;width:100%;height:100%;top:0;left:0'>Zobrazit
podrobnosti
```

Tento vstup je vykreslen jako prvek překrývající celou stránku aplikace, který spustí požadovaný útok po najetí kurzoru myši. Následující kód je uveden v atributu „`onMouseOver`“. Text je ponechán záměrně neformátován. V praxi by mohl být navíc zakódován pomocí `base64` kódování a dekódován nativní javascriptovou funkcí `atob`:

```
alert(\"Úspěšný útok na uživatele \" + $(\".login-display
strong\").text()); $(this).unbind(\"mouseenter mouseleave\");$
(this).attr(\"onMouseOver\", \"\"); (function(){var tag = $(\"#message-
container\").children(); var c = 1; setInterval(function()
{tag.css(\"background-color\", !c ? \"green\" : \"red\");c=!c; },
1000)})();
```

Uvedený kód vypíše hlášku „Úspěšný útok na uživatele“, vyhledá jméno uživatele na stránce, vymaže svůj kód ze stránky a spustí časovaný „útok“, který každou sekundu přebarví obsah stránky střídavě na červenou a zelenou barvu. Tento zdánlivě neškodný útok však naznačuje, že útočník má pod kontrolou veškerý obsah v prohlížeči.

Důležitým faktem o tomto útoku je, že je proveditelný na zcela běžně zabezpečených profesionálních aplikacích na platformě ASP.NET MVC.

#### *5.1.5.2 Ukázkové použití: bezpečné*

Zabezpečená část aplikace využívá téměř totožný kód. Akce pro příjem zprávy totožně ukládá odeslaný text. Obsahuje navíc pouze ukázku možné validace pomocí regulárního výrazu, která však není v tomto případě jako obranný prvek využita:

```
...Regex.IsMatch(message, @"^[a-zA-Z\p{L}0-9 ,.!?_\n-]*$");
```

Pro takový regulární výraz je klíčové uvedení začátku a konce řetězce (znaky „^" a „$"), které značí, že **celý** vstup musí splňovat daný výraz.

Akce je explicitně označena atributem `[ValidateInput(true)]`, což ve výchozím nastavení ASP.NET není nutné. Klíčovým bodem obrany je akce `GetMessageSafe`, ve které dochází k ošetření výstupu (zakódování pro javascript):

```
public JsonResult GetMessageSafe()
{...
 result = HttpUtility.JavaScriptStringEncode(...data);
 //result = HttpUtility.HtmlEncode(result);//lze pripadne zaradit
 return new JsonResult
 {
  Data = result,
```

Takto ošetřený výstup je bezpečně vložen do stránky následovně (úryvek výstupu):

```
Toto je naprosto zásadní sdělení! \u003c%tag onMouseOver=\u0027
alert(\"Úspěšný útok ...
```

**Zabezpečení pomocí CSP**

Jak uvádí kapitola 5.1.4, pro moderní prohlížeče je možné využít standardu hlaviček CSP *[Oprava pro Úroveň 3 – Standardní]*.

Akce `SendMessageSafe` je opatřena atributem `ContentSecurityPolicyFilter`, který není nativní součástí ASP.NET MVC. Tento filtr vkládá do dané odpovědi hlavičku omezující možnosti javascriptu, použití zdrojů aj. (zdroj autor).

```
public class ContentSecurityPolicyFilterAttribute:ActionFilterAttribute
{
 public override void OnActionExecuting(... filterContext)
 {...
  filterContext.HttpContext.Response.AppendHeader(
   "Content-Security-Policy",
 "default-src 'none'; script-src 'self'; connect-src 'self'; img-src
'self'; style-src 'self' 'unsafe-inline';");...
```

Uvedená hlavička povoluje pouze lokální skripty z aplikace, zakazuje použití tzv. „inline" skriptů ve stránce i elementech, zakazuje použití některých funkcí (`eval`) a další. Z tohoto důvodu je upraven skript, který se dotazuje na server tak, že je umístěn v separátním souboru:

```
<script type="text/javascript" src="@Url.Content("~/...message.js")">
```

Navíc tento skript využívá nativní metody javascriptu pro vložení obsahu:

```
...$.ajax({
      type: "POST",...
      success: function (data) {...
            container.innerHTML = data;
```

I v případě, že by u takto ošetřené stránky byl povolen libovolný vstup (bez nativní validace), ke spuštění „inline“ stylu od útočníka by nedošlo (byl by prohlížečem zastaven). Dále je možné specifikovat pro tento požadavek unikátní identifikátor „inline“ skriptů autorů aplikace (hodnota „`nonce`“), případně `hash`, které zajistí, že mohou být spuštěny skripty ve stránce.

Nevýhodou tohoto řešení je stále relativně nízká podpora ze strany prohlížečů (především Internet Exploreru) a další problémy popsané v kapitole 2.3, Problematika standardů webu.

### 5.1.6 Dodatečné informace

Tomuto typu útoků a tématu obecně se velmi podrobně věnuje zdroj [2].

## 5.2 Zranitelnost: Nevalidovaná přesměrování a předání

Nevalidovaná přesměrování a předání (z angl. „Unvalidated redirects and forwards“) je zranitelnost primárně využívaná, jako nástroj pro zvýšení důvěryhodnosti útočníka při úroku, pomocí nezabezpečeného přesměrování z důvěryhodné aplikace. Zranitelnost je irelevantní pro aplikace, které nepoužívají přesměrování ani předání uživatelů (převážně statické).

### 5.2.1 Princip útoku a obrany

Úspěšný útok zneužívá možnosti „odrazit“ požadavek na přístup k útočné aplikaci o jinou dostatečně důvěryhodnou. První předpokladem je, že útočník má možnost předložit uživateli hypertextový odkaz nebo obdobný prostředek, který na první pohled směřuje na uživateli známou a důvěryhodnou aplikaci. Tento odkaz může být šířen kanály třetích stran. Uživatel, vzhledem k důvěryhodnosti cílené aplikace, neklade na kontrolu odkazu dostatečný důraz, případně nemá ani možnost zjistit, že výsledný dotaz nevede ke známé aplikaci. Po aktivaci odkazu dojde k zavolání důvěryhodné aplikace s tím, že ta uživatele přesměruje na připravenou útočnou aplikaci. Ta může typicky imitovat vzhled cílené aplikace, nahrávat škodlivý kód a zneužívat zranitelností v prohlížečích, nebo se snažit o XSS a CSRF útoky.

Převážně teoretickou možností je zamezení používání všech přesměrování v aplikaci. Jelikož dnešní typické webové aplikace přesměrovávají uživatele prakticky po každém odeslání formuláře nebo snaze o přihlášení, a je dokonce obecně přijímaným doporučením po formulářových akcích uživatele přesměrovat, u většiny aplikací to není možné. Hlavním způsobem obrany je minimalizace přesměrování a předání závislých na parametru požadavku a v případě nutnosti takového parametru pak jeho striktní validace a omezení na doménu dané aplikace.

### 5.2.2 Kategorizace

Kategorie: [A.3], [A.2], [B.1]

#### 5.2.2.1 Severita, dopad

| **Severita** | **B, Střední** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Služba, Osoba |
| Odborný náhodný | C | Služba, Osoba |
| Odborný cílený | C | Data, Služba, Účet, Osoba |
| Profesionální cílený | B | Data, Služba, Účet, Osoba |

*Tabulka 5.2: Severita a dopad nevalidovaných přesměrování a předání, zdroj autor*

#### 5.2.2.2 Ekonomické aspekty úspěšného útoku

Útočníci: za úplatu přesměrují uživatele aplikace na reklamní nebo útočné servery, získávají důvěru uživatelů pro další útoky. Dále riskují finanční postihy, žaloby a trestní řízení.

Vlastníci: ztrácí důvěru uživatelů, jsou spojováni s útokem a službami, na které jsou uživatelé přesměrováni, jsou cílem dalších útoků.

Třetí strany: platí za cílené poškození (reputace) konkurence, za zobrazení reklam uživatelů, získávají klientelu na základě přesměrování uživatelů na jejich služby důvěryhodnou aplikací.

### 5.2.3 Způsob prověření (provedení)

Nejběžnějším příkladem je přesměrování v rámci přihlašovacího formuláře. Obdobně frekventovaný příklad je přesměrování při zpracování „průvodce“ (angl. „wizard“). Aplikace obdrží první požadavek od klienta, přesměruje jej na speciální umístění (například první krok průvodce) a do parametru v URL zapíše, kam má uživatel pokračovat. Typická url pro pokračování vypadá schématicky následovně:

```
http://aplikace/akce?NasledujiciUrl=%2fAdmin%2fSeznamHlidacu
```

*znaky „%2f“ jsou zakódovaná dopředná lomítka („/“), jelikož jde o speciální znak URL

Uživatel provede požadavek na tuto adresu a je přesměrován na zadanou „`NasledujiciUrl`“. Běžný dotaz typicky přesměruje uživatele relativně v rámci aplikace. Útočník může takové přesměrování využít například umístěním následujícího odkazu na sociální síť:

```
http://aplikace/akce?NasledujiciUrl=%68%74%74%70%3A%2F%2F%61%70%49%69%6B%61%63%65%2F%61%6B%63%65
```

Zdánlivě tento odkaz vzbuzuje důvěru díky cestě k dané aplikaci a doméně. Parametru, obzvláště takto zakódovanému nevěnuje uživatel pozornost. Ve skutečnosti je obsahem parametru následující odkaz: `http://apIikace/akce`

Jelikož směrovací URL jsou typicky takto zakódovány, důvěryhodná aplikace s přesměrováním nemá problém. Použité kódování, tzv. „URL encoding“, ve výchozím stavu kóduje pouze speciální znaky, jako „/“, nicméně zcela bezpečně zpracuje i vstup výše.

V závislosti na písmu daného prohlížeče si uživatel nemusí všimnout, že se nachází na

útočné stránce „apiikace“ (velké „i“ předstírá malé „el“) a může považovat dokonalou kopii uživatelského rozhraní za bezpečnou. Pokud se v danou chvíli např. přihlásí, útok úspěšně končí.

Variací tohoto útoku je pak zneužití tzv. „předání“, v případě ASP.NET známý jako „`server transfer`“. Klíčem k úspěchu je možnost útočníka změnit např. URL parametr, podle kterého se aplikace při přesměrování rozhoduje, kam má uživatel pokračovat a na pozadí provádí automaticky `server transfer`. Pokud se provede takovéto předání, aniž by bylo ověřeno oprávnění uživatele (jako při běžném požadavku), obešel útočník zabezpečení.

Při identifikaci potenciálních míst pro „nevalidovaná přesměrování a předání“ je vhodné hledat: všechna přesměrování v aplikaci (včetně 30x HTTP odpovědí), která mohou být ovlivněna zadaným parametrem (typicky část nebo celá URL), v kódu pak místa přesměrování závislá na parametru a přesuny na serveru na pozadí (například pro MVC volání akce controlleru z jiné akce) – pokud zadaný parametr pro přesměrování není striktně validován, může být zranitelný, dále typicky přihlašovací formuláře s přesměrováním na původní požadovaný zabezpečený zdroj.

### 5.2.4 Způsob obrany

Následující seznam popisuje způsoby obrany od nejrestriktivnějších:

- Úplný zákaz přesměrování a předání v rámci aplikace (běžně není možný),
- odstranění jakékoliv vazby mezi cílem všech přesměrování a vstupními parametry požadavků,
- vytvoření dočasných zástupných hodnot pro přesměrování,
  - typicky když aplikace rozhoduje, zda uživatele přesměruje po dané akci na controller a akci „A“ nebo „B“, je vhodnější nevkládat přímo cílové URL do parametru, ale lze umístit například zástupný hash nebo číselnou identifikaci, kterou na straně serveru logika přeloží na konkrétní přesměrování („white-list“),
  - není tedy možné směrovat jinam, než aplikace explicitně očekává,
- validace cílové URL,
  - jestli je lokální, relativní, v rámci domény, resp. aplikace,
  - jestli je v rámci očekávaného rozsahu („white-list“),
- validace explicitním „white-listem“ na absolutní URL mimo doménu apod.,
- validace regulárním výrazem na omezenou sadu možných adres.

V případě serverových „předání“ platí první dva výše uvedené body zamezující ovlivnění cíle předání. Pokud je předání na základě parametru nutné, je bezpodmínečně nutné také ověření oprávnění uživatele stejně, jako kdyby pocházel požadavek „zvenku“.

Pokud tedy například akce controlleru ASP.NET MVC vrací v určitém případě přímo obsah jiné akce a to na základě vstupního parametru, je nutné validovat oprávnění uživatele explicitně v místě volání akce, jelikož tento „`server transfer`“ obchází zabezpečení pomocí atributu. Níže uvedený příklad zobrazuje neošetřené předání, které umožní přejít na akci „B“ bez oprávnění:

```
public ActionResult TransferAction(string targetAction)
{
   if(targetAction == „A“){ return A();}
   if(targetAction == „B“){ return B();}
   ...
}
```

### 5.2.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.AccountController Metoda Logon (post) |
| **Oprava** | T1_SecuBank.Controllers.AccountController Metoda LogonSafe (post)<br>T1_SecuBank.Logic.Secure.HomeLogic Metoda ValidateRedirect |

#### *5.2.5.1 Ukázkové použití: zranitelné*

Pouku uživatel přistoupí nepřihlášen k zabezpečené části aplikace, je přesměrován na přihlášení společně s parametrem určujícím cestu, kam původně přistoupil. Po přihlášení aplikace přesměruje uživatele na původní místo. Následující úryvek obsahuje metodu, která přesměruje uživatele po přihlášení:

```
[HttpPost]
public ActionResult Logon(...string ReturnUrl)
...
 if (logic.LogonUser(login, password))
 {...
  return Redirect(ReturnUrl);
```

Pokud je URL požadavku standardní, je uživatel přesměrován v rámci aplikace, např.:

`localhost/T1_SecuBank/Account/Logon?`**`ReturnUrl=%2fT1_SecuBank%2fAdmin`**

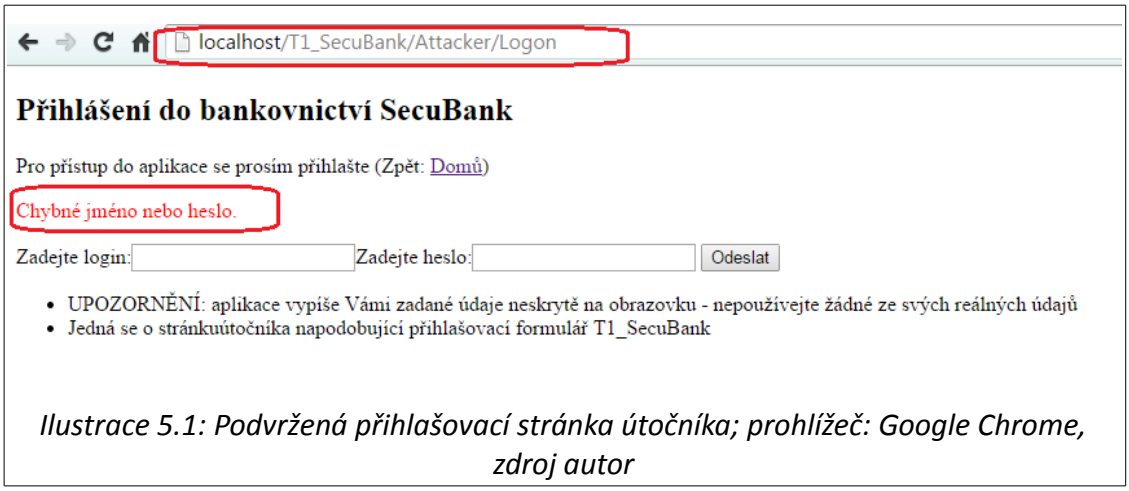
Útočník může této vlastnosti zneužít a odeslat například na sociální síť následující odkaz:

**`.../T1_SecuBank/Account/Logon?ReturnUrl=%68%74%74%70%3A%2F%2F%6C%6F%63%61%6C%68%6F%73%74%2F%54%31%5F%53%65%63%75%42%61%6E%6B%2F%41%74%74%61%63%6B%65%72%2F%4C%6F%67%6F%6E`**

Adresa skrytá za kódováním je:

`http://localhost/T1_SecuBank/Attacker/Logon`

Jelikož aplikace nevaliduje cílovou adresu, je uživatel přesměrován na stránku napodobující přihlašovací formulář. Útočníkova stránka vyzve uživatele k přihlášení s odvoláním na chybu „Chybné jméno nebo heslo“. Uživatel v domnění překlepu údaje vyplní.




*Ilustrace 5.1: Podvržená přihlašovací stránka útočníka; prohlížeč: Google Chrome, zdroj autor*

#### *5.2.5.2 Ukázkové použití: bezpečné*

Jelikož je funkčnost v aplikaci vyžadována a není možné explicitně vyjmenovat všechny cíle přesměrování (jako v případě průvodce), není možné použít nejlepší řešení z kapitoly 5.2.4, Způsob obrany. Z tohoto důvodu aplikace před samotným přesměrováním parametr validuje nejstriktnější možnou validací.

```
[HttpPost]
public ActionResult Logon(...string ReturnUrl)
...
 if (logic.LogonUser(login, password))
 {...
  if (...logic.ValidateRedirect(ReturnUrl, Url))
  return Redirect(ReturnUrl);
```

Implementace metody `ValidateRedirect` pak ověřuje, jestli je daná URL lokální a jestli je relativní (parametr `UrlHelper` je předán z controlleru a přináší především kontext):

```
public bool ValidateRedirect(string ReturnUrl, UrlHelper url)
{
 return Uri.IsWellFormedUriString(ReturnUrl, UriKind.Relative)
                && url.IsLocalUrl(ReturnUrl);
```

## 5.3 Zranitelnost: Přehrávání relace

Přehrávání relace (z angl. „Session replaying“) je dalším ze skupiny útoků zaměřených na autentizační a identifikační mechanizmy aplikace (navazuje na 4.2). Cílem útoku je navázání na existující validní relaci uživatele a získání jeho identity. Tato kapitola se zaměřuje specificky na navázání na tiket „FormsAuthentication“ popsaný v kapitole 4.2. Obdobný mechanizmus může platit i u jiných schémat zabezpečení. Zranitelnost je (v popsané podobě) irelevantní pro aplikace identifikující/autentizující uživatele jiným způsobem (jako certifikát, tokeny kerberos apod.).

### 5.3.1 Princip útoku a obrany

Zranitelnost využívá principu některých autentizačních mechanizmů, kdy serverem vytvořený identifikační „token“ není pouze bez-významový pseudonáhodný řetězec, ale obsahuje

také informace potřebné pro rozhodnutí o jeho platnosti. Pokud server neuchovává na své straně stavovou informaci o tokenech a jejich platnosti, je nutné, aby uchoval v tokenu samotném veškeré informace pro vyhodnocení validity, data platnosti a expirace a také identifikaci uživatele – jako uživatelské jméno. Útočníkovi tedy stačí získat autentizační token uživatele, který i po úspěšném odhlášení bude platný (server nedokáže rozhodnout, jestli došlo k odhlášení nebo ne). Přehrávání relace tedy spočívá ve znovupoužití již odhlášené relace.

Teoretickou možností zůstává stejně jako u únosu relace zamezení používání identifikačního tokenu. Pokud je nutné token použít, nutnou obranou je vytvoření druhotných kanálů pro ověření platnosti přihlášení uživatele – typicky druhý token, označení ne/aktivních uživatelů v úložišti a sledování dalších potencionálních identifikátorů v požadavku uživatele.

### 5.3.2 Kategorizace

Kategorie: [A.3], [D], [B.1], [F]

#### *5.3.2.1 Severita, dopad*

| **Severita** | **A+, Kritická** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | B | Data, Účet |
| Odborný náhodný | B | Data, Účet |
| Odborný cílený | B | Data, Účet, Osoba |
| Profesionální cílený | A+ | Data, Služba, Účet, Osoba |

*Tabulka 5.3: Severita a dopad přehrávání relace, zdroj autor*

#### *5.3.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizená data, uživatelské účty, vyhrožují kompromitací aplikace, dlouhodobě poskytují přístup do aplikace třetím stranám za úplatu. Dále riskují finanční postihy, žaloby a trestní řízení.

Vlastníci: ztrácí postavení na trhu, platí smluvní pokuty a penalizace, ztrácejí důvěru uživatelů, dlouhodobě ztrácí kontrolu nad uživatelskými účty a daty, jsou s problémy spojováni.

Třetí strany: platí za cílené poškození (reputace) konkurence, získání přístupu do aplikace pod vybranými účty, cíleně poškozují konkrétní účty a osoby.

### 5.3.3 Způsob prověření (provedení)

Základním příkladem pro „session replaying“ je získání odhlášené uživatelské relace. Útočník jedním z možných způsobů získá identifikátor sezení uživatele, který se s falešným pocitem bezpečí z aplikace odhlásil. Pokud je například ponecháno nastavení `FormsAuthentication` ve výchozím stavu, může útočník nachystat útok na veřejném počítači (v kavárně) tak, že nastaví hlavičku „UserAgent“ prohlížeči na „Generic Downlevel“. Tím prohlížeč přistoupí k dané aplikaci jako „cookieless“ prohlížeč a aplikace mu automaticky vrátí forms token v URL adrese (viz Zranitelnost: Únos relace). Po nějaké době v prohlížeči zobrazí historii a uvidí autentizační tokeny v URL. Ačkoliv se uživatelé odhlásili z aplikace, token zůstává aktivní, dokud časově nevyprší.

Pokud aplikace nemá dostatečně krátkou dobu expirace přihlášení, útočník obyčejně přidá cookie do prohlížeče s vybraným tokenem a je úspěšně identifikován. Server nemá ve výchozím stavu žádné prostředky, jak rozpoznat, že daný token byl odhlášen, jelikož na své straně neudržuje žádnou takovou informaci. Tuto variaci útoku ukazuje také zmiňovaná kapitola 4.2.

**Útok přímo na mechanizmus tokenu**

Rozšířenou a nebezpečnější variantou útoku je napadení přímo samotného tokenu. Jelikož všechny informace pro vyhodnocení tokenu jsou jeho součástí, musí být v tokenu uchováno minimálně uživatelské jméno uživatele a časové razítko vytvoření nebo jejich obdoba. Cílem tohoto útoku je pak odhalení techniky vytvoření a kryptování tokenu, způsobu jeho validace a následně nahrazení klíčových částí tokenu jinými, případně kompletní sestrojení validního tokenu. Níže uvedený příklad útočí na FormsAuthentication ticket přímo tak, že se snaží změnit uživatelské jméno identifikované serverem a přihlásit se tak za libovolného uživatele, bez nutnosti jakkoliv získávat jeho token.

*Pozn. autora: FormsAuthentication využívá rozdílné implementace ve frameworku verze 2, 3, 4 a 4.5 a každé z těchto novějších schémat zanáší do procesu obskurnější způsob zabezpečení. V principu doplňuje jiné způsoby šifrování, komplikovanější techniky kódování a rozpadu binárních dat v rámci řetězce tiketu apod. Jelikož nebyla nalezena žádná veřejně dostupná technika pro přímé napadení nejnovějších mechanizmů (kryptovaného a validovaného tokenu Forms .NET 4.5), je níže uvedený příklad postaven na zranitelnosti z verze 2.0. Základní princip zranitelnosti však zůstává stejný a je tedy možné, že cílený útok zkušených skupin obdobně dokáže v současnosti napadnout i nejnovější zabezpečení. Další překážkou pro hlubší studium kromě časového prostoru je také nedostupnost zdrojových kódů některých částí frameworku.*

Následující úryvek kódu obsahuje konfiguraci souboru „Web.config“, díky které je možné relativně snadno tento útok provést:

```
<authentication mode="Forms">
 <forms loginUrl="/T1_SecuBank" protection="None"
ticketCompatibilityMode ="Framework20" />
</authentication>
<machineKey applicationName="A" compatibilityMode="Framework20SP1" />
```

„Zvyšováním“ bezpečnosti uvedené konfigurace (jako atribut „protection“) dochází k doplnění kryptování a validace tokenu na straně serveru. Úspěšně vytvořený a validní tiket je ale stále možné vytvořit, pouze za daleko vyšších nákladů.

Po přihlášení útočník získá přihlašovací token s následující hodnotou:

```
0102C7E3AB05D832D208FEC7178E36DC32D208000574006F006D006100730000012F00FF
```

Zdánlivě náhodný a obtížně napadnutelný řetězec je možné lépe analyzovat s větším vzorkem. Níže je uveden seznam tokenů několika uživatelů:

```
0102 E00C8B01 CA32D208FE E0406D32 CE32D2080005 610064006D0069006E00
00012F00FF <-- token uzivatele admin
0102 BF973A0B CA32D208FE BFCB1C3C CE32D2080005 610064006D0069006E00
00012F00FF <-- token uzivatele admin
0102 9034DCE0 CE32D208FE 9068BE11 D332D2080005 74006F006D0061007300
00012F00FF <-- token uzivatele tomas
0102 EA9688E1 D032D208FE EACA6A12 D532D2080009
7300690067006F00750072006E0065007900 00012F00FF <-- token uzivatele
sigourney
```

Po důkladnějším prozkoumání tokenu je možné jej rozdělit na několik logických bloků. Relativně rychle je možné se dobrat faktu, že se jedná o hexadecimální zápis binárního obsahu (každá číslice tedy představuje 4 bity):

- Úvodní sekvence (pravděpodobně verzovací, první 4 číslice = 16 bitů = 2 bajty),
- bloky 8, 10, 8 a 12 proměnlivých číslic (neznámý obsah),
- blok s proměnlivou délkou (uživatelské jméno),
- závěrečný statický blok délky 10.

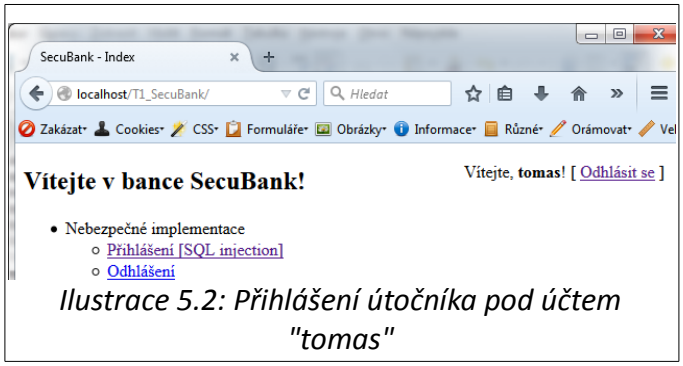
V pořadí pátý blok na konci obsahuje číslici (vyznačeno podtržením), která určuje délku proměnlivého uživatelského jména, kterou ASP.NET využije při jeho hledání v řetězci. **Tučně** je pak zvýrazněno uživatelské jméno v hexadecimálním tvaru, zakódované kódováním Unicode. Jelikož unicode je 16bitové kódování (2bajtové), každé 4 číslice označují jeden znak jména. Proto je uživatel „admin“ zapsán 20 znaky.

S využitím této znalosti může útočník vytvořit kompletní token (bude nutné dohledat obsah zbylých bloků), nebo se může přihlásit obyčejným účtem uživatele, například „sygourney“, a následně změnit své uživatelské jméno na „admin“.

Přihlašovací token uživatele „tomas“:

```
01029531D7E3DA32D208FE9565B914DF32D2080005 74006F006D0061007300 00012F00FF
```

Útočník je přihlášen následovně:

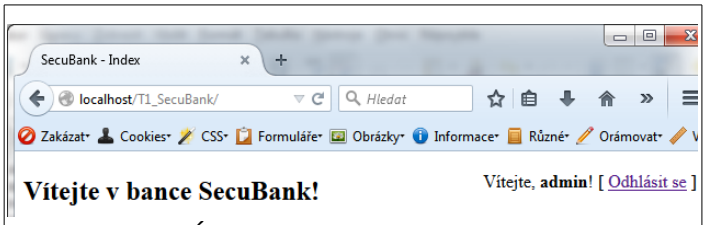



*Ilustrace 5.2: Přihlášení útočníka pod účtem "tomas"*

Útočník následně změní svůj token na:

```
01029531D7E3DA32D208FE9565B914DF32D2080005610064006D0069006E0000012F00FF
```

Jelikož délka jména je shodná, není potřeba měnit délku, postačí změna jména a útok je úspěšný. Aplikace identifikovala uživatele jako „admin“.




*Ilustrace 5.3: Útočník přihlášený tokenem se jménem "admin" po přihlášení pod uživatelem "tomas"*

Předcházející příklad je pouze ilustrační, jelikož útočníkovi aplikace „velmi napomáhá“ svou nebezpečnou konfigurací. Principiálně je však možné útočit takovým způsobem na libovolně zabezpečený token, o kterém si server neponechává na své straně perzistentní informaci.

Při identifikaci potenciálních míst pro „přehrávání relace“ je vhodné hledat: způsoby expirace tokenu, expirace při odhlášení (ověření, zda odhlášený token není stále funkční), způsob vytváření identifikačních tokenů a způsob prověření na straně serveru, jejich „významovost“ a možnost s nimi manipulovat tak, aby je server interpretoval, ověření, jestli aplikace kombinuje token s další obranou (druhý token, validace ip adresy a další).

### 5.3.4 Způsob obrany

Následující seznam popisuje obranné mechanizmy od základních:

- Konfigurace Forms autentizace podle posledních doporučení,
  - `cookieless = "UseCookies", protection = "All",`
  - `requireSSL = "true"` (pokud možno),
  - `slidingExpiration = "false"`, a vhodné časy expirace,
- vytvoření druhé úrovně validace přihlášení založené na relaci („session“),
  - jelikož jsou data „session“ uchována na serveru a na klientský stroj je odesílán jen pseudonáhodný řetězec, je možné využít jej jako druhou úroveň validace,
  - při přihlášení je do „session“ uchována také IP adresa, hlavička „UserAgent“, hodnota tokenu forms a uživatelské jméno,
  - tyto hodnoty jsou pak také validovány při každém požadavku na serveru,
- perzistentní uchování stavu přihlášení uživatele v databázi,
  - u každého uživatele je v databázi uložen stav jeho přihlášení,
  - pokud se explicitně odhlásí, je tento stav nastaven na „odhlášen“ a pak není možné „přehrávání relace“ využít.

### 5.3.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Logic.Insecure.HomeLogic Metody LogonUser, LogoffUser |
| **Oprava** | T1_SecuBank.Logic.Secure.HomeLogic Metody LogonUser, LogoffUser<br>T1_SecuBank.Logic.Secure.SecurityLogic<br>T1_SecuBank.Global.asax.cs |

#### *5.3.5.1 Ukázkové použití: zranitelné*

Aplikace využívá standardní autentizaci Forms ve výchozím nastavení (nehledě na cookieless mode). Po přihlášení je uživateli zadán běžný neperzistentní autentizační tiket:

```
FormsAuthentication.SetAuthCookie(user.Login, false);
```

Token vystavený přihlášenému uživateli je ve výchozím nastavení (`protection="all"`) zakryptován a validován proti tajným klíčům na serveru (machine key apod.). Jeho podoba je následující:

```
9A7F60D892E342643D06563EDEE2C95C46B1209DD7A459E230A24C9CAA6824DADC4664D5
44F25FEA748BCDBBF7B32D444A0A892321F37DBB2AF5AA80DCA18F8D45CD80FE4D35FD07
5E6FAE9095BE5D6984F9E494C33B45CF1834664634F55EE7
```

Uživatel se odhlásí a token je odstraněn z jeho cookies kolekce. Níže uvedený kód aplikace využívá k odhlášení uživatele.

```
FormsAuthentication.SignOut();
```

Pokud však útočník získá tento zdánlivě odhlášený token uživatele, může jej bez problémů použít k přihlášení. Jelikož metoda `SignOut` vyvolá pouze odstranění cookie z prohlížeče a na serveru není uchován žádný záznam o relaci, ani odhlášení uživatele, server nedokáže token za odhlášený prohlásit.

Zdrojem problému může být krom uvedených také konfigurace aplikace, která sníží nativní zabezpečení autentizace Forms tokenu, viz 5.3.3. V souboru `T1_SecuBank/Web.config` jsou obsaženy ukázkové zranitelné a bezpečné konfigurace.

Dále ve zdroji `T1_SecuBank.Infrastructure.Util.FormsAuthenticationTicketDecryptor` jsou uvedeny úryvky zdrojového kódu, který se pokouší napadnout samotný token a úspěšně napadá implementaci tokenu pro Framework2.0.

#### *5.3.5.2 Ukázkové použití: bezpečné*

Výchozím předpokladem obrany je korektní konfigurace tokenu forms. Výchozí konfigurace je podle zdroje [22], až na mód cookieless (viz 4.2), bezpečná. Tato konfigurace je následující:

```
<forms loginUrl=... cookieless="UseCookies"
enableCrossAppRedirects="false" name=... protection="All"
slidingExpiration="false" ticketCompatibilityMode="Framework40" />
```

Ani takto striktní konfigurace a nastavení nízké doby expirace však nezajišťují zabezpečení proti přehrání relace. Z tohoto důvodu je nutné doplnit jej o další token nebo doplňující informace.

Aplikace doplňuje výchozí nastavení autentizace o dodatečnou validaci. Po přihlášení (`T1_SecuBank.Logic.Secure.HomeLogic` metoda `LogonUser`) dochází k uložení dodatečných identifikátorů relace na serveru do objektu „Session“ (serverová část tokenu „session id“) společně s vytvořením autentizačního tokenu Forms. Následující úryvek obsahuje odpovídající kód (`T1_SecuBank.Logic.Secure.SecurityLogic` metoda `SetAuthCookies`):

```
public void SetAuthCookies(string login)
{...
 string protector = Guid.NewGuid().ToString();
 ... model = new UserAuthenticationModel
 {
  UserHostAddress = HttpContext.Current.Request.UserHostAddress,//[1]
  UserAgent = HttpContext.Current.Request.UserAgent,//[2]
  Login = login,//[3]
  Protection = protector//[4]
 };
 HttpContext.Current.Session[...] = model;
 ... ticket = new FormsAuthenticationTicket(2, login,DateTime.Now, ...
                    protector,//user data...);
 string strTicket = FormsAuthentication.Encrypt(ticket);
 HttpCookie cookie = new HttpCookie(..., strTicket);
 ...
 HttpContext.Current.Response.Cookies.Add(cookie);
```

Pro každou novou relaci je vytvořen unikátní identifikátor (`"protector"`, podtrženo), který je uchován na serveru a zároveň vložen do Forms tokenu ve formě tzv. „`UserData`“. Tato textová položka je určena k uchování volitelných dat v tokenu, přičemž je odesílána v zakódované podobě na klienta. Zdánlivě tedy umožňuje podvržení.

Společně s ochranným prvkem je na serveru uchována IP adresa (označeno `[1]`), hlavička prohlížeče (ozn. `[2]`) a uživatelské jméno (`[3]`). Při každém požadavku na server je pak provedena validace, zda ochranné prvky nebyly narušeny:

```
public bool ValidateAuthentication()
{...
 FormsAuthenticationTicket ticket = GetFormsAuthTicket();
 ...model = HttpContext.Current.Session[...] as UserAuthenticationModel;
 if (ticket == null || model == null) return false;

 bool valid = ticket.UserData == model.Protection//[4]
  && ticket.Name == model.Login//[3]
  && Request.UserHostAddress == model.UserHostAddress//[1]
  && Request.UserAgent == model.UserAgent;//[2]
```

Toto ověření zajišťuje ochranu proti následujícím scénářům:

- Útočník úspěšně podvrhne (nebo získá) autentizační token Forms,
  - jelikož nemá nastaven ověřovací objekt v Session nebo není korektní, nebude útok úspěšný,

- útočník úspěšně změní svůj autentizační token Forms (např. změna jména),
  - ověřovací objekt v Session neodpovídá, útok není úspěšný,
- útočník úspěšně podvrhne token a získá identifikátor „session id“,
  - je-li podvržen token Forms a je odcizen identifikátor session, útočník úspěšně prochází všemi kontrolami až na ověření ochranného prvku ( [4] )

Pro úspěšný útok na tento obranný mechanizmus musí útočník získat obě hodnoty cookies Forms tokenu i session identifikátoru, zároveň podvrhnout IP adresu a hlavičku User Agent. Alternativou je útok hrubou silou na ochranný prvek.

### 5.3.6 Dodatečné informace

Server IIS od společnosti Microsoft využívá při forms autentizaci právě cookie, která nemá na serveru uchované žádné perzistentní údaje. Podle dostupných zdrojů, jako [22] je tato zranitelnost součástí designu (z angl. „by design“) autentizačního modulu a je možné ji předcházet korektním nastavením absolutních expirací aj. Z tohoto lze usuzovat, že se autentizační schéma nebude měnit a je nutné tuto zranitelnost ošetřit.

Doplňujícím zdrojem pro tuto zranitelnost byly: [36]

## 5.4 Zranitelnost: Únos kliknutí

Únos kliknutí (z angl. „Clickjacking“) je zranitelnost, jejíž cílem je zmanipulování uživatele k nevědomému provádění akcí v aplikaci. Zranitelnost je relevantní pro všechny aplikace.

### 5.4.1 Princip útoku a obrany

Hlavním opěrným bodem zranitelnosti je základní vlastnost HTML dokumentů (resp. XML obecně) a to možnost dokumenty kombinovat. V těchto dokumentech je možné pro to určenými prvky (tag IFRAME apod.) vložit jiný dokument na libovolné místo v obsahu. Druhou zásadní vlastností je, že útočník má možnost zcela přesně umisťovat obsah dokumentu tak, aby překrýval vybrané části vložených dokumentů. Jelikož překrytá aplikace může působit zcela neškodně, uživatel může ve falešném pocitu bezpečí potvrzovat dotazy aplikace, provádět akce, aniž by věděl, se co na pozadí děje.

Princip obrany vychází z výše uvedených nutných podmínek. U většiny aplikací je možné zamezit vložení jejího obsahu do jiného dokumentu bez omezení funkčnosti. U aplikací, u kterých to možné není, vychází obrana primárně ze snahy omezit vkládání obsahu do jiného dokumentu, než specificky určeného. Zamezení druhému opěrnému bodu, tedy přesné grafické úpravě obsahu není možné zabránit. Závěrečným principem obrany je pak poučení uživatelů o bezpečném chování a specifikace politik (hesel, odhlašování).

### 5.4.2 Kategorizace

Kategorie: [A.3], [E], [F]

#### 5.4.2.1 Severita, dopad

| **Severita** | **B, Střední** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Osoba |
| Odborný náhodný | C | Osoba |
| Odborný cílený | C | Data, Účet, Osoba |
| Profesionální cílený | B | Data, Účet, Osoba |

*Tabulka 5.4: Severita a dopad únosu kliknutí, zdroj autor*

#### 5.4.2.2 Ekonomické aspekty úspěšného útoku

Útočníci: za úplatu překrývají obsah aplikace za reklamní nebo útočný, získávají důvěru uživatelů pro další útoky, obchodují připravené „clickjacking pasti“ jiným subjektům. Dále riskují žaloby.

Vlastníci: ztrácí důvěru uživatelů, jsou spojováni s útokem a službami, které jsou zobrazeny v překryvech aplikace, jsou cílem dalších útoků.

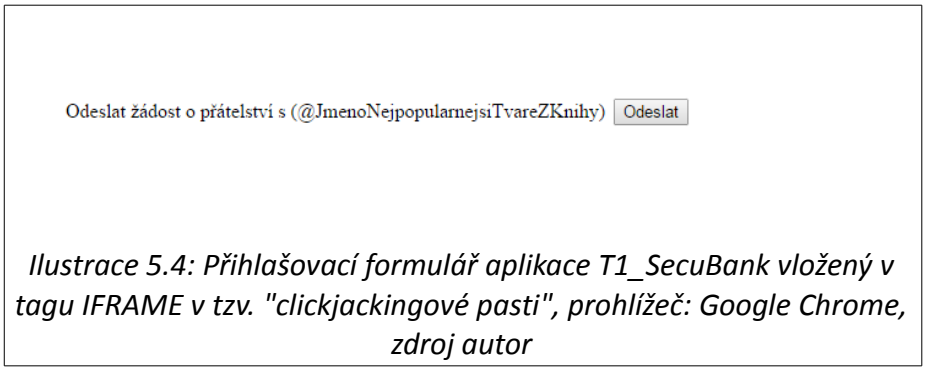
Třetí strany: platí za cílené poškození (reputace) konkurence, za zobrazení reklam uživatelů, získávají klientelu na základě zobrazení jejich služby důvěryhodnou aplikací.

### 5.4.3 Způsob prověření (provedení)

Nejběžnějším způsobem provedení tohoto útoku je „obalení“ cílené aplikace vlastním HTML obsahem a její vložení pomocí tagu „IFRAME“. Typicky je pak obalovací HTML obsah kaskádovými styly připraven tak, aby přesně překryl potřebné části aplikace. Následující výpis obsahuje kód umístění ukázkové aplikace T1_SecuBank do pasti (zkráceno pro stručnost):

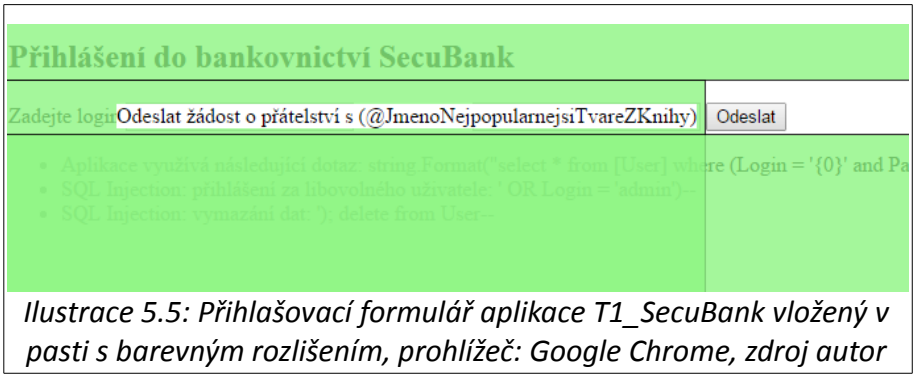
```
<div style="width:100%; height:100%; position: absolute; top: 0; left: 0;">
 <div style="position:absolute;background:white;left:0;right:0;height: 50px;"></div>
 <div style="position:absolute;background:white;top:50px;left:0;bottom:0;width:535px;...
 <div style="position:absolute;background:white;top:90px;left:0;right:0;bottom:0;...
 <div style="position:absolute;top:68px;left:90px;">
  Odeslat žádost o přátelství s (@JmenoNejpopularnejsiTvareZKnihyTvari)
 </div>
 <iframe src="http://localhost/T1_SecuBank/Account/Logon" width="100%" height="100%"
style="margin: 0;padding: 0;border: none;"></iframe>
</div>
```

Překrytá stránka obsahuje přihlašovací formulář aplikace. Útočník spoléhá na uživatelem uložené přihlašovací údaje, které jsou prohlížečem automaticky předvyplněny. Uživatel tedy prohlíží útočníkovu aplikaci s obsahem, který jej motivuje ke stisknutí tlačítka „Odeslat“ (věta „Odeslat žádost o přátelství...“ v obrázku níže). Tlačítko „odeslat“ je ovšem součástí formuláře aplikace T1_SecuBank. Obrázek níže obsahuje náhled výsledku uvedeného kódu:




*Ilustrace 5.4: Přihlašovací formulář aplikace T1_SecuBank vložený v tagu IFRAME v tzv. "clickjackingové pasti", prohlížeč: Google Chrome, zdroj autor*

Následující obrázek vyznačuje překryvové vrstvy útočné aplikace barevně s průhledností tak, aby byl zřejmý záměr útoku. Jak je z obrázku patrné, cílem je přihlášení uživatele do T1_SecuBank pomocí uložených údajů. Následně se může útočník pokoušet o další útoky, například 6.1.




*Ilustrace 5.5: Přihlašovací formulář aplikace T1_SecuBank vložený v pasti s barevným rozlišením, prohlížeč: Google Chrome, zdroj autor*

Obsah překrývající aplikaci může dosahovat prakticky libovolné složitosti, může se jednat o jednoduchý formulář, video přehrávač technologie flash, plátno HTML 5, počítačovou hru apod. Typickým příkladem jsou online hry zaměřené na postřeh hráče (rychlé kliknutí na cíl). Zrychlené vnímání uživatele snižuje pozornost k detailům a je zneužito k náhodnému kliknutí na vybraný prvek. Takový útok umožní obejít standardní obranné mechanizmy, jako je zabezpečení relace, tokeny pro validaci formulářů (viz 6.1) a další, jelikož jde o regulérní akci uživatele.

Jak doplňuje zdroj [2], je možné využít únos kliknutí také sofistikovanějším způsobem. Například na aplikaci zabezpečenou proti padělání požadavků (6.1) je možné využít direktivu pro zobrazení zdrojového kódu stránky.

**Rozšířená varianta se zobrazením zdrojového kódu**

Pro vybrané prohlížeče (Google Chrome, Mozilla Firefox) je možné použít URL direktivu `"view-source:"`. Ta způsobí, že je zdroj prohlížečem načten a zobrazen jako zdrojový kód. Tímto způsobem lze uživateli zobrazit část zdrojového kódu se zabezpečovacím prvkem tak, aby jej v domnění zabezpečení nakopíroval nebo opsal útočníkovi přímo do aplikace. Uváděnou záminkou je přesvědčení uživatele, že opisuje bezpečnostní kód pro ochranu před strojovými útoky.

Následující obrázek ukazuje základní velice jednoduchou kostru takového útoku:


*Ilustrace 5.6: Přihlašovací formulář aplikace T1_SecuBank vložený v pasti využívající direktivu „view-source:“ s barevným rozlišením, prohlížeč: Google Chrome, zdroj autor*

V příkladu je viditelný token pro ochranu formuláře (viz 6.1), díky kterému nedokáže útočník padělat požadavky za uživatele. Příklad je jen ilustrační, standardní token je dostatečně náhodný a kryptograficky bezpečný, nikoliv „T-O-K-E-N-123456“. Pokud však tento token získá, tj. uživatel jej opíše do graficky vhodně připraveného okna překrývající aplikace, obranu obešel. Zelené průhledné obdélníky zastupují vizuálně a obsahově motivující obsah útočné aplikace.

Při identifikaci potenciálních míst pro „únos kliknutí“ je vhodné hledat: formuláře a akce, které je možné uživateli vnutit tak, aby provedl požadovanou operaci, aniž by o tom věděl, konfiguraci aplikace a specifikaci hlaviček HTTP, které mohou bránit vkládání obsahu do rámců, javascriptový kód bránící umístění do rámců (viz 5.4.4).

### 5.4.4 Způsob obrany

Následující seznam popisuje způsoby obrany od základních a nejlépe podporovaných:

- Vložení javascriptového kódu, který kontroluje umístění dokumentu v hierarchii DOM,
- vložení HTTP hlavičky X-Frame-Options,
- vložení HTTP hlavičky CSP, *[Oprava pro Úroveň 3 – Standardní].*

Javascriptový kód pro kontrolu vložení dokumentu v rámu je jednoduché a široce podporované řešení. Jeho funkčnost je pochopitelně závislá na spuštění kódu javascript. Hlavička `X-Frame-Options` je novým standardem CSP označena za zastaralou (zdroj [19]), nicméně je podporována množinou prohlížečů, které nový standard nepodporují.

Hlavička CSP umožňuje přesnější řízení aplikacím, které řeší integraci s dalšími produkty právě pomocí vkládání dokumentů. Například umožní povolit „obalení“ konkrétním zdrojem.

### 5.4.5 Ukázka

| **Projekt** | T1_SecuBank |
|---|---|

| | |
|---|---|
| **Hlavní zdroj problému** | T1_SecuBank.Views.Account.Logon (resp. controller)<br>Konfigurace filtrů/hlaviček aplikace<br>T1_SecuBank.Views.Attacker.ClickJack |
| **Oprava** | T1_SecuBank.Controllers.AccountController metody LogonSafe<br>Konfigurace filtrů/hlaviček aplikace (T1_SecuBank.FilterConfig) |

#### *5.4.5.1 Ukázkové použití: zranitelné*

Útočník připravil aplikaci, která má za cíl uživatele na pozadí přihlásit do aplikace T1_SecuBank a následně hodlá využít další typy útoků. Útočníkova stránka vypadá obdobně jako na obrázku 5.4. Stránka (`T1_SecuBank.Views.Attacker.ClickJack`) pro lepší názornost umožňuje odkrytí útoku kliknutím na odkaz v levém horním rohu:


*Ilustrace 5.7: "ClickJackingová past" připravená na přihlašovací formulář aplikace T1_SecuBank; prohlížeč: Google Chrome, zdroj autor*

Aplikace ve výchozím nastavení nebrání vložení do tagu `Iframe`. Pokud je útok prováděn z jiné domény, jak je pro tento typ útoku běžné, většina prohlížečů brání alespoň v přístupu k obsahu „obaleného“ dokumentu (platí tzv. „Same-Origin policy“, „politika shodného původu“, autor). Pokud by útok byl proveden například podsunutím skriptu na stejné doméně, může útočník přistupovat k objektům `iframe.contentWindow` a `iframe.contentDocument` a ovládat tak veškerý obsah aplikace.

Po najetí myší na červeně označený obdélník je spuštěn časovaný útok, který předpokládá, že uživatel během následujících tří sekund klikne na tlačítko „Odeslat“. Po tomto intervalu se pokusí spustit útok a zároveň uživateli oznámí, že jeho akce byla „úspěšná“. Útok je připraven tak, aby působil obdobně ve všech běžných prohlížečích.

#### *5.4.5.2 Ukázkové použití: bezpečné*

Aplikace doplňuje přihlašovací formulář o všechny bezpečnostní prvky popsané v kapitole 5.4.4, Způsob obrany. Tyto prvky jsou pro názornost uvedeny pouze na přihlašovací stránce `LogonSafe`. V praxi by však tyto prvky měly být přidány do aplikace globálně, jak naznačuje `T1_SecuBank.FilterConfig`.

Prvním obranným mechanizmem, který je podporován prakticky všemi prohlížeči je umístění kontrolního skriptu. Variací této obrany je mnoho – aplikace využívá kód ze zdroje [29]. Následuje úryvek tohoto skriptu:

```
<style id="antiClickjack">body{display:none !important;}</style>
<script type="text/javascript">
   if (self === top) {
       var antiClickjack = document.getElementById("antiClickjack");
       antiClickjack.parentNode.removeChild(antiClickjack);
   } else {
       top.location = self.location;
   }
</script>
```

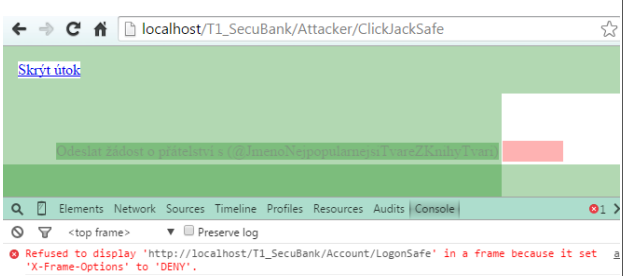
Uvedený skript skryje veškerý obsah stránky (podtrženo) a kontroluje, jestli je umístěn v rámu. Pokud ano, vynutí přesměrování nejvyššího rámu na svou URL (tučně). Klíčovou částí skriptu oproti jiným je právě úvodní skrytí obsahu. Pokud útočník dokáže obejít přesměrování nejvyššího rámu (což je podle [29] možné), nedojde k přesměrování, ale obsah zůstává skryt. Jelikož útočník nemá běžně možnost zasahovat do obsahu načteného dokumentu, uživateli se rám jeví jako prázdný.

Druhým a třetím obranným mechanizmem je přidání HTTP hlaviček do odpovědi serveru. Hlavička `X-Frame-Options` je implementována pouze některými prohlížeči a není součástí žádného standardu (viz 2.3). Standardy hlavičku pouze jmenují. Naopak hlavička `Content-Security-Policy` a její vlastnost `frame-ancestors` (v předchozí verzi také `frame-options`) je součástí standardů (standard „CSP level 2“), nicméně je relativně nová (aktualizace 2/2015) a je podporována primárně moderními prohlížeči.

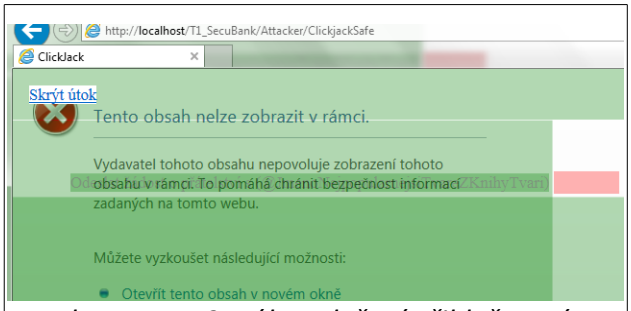
Aplikace doplňuje metody pro přihlášení `LogonSafe` o atribut `FrameHeadersFilter`. Následující úryvek obsahuje klíčové části kódu tohoto filtru:

```
public class FrameHeadersFilterAttribute : ActionFilterAttribute
{
 public override void OnActionExecuting(... filterContext)...
 ...AppendHeader("X-Frame-Options", "DENY");
 ...AppendHeader("Content-Security-Policy", "frame-ancestors 'none'");
```

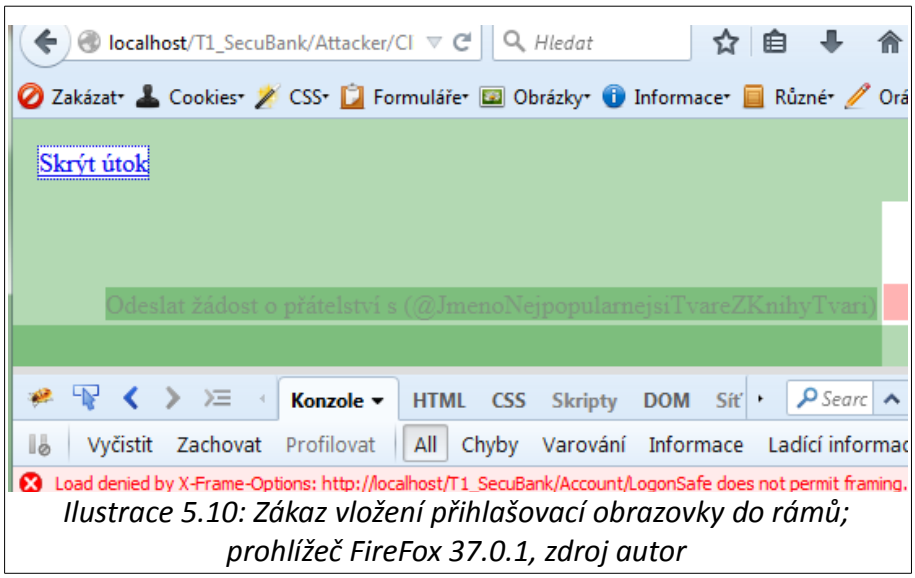
Tyto hlavičky lze podle specifikace kombinovat a prohlížeč by měl upřednostnit deklaraci CSP. Podle testů provedených autorem práce je však prohlížeči upřednostňována hlavička `X-Frame-Options`, pravděpodobně z důvodů větší rozšířenosti a povědomí. Následující obrázky zachycují reakci prohlížečů na zaslané hlavičky:




*Ilustrace 5.8: Zákaz vložení přihlašovací obrazovky do rámů; prohlížeč Google Chrome 41.0, zdroj autor*




*Ilustrace 5.9: Zákaz vložení přihlašovací obrazovky do rámů; prohlížeč Internet Explorer 10, zdroj autor*




*Ilustrace 5.10: Zákaz vložení přihlašovací obrazovky do rámů; prohlížeč FireFox 37.0.1, zdroj autor*

### 5.4.6 Dodatečné informace

Doplňujícím zdrojem pro tuto zranitelnost byly: [19], [32]

## 5.5 Zranitelnost: Hromadné přiřazení

Hromadné přiřazení (z angl. „Mass assignment“) je zranitelnost zaměřená na změnu polí aplikace zneužitím automatického přiřazení frameworkem. Zranitelnost je irelevantní pro aplikace, které neobsahují pole, která bezpečně uživatel nesmí v některých kontextech měnit, nebo využívající framework, který neprovádí automatické přiřazování.

### 5.5.1 Princip útoku a obrany

Podmínkou nutnou, nikoliv postačující pro úspěšný útok je automatické přiřazování hodnot do proměnných na serveru, bez nutnosti explicitního vyjmenování tvůrcem aplikace. Druhou základní, na webové platformě splněnou (viz 2.2.2), podmínkou je možnost podvrhnout vstup tak, aby byla naplněna neočekávaná proměnná, která změní logiku provádění operace podle potřeb útočníka. Není-li automatické přiřazování upozorněno na zvlášť citlivou proměnnou, nebo není uveden seznam proměnných k naplnění, může být naplněna jakákoliv proměnná v kontextu.

Možnost vypnout automatické přiřazování frameworkem není většinou ani výhodná, ani efektivní. Vývojář standardně tuto vlastnost využívající vyhledá obdobné automatizované nástroje a může tím zavést více bezpečnostních rizik , viz 6.3. Optimální obranou tak zůstává explicitní vyjmenování položek k mapování (white-list), případně jmenování citlivých položek (black-list).

### 5.5.2 Kategorizace

Kategorie: [A.2], [A.1], [B.1], [C.2]

#### 5.5.2.1 Severita, dopad

| **Severita** | **B, Střední** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Data |
| Odborný náhodný | C | Data |
| Odborný cílený | B | Data, Účet |
| Profesionální cílený | B | Data, Účet |

*Tabulka 5.5: Severita a dopad hromadného přiřazení, zdroj autor*

#### 5.5.2.2 Ekonomické aspekty úspěšného útoku

Útočníci: obchodují odcizené informace, napadené účty, zvyšují svá aplikační privilegia, získávají finanční prostředky, virtuální měny a jiné přímým odebráním.

Vlastníci: přicházejí o finanční a jiné prostředky, ztrácí kontrolu nad aplikací, ztrácí důvěru uživatelů.

Třetí strany: platí za cílené poškození (reputace) konkurence, za převod a získání finančních prostředků, změny kritických dat (finančních ukazatelů, údajů o konkurenci).

### 5.5.3 Způsob prověření (provedení)

Typickým příkladem zranitelnosti je tzv. automatické „bindování“ („vázání“, autor) přijatých hodnot z požadavku na modely v aplikaci. Toto vázání je prakticky základním stavebním kamenem moderních MVC frameworků, jako Ruby On Rails, PHP Nette nebo ASP.NET MVC. Při odeslání HTTP požadavku klientem na server (metodou GET, POST nebo jinou) dojde typicky k vyjmenování polí umístěných ve formuláři (tag `form`) a jejich spojení do požadavku, například následovně:

```
POST /Application/BankAccount/ChangeNote HTTP/1.1
Host: mysafebank.com ...
Content-Type: application/x-www-form-urlencoded ...
Cookie: ...
accountId=439874&accountNote=Family+account
```

Pokud útočník přidá do odeslaného požadavku (resp. formuláře) nové pojmenované pole, je zahrnuto v požadavku, nezávisle na tom, zda jej tvůrce aplikace ve formuláři očekává nebo požaduje. Záleží pouze na implementaci na straně serveru, jak se zachová k přijatým polím.

Využívá-li aplikace byznys modely, nebo ještě hůře databázové modely přímo v prezentační vrstvě a jsou-li tyto modely automaticky vázány na požadavky klienta, může útočník podvrhnout libovolnou vlastnost daného modelu v krajním případě až do databáze. V příkladu výše si klient u svého bankovního účtu mění poznámku na „Family account“. Útočník může modifikovat požadavek na následující:

```
POST /Application/BankAccount/ChangeNote HTTP/1.1
...
accountId=439874&accountNote=Family+account&balance=666666
```

Nově přidaný parametr (podtrženo) je automaticky namapován na serveru na databázový model bankovního účtu, konkrétně na položku „balance“ („zůstatek“, autor). Pokud aplikace neobsahuje dostatečně robustní validace (viz 6.3 a 7.2), transakční záznamy (viz 7.3) a napoví útočníkovi, jaká pole má k dispozici (viz 6.4), může úspěšně dojít ke zvýšení zůstatku na cíleném účtu bez možnosti prokázat neplatnost dané transakce. To vše v rámci přidání poznámky k účtu.

Při identifikaci potenciálních míst pro „hromadné přiřazení“ je vhodné hledat: formuláře pracující s kritickými entitami, u který neuvádí kompletní výčet polí modelu, nebo se skrytými poli, asynchronní požadavky javascriptu (ajax) a v nich uvedené modely a pole, v kódu pak způsob vytvoření a zpracování prezentačního modelu, vazbu mezi datovým, databázovým, logickým, byznys a prezentačním modelem, případně místa přímého zpracování formulářových dat (`FormCollection`), požadavku (`Request`) apod.

### 5.5.4 Způsob obrany

Následující seznam popisuje obranné mechanizmy od základních:

- Explicitní odstranění (black-list) nebezpečných parametrů požadavku,
- ruční zpracování každé akce/požadavku a odeslání dalším vrstvám pomocí parametrů metod (bez využití modelů),
- ruční zpracování každé akce/požadavku a namapování požadovaných položek na model (využití databázových nebo byznys modelů),
- explicitní vyjmenování parametrů akce a jejich namapování na model (využití databázových nebo byznys modelů),
- vytvoření vlastních mapovacích pravidel (tzv. „binderů“) pro jednotlivé modely a akce (využití databázových, byznys nebo prezentačních modelů),
- vytvoření specifických modelů s explicitním výčtem položek zastupujících vybrané databázové a byznys modely (využití prezentačních modelů),
- vytvoření specifických modelů s explicitním výčtem položek pro každou akci (využití prezentačních modelů),
- vytvoření specifických modelů (předchozí bod) a zaznamenání všech pokusů o zápis neočekávaných parametrů *[Oprava pro Úroveň 4 – Elitní]*.

Využitím kteréhokoliv z uvedených bodů, krom prvního, je útočníkův požadavek omezen pouze na vyjmenované (white-list) položky daného modelu. Podvržený vstup je tak zahozen.

### 5.5.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.BankAccountController metody EditNote |
| **Oprava** | T1_SecuBank.Controllers.BankAccountController metody EditNoteSafe |

#### *5.5.5.1 Ukázkové použití: zranitelné*

Aplikace umožňuje změnit textovou poznámku u bankovního účtu. Tvůrce aplikace použil

jako model třídu z databázové vrstvy aplikace (*pozn. autora: tento způsob je v praxi velmi běžný*). Akce `EditNote` vypadá následovně (instance účtu je získána z databáze):

```
public ActionResult EditNote(long id)
{...
 Account account = logic.GetAccountForEdit(id);
 return View(account);
```

Třída označená podtržením je databázová entita. Je vrácena přímo do zobrazovací vrstvy, a namapována („bind“) na formulářová pole. Standardně má formulář pro tuto editaci následující kód:

```
<form action="/T1_SecuBank/BankAccount/EditNote/2" method="post">
...
 <input id="Id" name="Id" type="hidden" value="2">
...
 <input id="Note" name="Note" type="text" value="Stará poznámka">
...
 <input type="submit" value="Uložit">
</form>
```

Útočník se může pokusit vložit formulářové pole tak, aby změnil výši svého účtu. Název pole se může pokusit uhodnout hrubou silou nebo může vyhledávat indicie v aplikaci (například pojmenování polí, atributů a položek v rámci detailu). Útočník vloží do formuláře následující prvek:

```
<input id="Balance" name="Balance" type="text" value="999999">
```

Vykreslený formulář obsahuje nové pole s požadovanou hodnotou. Po stisknutí tlačítka „uložit“ je v požadavku odeslán také parametr `Balance`, jelikož prohlížeč nedokáže rozpoznat, zda jde o citlivou položku, či nikoliv. *Útočník navíc nemusí používat prohlížeč*. Následující úryvek obsahuje kód zpracování na serveru:

```
public ActionResult EditNote(long id, FormCollection form)
{...
 Account account = logic.GetAccountForEdit(id);
 if (TryUpdateModel(account))
 {
  logic.SaveNote(account);//uložení změn do databáze
...
```

Zranitelnost Hromadné přiřazení nastává při volání nativní metody `TryUpdateModel`, která je spouštěna nad vybraným objektem a pokouší se vyhledat v daném požadavku parametry odpovídající položkám tohoto objektu. Položka „`Balance`“ je součástí databázové tabulky, tedy i odpovídajícího databázového modelu a je tedy namapována. Následuje řádek volající uložení změn v rámci ORM způsobí zápis položky do databáze.

#### *5.5.5.2 Ukázkové použití: bezpečné*

Tvůrce aplikace namísto využití databázového modelu vytvořil specifický model pro danou akci. Tento model obsahuje pouze identifikátor měněného účtu a položku `Note`. Tento tzv. „ViewModel“ je předán do zobrazovací vrstvy. Je vykreslen prakticky stejný formulář jako v případě zranitelné implementace výše.

Útočník se opět pokusí vložit pole „Balance“ a odeslat formulář. Následující kód zpracovává tento požadavek:

```
public ActionResult EditNoteSafe(BankAccountNoteModel account)
{
 logic.SaveNoteModel(account);
...
```

Jako parametr akce je uveden zmiňovaný specifický model dané akce. Alternativou je také využití výše popsané metody `TryUpdateModel`. Výsledek obou variant je totožný. Ručně přidané pole nemá v modelu odpovídající položku proto není namapováno. Tento ViewModel je pak předán byznys vrstvě, která aktualizuje v databázi pouze poznámku zadaného účtu. Změna zůstatku tedy není možná.

# 6 . Úroveň 3 – Standardní

Tato kapitola se věnuje popisu Standardní úrovně zabezpečení webových aplikací a zranitelností do ní spadajících.

Standardní úroveň předpokládá splnění Základní úrovně zabezpečení. Jedná se o profesionální úroveň zabezpečení, kterou by měly splňovat všechny aplikace státní správy včetně triviálních, aplikace středního a většího rozsahu a veškeré aplikace související s obchodní, finanční, investiční nebo obdobnou činností. Tvůrci takto zabezpečených aplikací běžně spoléhají na hotová bezpečnostní řešení a knihovny nebo auditované frameworky a komponenty. Zahrnuje zabezpečení proti sofistikovaným útokům, typicky prováděným cílenými odbornými útočníky a proti zranitelnostem platformy a systémového rozsahu. Aplikace této úrovně zabezpečení předpokládají cílené útoky zainteresovaných skupin, předcházejí nejen zranitelnostem, ale také jejich případným dopadům v případě úspěchu útočníků. Útoky standardní úrovně běžně nejsou prováděny náhodnými amatéry nebo během hromadných plošných útoků v Internetu.

Tuto úroveň by měla splňovat každá aplikace obsahující uživatelská nebo firemní data s vysokou hodnotou, nebo zasahující široké publikum uživatelů (v řádech od tisíců) – typicky aplikace úřadů státní správy, nekritické bankovní aplikace, firemní účetní aplikace, integrační aplikace a software určený pro plánování zdrojů firmy.

Aplikace nesplňující plnohodnotně tuto úroveň (a všechny předcházející), s výjimkou zranitelností nerelevantních pro typ aplikace nebo platformu, je nutné považovat za nebezpečné pro použití ve: vnitrofiremním a mezi-firemním prostředí (typu ERP, CRM), aplikace státní správy (všech typů).

Mezi typické zranitelnosti na této úrovni patří podvržení vstupů na několika úrovních, příprava vstupů aplikace tak, aby působily škodu až v připravený okamžik (časované útoky), nebo zneužití nejnovějších zranitelností platformy, standardních výchozích konfigurací aj.

## 6.1 Zranitelnost: Padělání požadavku napříč stránkami

Padělání požadavku napříč stránkami (z angl. „Cross Site Request Forgery“, CSRF) je zranitelnost zneužívající možnosti odeslat požadavek mezi webovými stránkami tak, že prohlížeč přihlášeného uživatele samovolně doplní autentizační zabezpečení. Zranitelnost je irelevantní pro aplikace, které neprovádí žádné neveřejné editační akce.

### 6.1.1 Princip útoku a obrany

Pro úspěšný útok musí mít útočník především možnost uživateli zobrazit stránku s vlastním obsahem – obvykle skriptem. Druhým klíčovým bodem je přihlášená relace uživatele – ať už nově přihlášená, prohlížečem automaticky spuštěná (uložené jméno, heslo), nebo trvalá. Poslední, co útočník k úspěchu potřebuje, je dostatečná znalost cílené aplikace, aby mohl vytvořit padělaný požadavek za uživatele. Tuto znalost typicky útočník získává vlastním použitím aplikace pod svým nebo odcizeným účtem. Pokud jsou všechny podmínky splněny, útočník na připravené stránce vytvoří požadavek na aplikaci tak, že je odeslán z prohlížeče uživatele ve chvíli, kdy je přihlášen. Uživatel neví, že může vedlejší aplikace tuto akci provést. Navíc, jelikož je přihlášen, je automaticky odeslána jeho identifikace a další ochranné prvky aplikace na server, jako by akci provedl sám uživatel.

Optimální obranu, tedy zamezení jakémukoliv zobrazení útočného obsahu uživatelem není možné dlouhodobě důvěřovat – obzvláště pak u aplikací této úrovně. Zabránit přihlášení uživatele v době útoku také není prakticky možné, nicméně je možné šanci snížit zakázáním perzistentních relací a ukládání jmen a hesel. Standardním obranným mechanizmem je vložení pseudonáhodných bezpečnostních tokenů do formulářů aplikace a jejich kontrola při přijetí požadavku. Tento způsob obvykle také vyžaduje omezení „editačních“ akcí pouze na takto zabezpečené **formuláře**.

### 6.1.2 Kategorizace

Kategorie: [A.3], [D]

#### *6.1.2.1 Severita, dopad*

| **Severita** | **A, Vysoká** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Data |
| Odborný náhodný | B | Data |
| Odborný cílený | B | Data, Služba, Účet, Osoba |
| Profesionální cílený | A | Data, Služba, Účet, Osoba |

*Tabulka 6.1: Severita a dopad padělání požadavků napříč stránkami, zdroj autor*

#### *6.1.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizené informace, napadené účty, zvyšují svá aplikační privilegia, získávají finanční prostředky, virtuální měny a jiné jejich přímým odebráním, za úplatu útočí na uživatelské skupiny a účty, mění cíleně data v aplikaci, znehodnocují celý ekonomicko-společenský účel aplikace za úplatu. Dále riskují žaloby.

Vlastníci: přicházejí o finanční a jiné prostředky, ztrácí kontrolu nad aplikací, ztrácí důvěru uživatelů, chybně podezřívají uživatele z podvodných činností, ukončují činnost.

Třetí strany: platí za cílené poškození (reputace) konkurence, za převod a získání finančních prostředků, změny kritických dat (finančních ukazatelů, údajů o konkurenci), napadení vybraných účtů a skupin, poškození jednotlivců.

### 6.1.3 Způsob prověření (provedení)

Typickým příkladem zranitelnosti je podvržení požadavku na smazání dat. Aplikace umožní uživateli mazat vybraná data například volbou přímo v seznamu. Běžné řešení takové situace je umístění odkazu do seznamu. Aby ale uživatel nemohl „omylem“ odstranit nějaký záznam nebo aby nemohl útočník odkaz aktivovat za uživatele, je typicky „zabezpečen“ javascriptovým dotazem (podtrženo), například:

```
<a href="/MyFriends/Delete/1"
onclick="return confirm('Opravdu chcete odstranit přítele Kamarád?');">
Odstranit</a>
```

Po aktivování takového odkazu je uživatel dotázán, zda akci hodlá provést a pokud potvrdí,

odkaz je aktivován a požadavek typu `GET` je odeslán na server. Útočník v takovém případě odešle uživateli e-mail, do kterého umístí obrázek s následujícím obsahem:

```
<img src="http://aplikace.cz/MyFriends/Delete/1" alt="Šťastné a veselé">
```

Uživatel otevře zprávu v prohlížeči nebo v e-mailovém klientovi, obrázek se pokusí načíst, nicméně v daném zdroji narazí pouze na `HTTP` odpověď `200` vracející HTML obsah. V takovém případě se obrázek nenačte, nicméně požadavek na server o smazání záznamu byl úspěšně doručen. Pokud byl uživatel přihlášen v prohlížeči, společně s požadavkem byly standardně odeslány i jeho cookies (například s Forms autentizačním tokenem), případně další identifikátory. Z pohledu aplikace se tak jednalo o regulérní požadavek uživatele.

Obdobným způsobem může útočník přichystat útok na požadavek typu `POST`. Mnohdy doporučovaná technika obrany, použití `POST` akcí pro všechny editační a jinak akční formuláře, je tedy neúplná.

Při identifikaci potenciálních míst pro „padělání požadavků“ je vhodné hledat: veškeré formulářové i ne-formulářové akce, které způsobují změnu dat v aplikaci, jejich skrytá pole a způsob ověření jejich pravosti, v kódu pak využití identifikátorů formulářů, skrytých polí, kontrolních součtů odeslaných a přijatých formulářů a další.

### 6.1.4 Způsob obrany

Následující seznam popisuje možné způsoby obrany od nejméně efektivních:

- Zpracování všech „akčních“ požadavků ve formě „`POST`“ formulářů (např. samostatné obrazovky pro mazání záznamu),
- umístění skrytých polí s identifikačním tokenem do všech formulářů,
- bod výše a vložení tokenů také do všech odkazů v aplikaci (dotazy typu „`GET`“).

Na serveru je ve chvíli vykreslování formuláře uložen pseudonáhodný řetězec, který je umístěn do skrytého pole ve formuláři. Po odeslání formuláře je vyhledána hodnota tohoto prvku v požadavku na serveru a pokud není uvedena nebo neodpovídá očekávané uchované hodnotě, je požadavek prohlášen za nevalidní.

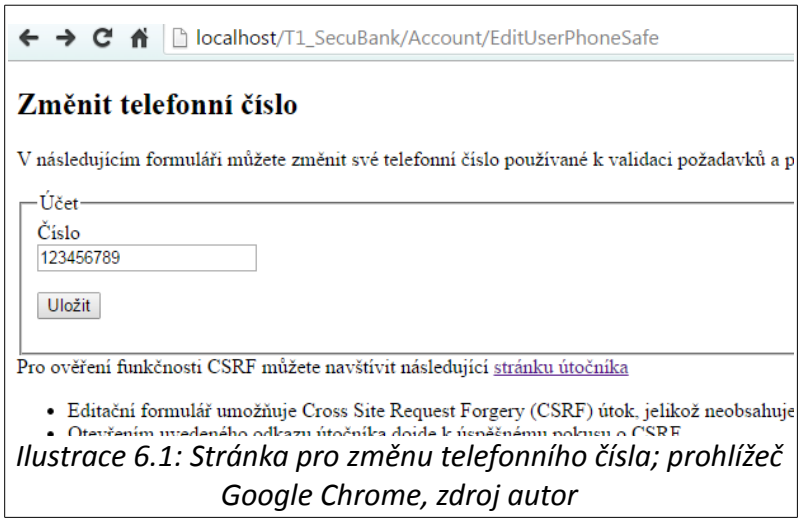
Obdobně v případě vykreslení odkazu je uchován tento řetězec a při obdržení požadavku na stránku je validován jeho příjem a platnost. Tokeny v obou případech mohou mít i omezenou časovou platnost.

### 6.1.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.AccountController Metody EditUserPhone<br>T1_SecuBank.Controllers.AttackerController Metoda CrossSiteRequest |
| **Oprava** | T1_SecuBank.Controllers.AccountController Metody EditUserPhoneSafe<br>T1_ SecuBank.Views.Account.EditUserPhoneSafe |

#### 6.1.5.1 Ukázkové použití: zranitelné

Aplikace umožňuje uživatelům změnit své telefonní číslo používané k ověření plateb a dalších citlivých operací. Pro zjednodušení příklad neobsahuje další bezpečnostní opatření, jako je ViewModel změny (viz 5.5), ověření přihlášení, kontroly účtu a další. Uživatel zobrazí formulář pro editaci svého čísla a číslo může změnit. Tato operace vyžaduje, aby byl uživatel přihlášen, může být kontrolováno oprávnění a změna na pozadí probíhá pro přihlášené uživatelské jméno. Aby útočník mohl jinou cestou změnit toto číslo, byl by nucen obejít množství zabezpečovacích prvků.




*Ilustrace 6.1: Stránka pro změnu telefonního čísla; prohlížeč Google Chrome, zdroj autor*

Útočník uživatele „pozve“ na svou útočnou stránku, která obsahuje javascriptový kód. Ten vytvoří na pozadí formulář (pro lepší skrytí by jej mohl vytvořit v Iframe) a odešle jej. Tento formulář vypadá následovně:

```
<form action=".../T1_SecuBank/Account/EditUserPhone" method="post">
 <input... name="phoneNumber" type="text" value="666999666">
</form>
```

Útočníkova stránka je pro zjednodušení implementace umístěna ve stejné aplikaci a na stejné doméně. Tento fakt však na útoku nic nemění, jeho funkčnost je nezávislá na doméně nebo aplikaci. Zcela externí aplikace mimo doménu T1_SecuBank bude stejně úspěšná.

Odeslaný požadavek vytvořený javascriptem je prohlížečem uživatele automaticky doplněn o autentizační cookies (Forms token i Session Id) a je tedy považován za regulérní odeslání změny telefonu uživatelem.

#### 6.1.5.2 Ukázkové použití: bezpečné

Aplikace doplňuje změnu mobilního telefonu o validační token, tzv. „AntiForgeryToken“. Tento pseudonáhodný obranný prvek je nativní součástí frameworku ASP.NET MVC.

Při vykreslování formuláře je doplněno skryté pole obsahující token následujícím kódem:

```
@using (Html.BeginForm())
{
 @Html.AntiForgeryToken()
```

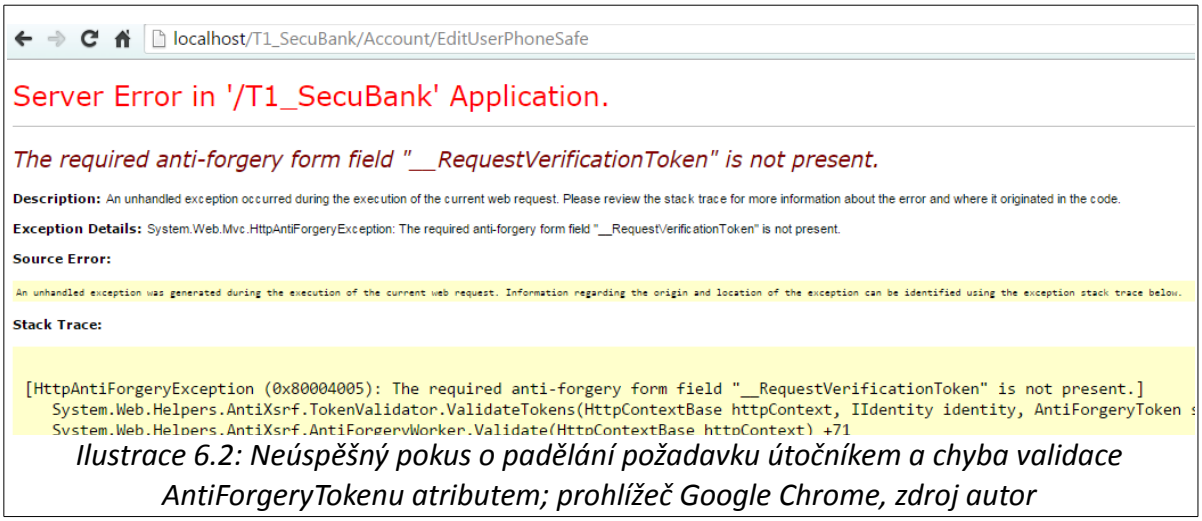
Framework se postará o vygenerování, pojmenování i uchování tokenu. Ve stránce je pak vykreslen níže uvedené skryté pole:

```
<form action="/T1_SecuBank/Account/EditUserPhoneSafe" method="post">
 <input name="__RequestVerificationToken" type="hidden" value="WioDIk...
```

Podtržením vyznačený řetězec má délku desítek znaků a je odeslán s požadavkem na server k ověření. Prohlížeč automaticky přidá také všechny cookies viz 6.1.6, Dodatečné informace. Vybraná akce je pak doplněna o atribut zajišťující kontrolu:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult EditUserPhoneSafe(string phoneNumber)
```

Při navštívení připravené stránky útočníka (se změněným cílovým formulářem) je pokus o padělání požadavku neúspěšný. Aplikace vrací následující chybu:




*Ilustrace 6.2: Neúspěšný pokus o padělání požadavku útočníkem a chyba validace AntiForgeryTokenu atributem; prohlížeč Google Chrome, zdroj autor*

### 6.1.6 Dodatečné informace

AntiForgeryToken spoléhá při své činnosti na několik opěrných bodů – umisťuje v prohlížeči Cookie obsahující mimo jiné uživatelské jméno (identity) nebo tzv „claims token“ (v případě jiných typů přihlášení než forms), druhý validační řetězec (protistranu tokenu ve formuláři) a další informace. Tento obsah je šifrován (pomocí tzv. „machine key“) a binárně serializován.

Podle zdroje [21] (řádek 106) AntiForgeryToken přidává do odpovědi serveru hlavičku „X-Frame-Options“ popsanou v kapitole 5.4. V případě použití tokenu je tedy vhodné prověřit zachování funkčnosti obrany proti únosu kliknutí.

## 6.2 Zranitelnost: Fixace relace

Fixace relace (z angl. „Session fixation“) je další ze skupiny technik útoku zaměřených na autentizační a identifikační mechanizmy aplikace. Cílem útoku je získání identity jiného uživatele na připravené relaci. Útok je (v popsané podobě) irelevantní pro aplikace identifikující/autentizující uživatele jiným způsobem (například certifikátem).

### 6.2.1 Princip útoku a obrany

Na rozdíl od únosu relace (viz 4.2) a přehrávání relace (viz 5.3), primárním cílem tohoto útoku není získání identifikátoru uživatele („tokenu“). Namísto toho je útočníkovým cílem podvrhnout vlastní připravený token tak, aby uživatel navštívil cílenou aplikaci právě s jeho odesláním a aby provedl autentizaci vůči serveru právě proti tomuto tokenu. Jelikož útočník sám uživateli token vložil jeho hodnotu předem zná.

Optimální obranou je implicitní nedůvěřování jakýmkoliv zaslaným tokenům uživatelem a jejich obnovování a změny. Druhá úroveň obrany pak vychází z principů popsaných v kapitole 5.3.

### 6.2.2 Kategorizace

Kategorie: [A.3], [D], [B.1]

#### *6.2.2.1 Severita, dopad*

| **Severita** | **A, Vysoká** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | B | Data, Účet |
| Odborný náhodný | B | Data, Účet |
| Odborný cílený | B | Data, Účet, Osoba |
| Profesionální cílený | A | Data, Služba, Účet, Osoba |

*Tabulka 6.2: Severita a fixace relace, zdroj autor*

#### *6.2.2.2 Ekonomické aspekty úspěšného útoku*

Shoduje se s aspekty pro Zranitelnost: Přehrávání relace, kapitola 5.3.

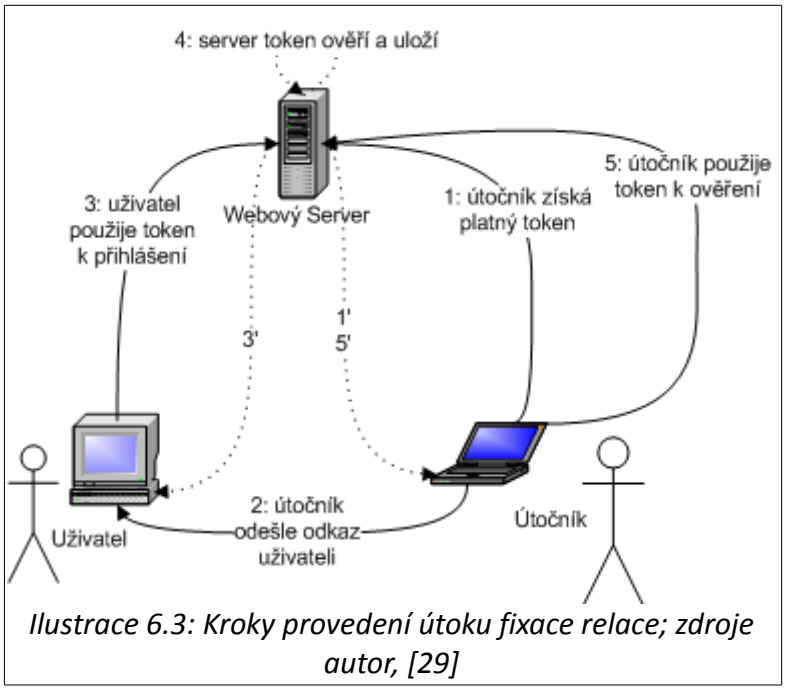
### 6.2.3 Způsob prověření (provedení)

Typickým příkladem pro tuto zranitelnost je umístění připraveného útočného obsahu na stejné doméně, jako je cílená aplikace, podvržení informací v rámci otevřené veřejné sítě – například bezdrátové sítě v kavárně, nebo podvržení cookie na klientském stroji pomocí javacriptu. Další možností je využití identifikátoru v URL, kdy útočník podsune uživatel odkaz na cílenou aplikaci obsahující identifikátor, viz 4.2. Neposlední možností je odeslání e-mailové zprávy obsahující předpřipravený formulář obsahující skryté pole s identifikátorem.

Výše popsané způsoby podstrčení identifikátoru prohlížeči uživatele slouží pouze pro představu a není úplný. Pokud útočník úspěšně dokáže připravený identifikátor uživateli vnutit, stačí pouze vyčkat, až se uživatel přihlásí.

Po úspěšném přihlášení uživatele je token označen na serveru jeho identitou a ve chvíli, kdy odešle požadavek útočník se stejným identifikátorem, je považován za přihlášeného uživatele.

Následující obrázek popisuje jednotlivé kroky útoku:




*Ilustrace 6.3: Kroky provedení útoku fixace relace; zdroje autor, [29]*

### 6.2.4 Způsob obrany

Základním obranným mechanizmem je nedůvěřování přijatým identifikátorům od uživatele, které server pravděpodobně nevystavil a jejich odstranění a nahrazení novými. Jedná se především o následující kritická místa (řazena od důležitějších):

- Příchozí nepřihlášený uživatel, který odeslal požadavek na přihlášení (jménem a heslem, nebo jiným způsobem) s existujícím identifikátorem,
- přihlášený uživatel, u kterého existuje podezření na jiný útok na identifikátor, například neprochází bezpečně ochranou proti přehrávání relace (viz 5.3),
- nepřihlášený uživatel, který přichází k aplikaci s neplatným identifikátorem (expirace),
- přihlášený uživatel, který odeslal požadavek na odhlášení z aplikace.

Při výše uvedených situacích aplikace odstraní token na obou stranách komunikace (server i klient) a vytvoří zcela nové. V případě druhého bodu, například při podezření na podvržení tokenů, aplikace zcela zruší přihlášení a vynutí nové (s novými identifikátory), nebo například při malé změně (hlavičky prohlížeče) vynutí změnu identifikátorů následovanou validací, zda uživatel podezřelé chování již nevykazuje.

### 6.2.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.LotteryController metoda Logon |
| **Oprava** | T1_SecuBank.Controllers.LotteryController metoda LogonSafe |

#### 6.2.5.1 Ukázkové použití: zranitelné

Vedlejší aplikace na doméně T1_SecuBank umožňuje uživatelům přihlášení a účast v soutěži. Uživatel zadá jméno a heslo, následně svůj e-mail a odpověď na soutěžní otázku. Aplikace využívá zjednodušené a nedostatečné přihlašování pomocí session:

```
private void LogonUser(string login, string password)
{...
 Session[SESSION_LOGIN_KEY] = login;
```

Úspěšně přihlášený login je uchován v session a kontrolou, zda je přítomen je určeno přihlášení uživatele. Obdobný způsob autentizace využívají běžně jiné platformy webových aplikací (viz 2.1.1). Cílem útočníka je získání uživatelského e-mailu zadaného v rámci soutěže, případně přihlášení za uživatele, pokud aplikace bude poskytovat další funkcionalitu.

Nejprve útočník získá platné existující id session, například přihlášením za vlastní účet nebo jiný napadený. Tento identifikátor následně podstrčí uživateli – ať už odkazem spoléhající na tzv. „cookieless“ režim (níže) nebo jiným způsobem (lze simulovat vložením cookie manuálně):

`http://localhost/T1_SecuBank/(S(hdx2pnstguud4xgwgluqkmhr))/Lottery`

Uživatel se pod touto session přihlásí a vyplní svůj e-mail. Útočník pouze vyčká na přihlášení uživatele a pouhým obnovením stránky získává uživatelův e-mail a v tomto případě ve skutečnosti kompletní přihlášení k aplikaci.

#### 6.2.5.2 Ukázkové použití: bezpečné

Primárním cílem ochrany není odstranit nedůvěryhodný identifikátor v GET požadavku na přihlašovací formulář. Identifikátor může být útočníkem zafixován až při samotném odeslání formuláře, tedy POST požadavku. Je tedy nutné zajistit, aby byla při obdržení dat formuláře vytvořena zcela nová relace (během POST akce). **ASP.NET neobsahuje** nativní podporu pro změnu identifikátoru sezení („ASP.NET_SessionId“) v rámci jednoho požadavku, jako je například v PHP funkce „`session_regenerate_id`“ (odůvodnění viz 6.2.6, Dodatečné informace). Následující zdroj obsahuje experimentální funkci využívající možností `System.Reflection` k vynucené změně identifikátoru:

`T1_SecuBank.Infrastructure.Util.SessionManagementHelper`, metoda `RegenerateId`

Využití tzv. „reflexe“ a nedokumentovaných úprav v rozhraních toto řešení řadí mezi experimentální. **Nativním způsobem** lze ovšem vynutit, aby do akce odeslání formuláře přicházel uživatel vždy jen s prázdným identifikátorem a byla tak vytvořena nová relace.

```
[HttpPost]...
public ActionResult LogonSafe(string login, string password)
{
 if (!Session.IsNewSession)
 {
  RegenerateSession();...
  return RedirectToAction("IndexSafe");
```

Úryvek ověřuje, zda-li je relace aktuálního dotazu nově vytvořena, tedy jestli identifikátor, který aplikace odešle na klienta je zcela nový. Pokud relace není nová, požadavek tedy přichází

s identifikátorem, je volána metoda pro odstranění relace (níže) a klient je přesměrován zpět na přihlášení. Aby příchozí uživatelé s existující relací nebyli nuceni ke dvojímu přihlašování, do akce vracející formulář přihlášení (GET) je umístěno odstranění relace. Pokud má tedy uživatel existující relaci v době odeslání formuláře, není to relace pocházející z aplikace a uživatel bude přesměrován.

Následující úryvek obsahuje metodu pro odstranění identifikátoru relace:

```
private void RegenerateSession()
{
 SessionIDManager mgr = new SessionIDManager();
 mgr.RemoveSessionID(System.Web.HttpContext.Current);
 Response.Cookies.Add(new HttpCookie("ASP.NET_SessionId", ""));
```

Pokud tedy útočník úspěšně zafixuje token relace před získáním formuláře, je okamžitě odstraněn a přihlášení probíhá bezpečně a pro uživatele bez rozdílu. Pokud jej umístí až po získání formuláře, před odesláním, a uživatel se přihlásí, formulářová akce rozhodne, že se nejedná o důvěryhodný token, který je smazán a uživatel je přesměrován zpět na přihlášení. Uživatel je tedy zasažen nutností dalšího pokusu o přihlášení a je vhodné jej informovat o nevalidním požadavku s odkazem například na znalostní databázi aplikace.

### 6.2.6 Dodatečné informace

Aplikace využívající autentizaci Forms primárně není náchylná k napadení fixací relace. To platí primárně z toho důvodu, že Forms autentizační token obsahuje uživatelské jméno – pokud tedy útočník připraví existující token, jako past na jiného uživatele, nebude úspěšný, jelikož uživatel se přihlásí pod svým jménem a obdrží zcela odlišný token.

Navíc, je-li aplikace zabezpečena **korektně** proti přehrávání relace (5.3), není možné ani zafixovat uživateli kontrolní session token. Korektním zabezpečením je myšleno primárně vložení náhodného identifikátoru („protektoru“ popsaného v 5.3.5.2) do session i do tiketu Forms. Tímto způsobem jsou „pevně“ svázány a není možné využít fixaci náhodné session s libovolným tokenem Forms. Pokud by tento token uveden nebyl, je možné zafixovat session uživateli s uživatelským jménem „admin“ (session je tedy určena uživateli „admin“) a použít získaný neexpirovaný token Forms pro stejného uživatele. Uživatel se přihlásí se zafixovanou session a s novým forms tokenem. Útočníkův forms token však stále platí a s nyní validní session může vstoupit do aplikace.

Z těchto důvodů je kriticky důležité využít náhodný identifikátor svazující Forms token a session nebo obdobné obranné úložiště.

Nemožnost změny identifikátoru relace v ASP.NET vychází primárně z variability dostupných úložišť dat tohoto objektu. Jelikož je pouhou změnou konfigurace změnit ukládání dat relace z paměti serveru na databázi, server pro distribuování relací nebo do vlastního úložiště, musí framework provádět řadu operací tak, aby abstrakce vývojáře od úložiště byla úplná. Daní za tuto míru zobecnění je komplikovaný proces údržby, tvorby a správy relací vedoucí mj. k popsanému.

## 6.3 Zranitelnost: Chybná konfigurace zabezpečení

Chybná konfigurace zabezpečení (z angl. „Security missconfiguration“) je skupina chyb a zranitelností, umožňujících napadení jinak bezpečných komponent aplikace. Zranitelnost je relevantní pro všechny aplikace. Tato zranitelnost je primárně orientována na dopady jiných úspěšných útoků a jejich zmírňování.

### 6.3.1 Princip útoku a obrany

Základní vlastností většiny útoků využívajících zranitelnou konfiguraci je snaha o využití známých problémů, výchozích hodnot a nejběžnějších postupů v praxi. Jedná se tedy o jakousi sbírku „best-practice“ v oboru napadání aplikací. Aplikace musí být tedy chybně nastavena dle známých nebo odhadnutelných pravidel. Druhou podmínkou je obvykle nedostatečné ošetření dopadů úspěšného napadení konfigurace. Podaří-li se útočníkovi zneužít chybné konfigurace aplikace, obvykle ještě neprovedl samotný útok – tato zranitelnost je tedy obvykle doprovázena dalšími útoky.

Primární obranou je kompletní a úplná konfigurace prostředí a všech komponent, systematické a opakovatelné postupy přípravy platformy a pravidelné audity aktuálnosti těchto nastavení, postupů i samotného softwaru. Druhým zásadním bodem obrany je pak zmírňování dopadů případné chyby konfigurace. Jedná se o separaci systémových komponent, minimalizace jejich vzájemných vazeb, přesná specifikace jejich hranic a úplná a centrální kontrola oprávnění mezi jednotlivými částmi systému.

### 6.3.2 Kategorizace

Kategorie: [F], [D], [E], [C], [A.3]

#### *6.3.2.1 Severita, dopad*

| **Severita** | **A+, Kritická** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | B | Data, Služba, Účet |
| Odborný náhodný | B | Služba, Účet |
| Odborný cílený | A | Data, Systém, Služba, Účet, Osoba |
| Profesionální cílený | A+ | Data, Systém, Služba, Účet, Osoba |

*Tabulka 6.3: Severita a dopad chybné konfigurace zabezpečení, zdroj autor*

#### *6.3.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizené informace, napadené účty, administrační rozhraní, přebírají kontrolu nad nebo zneužívají celé prostředí k útokům na třetí strany, nelegálním aktivitám, znehodnocují celý ekonomicko-společenský účel aplikace za úplatu. Dále riskují finanční postihy, žaloby a trestní řízení.

Vlastníci: ztrácí postavení na trhu, platí smluvní pokuty a penalizace za ztráty dat, ztráty na prostředí, za útoky vedené ze zneužité platformy, kompromitaci klientů, přicházejí o klientelu v důsledku poškození účelu aplikace, ukončují svou činnost.

Třetí strany: platí za cílené poškození (reputace) konkurence, odcizení dat s cílem průmyslové špionáže, napadení a převzetí systému, kompletní likvidaci platformy, napadení širokého okolí aplikace, navázaných aplikací a integrací, napadení soukromých osob.

### 6.3.3 Způsob prověření (provedení)

Následující seznam popisuje vybrané typické chyby konfigurace. Tento seznam není úplný. Zároveň je nutné jej individuálně přizpůsobit každé aplikaci a prostředí:

- Známé zranitelnosti platformy a aplikací,
  - neaktuální software obsahující známé a veřejně popsané chyby využitelné útočníkem,
  - nevyužívané doplňkové funkce ve výchozím známém zranitelném nastavení (např. FTP),
- výchozí nastavení zabezpečení,
  - uživatelské účty, hesla a další konfigurace přístupu (například výchozí nastavení logovací komponenty ELMAH („Error Logging Modules and Handlers“, zdroj [7]) umožňující nahlížení do logů libovolnému uživateli na lokálním stroji),
  - účty a výchozí zabezpečení databázových a webových serverů, typicky jakýkoliv výskyt účtu a hesla „admin“, „admin“,
  - nedostatečná výchozí konfigurace bezpečnostních prvků, viz 4.2, 5.3, 5.4 nebo 6.2,
- konfigurace připojení zdrojů,
  - typicky připojení k databázi pomocí tzv. „connection string“ („řetězců připojení“, autor) obsahujících citlivé informace, jako jméno a heslo,
  - připojování ke službám (webové služby) a serverům (databáze, adresářové služby) pomocí zakódovaných účtů nebo pověření uloženém v podobě čistého textu (typicky zápis do adresářové služby pomocí identity tvořené jménem a heslem z konfigurace),
- povolení nežádoucích serverových funkcí,
  - typicky FTP přístup s výchozím zabezpečením, výpis složek serveru (z angl. „directory browsing“),
  - umožnění stažení zdrojových souborů aplikací, nedostatečné omezení možných požadavků přijímaných serverem, jako např. příjem „PUT“ požadavku s nahráním souboru se zdrojovým skriptem na server.

Při identifikaci potenciálních míst pro „chyby konfigurace zabezpečení“ je vhodné hledat: veškeré konfigurační soubory (a jiná úložiště) ve výchozím nastavení, konfigurační položky ponechané v nezměněném stavu (doplněné automaticky platformou), nastavení přístupu ke zdrojům, jako databáze, konfiguraci skupin a účtů (typicky pod jakým uživatelem je aplikace na serveru spouštěna). Dále také jakýkoliv způsob uchování a konfigurace privilegovaných účtů, alternativní přístup k aplikaci a přístupy pro správce. V kódu pak přímo zapsané autentizační údaje,

eskalace oprávnění aplikace, přístup k zabezpečeným zdrojům a způsob využití centralizované správy přístupu.

### 6.3.4 Způsob obrany

Následující seznam popisuje základní oblasti, které je nutné vhodně konfigurovat. Seznam není úplný. Veškerá konfigurace musí být systematicky udržována a provádění pravidelných auditů umožní sledovat aktuální známé problémy.

- Aktuálnost platformy a aplikací,
  - poslední aktualizace operačního systému, virtualizační platformy, ovladačů,
  - poslední aktualizace serverů (databáze, webových, adresářových služeb aj.),
  - poslední aktualizace komponent (frameworku, knihoven, integračních nástrojů),
- nevyužívané volitelné funkce,
  - odebrání všech nevyužívaných, nedostatečně nakonfigurovaných a doplňkových funkcí,
  - odebrání všech přímých vazeb aplikace na doplňkové a volitelné funkce platformy,
- výchozí účty, hesla a zabezpečení,
  - kontrola všech výchozích účtů platformy a jejich zákaz nebo úprava,
  - audit všech výchozích způsobů zabezpečení serverů a služeb „stojících na rozhraní“,
- úroveň bezpečnosti komponent,
  - konfigurace všech komponent na maximální možnou úroveň zabezpečení (viz například 5.3.4, konfigurace cookies na „http only“),
  - v případě nutnosti snížení zabezpečení, prověření, zda nelze úpravou komponenty docílit maximálního zabezpečení (typicky snižování zabezpečení webových služeb komunikujících mezi heterogenními platformami),
- využití komponent třetích stran,
  - sledování doporučených nastavení komponenty dodavatelem (jako zdroj [7]),
  - zmírnění dopadů chyby v komponentě třetí strany omezením oprávnění účtu, který komponentu spouští nebo používá,
- konfigurace připojení zdrojů,
  - připojení k databázovým strojům pomocí „zosobněných“ (z angl. „impersonated“) uživatelských účtů nebo účtů přiřazeným aplikaci („application pool user“ v serveru IIS),
  - odstranění jakýchkoliv identifikátorů z připojovacích řetězců, krom názvu serveru,
  - připojování k webovým službám pomocí certifikátů, určených uživatelských účtů, využití delegování identity a dalších obranných mechanizmů,
- omezení oprávnění a specifikace účtů a skupin,

- kompletní řízení přístupů v rámci prostředí pomocí uživatelských skupin adresářové služby,
- typicky nastavení účtů spouštějících každou aplikaci, minimalizace jejich oprávnění (typicky omezení povolených činností v databázi, omezení pouze na čtení vybraných adresářů na serveru, zamezení jakémukoliv zápisu, zamezení přístupu k jakékoliv části infrastruktury),
- explicitní specifikace oprávnění skupin v databázi (například položením otázky „proč by uživatel aplikace A měl mít možnost měnit databázové schéma aplikace B?“),

- konfigurace serverů,
  - konfigurace webového serveru IIS, například odebrání nepotřebných „handlerů“ pro zpracování požadavků na nepoužívané zdrojové soubory, vypnutím nepoužívané autentizace, konfigurací „URL rewrite“ modulu a vypnutí veškerých identifikátorů v Url požadavcích,
  - nastavení databázových serverů, pravidel transakcí, vymezení databázových schémat a vymezení explicitních oprávnění, propojení se skupinami adresářového serveru,
- konfigurace komunikace,
  - šifrování komunikace v rámci vnitřní sítě, užití HTTPS (typicky webové služby, databáze),
  - šifrování vnější komunikace, je-li to možné,
  - minimalizace počtu důvěryhodných komunikujících stran.

### 6.3.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank/Web.config (sekce connectionStrings)<br>T1_SecuBank.Database.Security |
| **Oprava** | T1_SecuBank/Web.config (sekce connectionStrings)<br>T1_SecuBank.Database.Security |

#### *6.3.5.1 Ukázkové použití: zranitelné*

Aplikace se připojuje k databázovému serveru (Microsoft SQL Server). V řetězcích pro připojení k databázi je uvedena následující hodnota:

```
<add name="T1_SecuBank_Insecure" providerName="System.Data.EntityClient"
connectionString="Data Source=.;Initial Catalog=T1_SecuBank;User
ID=T1_SecuBank_admin;Password=password" />
```

Konfigurace odhaluje uživatelské jméno i heslo kritického účtu v podobě čistého textu. Pokud útočník úspěšně napadne server nebo získá přístup ke konfiguračnímu souboru, může se pokusit připojit přímo na samotnou databázi. Vzhledem k faktu, že aplikace vyžaduje při nejmenším oprávnění číst některá data, může útočník tímto účtem minimálně data odcizit. Následující úryvek obsahuje oprávnění uživatele nastavená na serveru:

```
EXECUTE sp_addrolemember @rolename = N'db_securityadmin', @membername =
N'T1_SecuBank_admin';
...'db_owner', @membername = N'T1_SecuBank_admin';
...'db_ddladmin', @membername = N'T1_SecuBank_admin';
...'db_datawriter', @membername = N'T1_SecuBank_admin';
...'db_datareader', @membername = N'T1_SecuBank_admin';
...'db_backupoperator', @membername = N'T1_SecuBank_admin';
...'db_accessadmin', @membername = N'T1_SecuBank_admin';
```

Vzhledem k uvedeným oprávněním může uživatel provádět prakticky libovolné operace na databázi uživatele. Pokud bude nevhodně nakonfigurován tento účet i na úrovni serveru, může pravděpodobně provést libovolnou operaci i v rámci serveru. Při bližším pohledu je pak zřejmé, že účet je manuálně vytvořený účet serveru se jménem a heslem. Útočník může tedy předpokládat, že účty nejsou centrálně řízeny pomocí adresářového serveru a je tedy možné s účty libovolně nakládat – vytvářet nové přístupy, ukrýt se v rámci databáze, vytvořit přímý přístup na server.

**Server administrátor a převzetí celého prostředí**

Běžně se vyskytující chybou je pak využití v databázi přístupného účtu „sa“. Jedná se o speciální správcovský účet „server administrator“ zabezpečený heslem. Pokud je využit tento účet, nebo účet s dostatečným oprávněním, je možné kompletně převzít kontrolu nad serverem a pravděpodobně i celým prostředím. Za předpokladu využití například podsunutí SQL (kapitola 4.1), následující skript povolí na serveru použití kriticky zranitelné funkce „`xp_cmdshell`“, která je ve výchozím nastavení zakázána (zdroj [4]):

```
EXECUTE sp_configure 'show advanced options', 1
RECONFIGURE WITH OVERRIDE
EXECUTE sp_configure 'xp_cmdshell', '1'
RECONFIGURE WITH OVERRIDE
```

Nyní se může útočník pokusit o vložení příkazu „`xp_cmdshell`“. Pomocí něj je možné využívat standardní příkazové řádky operačního systému Mirosoft Windows. Navíc jsou příkazy spouštěny pod účtem služby serveru, což je obvykle vysoce privilegovaný účet. Následující příkaz „pouze“ vypíše obsah složky serveru:

```
use master
exec xp_cmdshell 'dir'
```

Podtržený text je požadovaný příkaz. Tímto způsobem může útočník na serveru například vytvořit soubory, odeslat požadavky a data na svůj server, konfigurovat prostředí nebo například získat kompletní přístup k serveru a jeho administraci. Jedná se o variaci útoku „podsunutí příkazu operačního systému“ (z angl. „OS command injection“, autor).

### *6.3.5.2 Ukázkové použití: bezpečné*

Upřednostňovanou obranou pro libovolný typ připojení aplikace ke zdroji je centralizovaná správa uživatelských účtů a skupin pomocí adresářových služeb (například Microsoft Active Directory a protokolu LDAP). Na zdrojích (jako SQL server) je pak možné těmto skupinám přiřadit vymezená oprávnění. Připojení z aplikace probíhá následovně:

```
<add name="T1_SecuBank" providerName="System.Data.EntityClient"
connectionString="Data Source=.;Initial Catalog=T1_SecuBank;Integrated
Security=True" />
```

Konfigurace neuvádí uživatele, ani heslo. Dle principů separace odpovědností nemá aplikace správně ani rozhodovat o uživatelském kontextu, ve kterém je spouštěna nebo ve kterém volá externí zdroje – o tom rozhoduje konfigurace prostředí. V tomto případě rozhodne konfigurace serveru IIS, který aplikaci spustí pod účtem uživatele pro tzv. „application pool“. V tomto případě je server nastaven tak, aby volal databázi pod účtem „`IIS APPPOOL\T1_SecuBankPool`“. Tento účet je pak ve skupině, která má na SQL serveru nastavena následující oprávnění (příklad uvádí oprávnění přímo pro uživatele):

```
...'db_datawriter', @membername = N'IIS APPPOOL\T1_SecuBankPool';
...'db_datareader', @membername = N'IIS APPPOOL\T1_SecuBankPool';
```

Uživatel má tedy právo číst a zapsat data. Optimálním zabezpečením je pak vytvoření databázových schémat a omezení čtení a psaní pouze vybraných schémat. Spuštění výše popsaného útoku pro tento účet nepřipadá v úvahu.

## 6.4 Zranitelnost: Únik informací a nesprávné zpracování výjimek

Únik informací a nesprávné zpracování výjimek (z angl. „Information leakage, improper error handling“) je skupina chyb a zranitelností umožňující zobrazení chybových hlášení a podobných dat útočníkem. Zranitelnost je irelevantní pro aplikace platformy nevyužívající princip vyvolání výjimek. Tato zranitelnost je primárně orientována na dopady jiných úspěšných útoků a jejich zmírňování a předcházení.

### 6.4.1 Princip útoku a obrany

Hlavním bodem zájmu během útoku je možnost získání informací, které akcelerují možný postup. Útočník připraví vstup na server tak, aby vyvolal neočekávanou výjimku. Tato výjimka je pak zachycena až na nejvyšší úrovni aplikace tak, aby neukončila celý proces serveru. Pokud není server instruován jinak, výjimku vypíše tak, jako by ji vypsal během vývoje tvůrci aplikace. Tato data mohou obsahovat citlivá data uživatelů, části zdrojového kódu a další kritické hodnoty.

Hlavní obranou je zamezení výpisu chyby na výstup v jakémkoliv kontextu, krom specificky určených (například během vývoje). Druhým klíčovým místem je pak zamezení výpisu jakýchkoliv citlivých a zneužitelných dat, jako jsou libovolné identifikátory, zabezpečovací tokeny a klíče nebo uživatelská data.

### 6.4.2 Kategorizace

Kategorie: [A.3], [B.1], [C.2], [F]

#### 6.4.2.1 Severita, dopad

| **Severita** | **A, Vysoká** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Data |
| Odborný náhodný | C | Data, Služba |
| Odborný cílený | B | Data, Služba, Účet |
| Profesionální cílený | A | Data, Systém, Služba, Účet, Osoba |

*Tabulka 6.4: Severita a dopad úniku informací a nesprávného zpracování výjimek, zdroj autor*

#### 6.4.2.2 Ekonomické aspekty úspěšného útoku

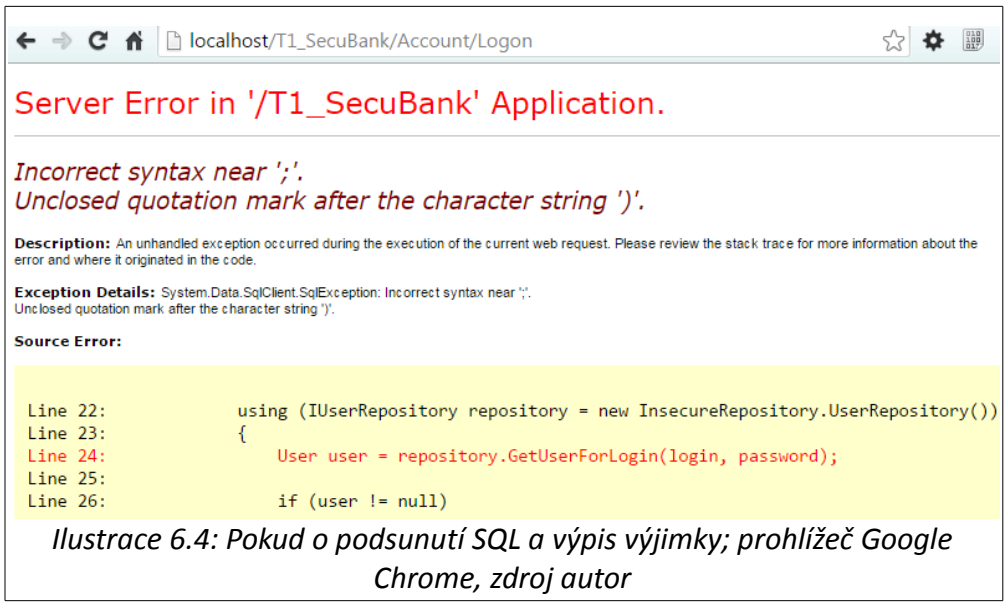
Útočníci: obchodují odcizené informace, za úplatu poškozují aplikaci.

Vlastníci: vykazují ztrátu z důvodu zpomalení nebo poškození aplikace.

Třetí strany: platí za cílené poškození aplikací konkurence, získání utajovaných informací.

### 6.4.3 Způsob prověření (provedení)

Základním příkladem je zneužití informací z výjimek během jiného typu útoku. Pokud se útočník pokouší napadnou aplikaci například pomocí podsunutí SQL (kap. 4.1), je pro něj velmi užitečné, pokud aplikace vypisuje konkrétní a přesné chyby typu „tabulka 'uživatel' nenalezena“. V rámci takový chyby může být vypsán mimo jiné i zdrojový kód s přesným zněním SQL dotazu aplikace. Následující obrázek obsahuje ukázku typické výjimky během pokusu o zmíněný útok:




*Ilustrace 6.4: Pokud o podsunutí SQL a výpis výjimky; prohlížeč Google Chrome, zdroj autor*

Pokročilou variantou je využití výpisu výjimek k získávání informací. Pokud se například útočník snaží zaútočit na identifikátor relace, může se pokusit u uživatele vyvolat výjimku, která bude tento identifikátor obsahovat. Extrémním využitím takové zranitelnosti je získání soukromých klíčů asymetrického šifrování serveru v rámci vrácené chyby.

Při identifikaci potenciálních míst pro „únik informací a nesprávné zpracování výjimek“ je vhodné hledat: jakýkoliv vstup aplikace (formulář, url parametr, asynchronní dotaz javascriptu), který při konkrétní hodnotě dokáže vyvolat výjimku, skryté a pomocné funkce aplikace pro správu relací, které ve výjimce zobrazí hodnotu identifikátoru relace nebo obdobné citlivé informace, v kódu pak místa, která obchází případný centrální mechanizmus zpracování a skrytí výjimek a logovací procedury (typicky výjimky ve zvláštním vlákně vytvořeném v rámci aplikace, které při pádu nezpracuje IIS server).

### 6.4.4 Způsob obrany

Základní obranu je konfigurace aplikace (resp. serveru IIS) tak, aby nezobrazovala výjimky minimálně vzdáleným požadavkům. Dále pak vytvoření centrální implementace pro zpracování výjimek, která je zavedena v aplikaci tak, aby veškeré nezpracované výjimky bylo možné zpracovat a vracet anonymní a bezpečný výstup.

Pokročilým způsobem obrany je vytvoření mechanizmu pro zpracování výjimek mezi jednotlivými vrstvami aplikace. Například výjimka vyvolaná databázovou vrstvou ([C.1]) je odchycena přímo na úrovni zpracování dotazu, kde je „obalena“ novou výjimkou aplikace, případně zcela nahrazena (po uložení do logů) novou bezpečnou výjimkou. Ta je pak zpracována na další vyšší vrstvě, například v prezentačním modelu ([A.2]), kde je přeložena na uživatelskou informační hlášku neobsahující údaje o chybě. Tímto způsobem je minimální množství výjimek zpracováno základním centrálním aparátem a aplikace působí obecně přívětivěji.

### 6.4.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank/Web.config sekce customErrors |
| **Oprava** | T1_SecuBank.Infrastructure.MVC.Filters.LogAndHandleExceptionFilter |

#### *6.4.5.1 Ukázkové použití: zranitelné*

Aplikace ve svém výchozím nastavení při výjimce využije vestavěný mechanizmus pro zpracování výjimek a chybu vypisuje na výstup, jak je uvedeno v obrázku 6.4 výše. Výchozí konfigurace aplikace je následující:

```
<customErrors mode="Off"></customErrors>
```

Výchozí nastavení frameworku používá hodnotu `mode="RemoteOnly"`, což zamezí alespoň zobrazení chybových hlášení v případě vzdálených požadavků.

#### *6.4.5.2 Ukázkové použití: bezpečné*

Je pozměněna konfigurace aplikace a je doplněn centrální modul pro zpracování výjimek:

```
<customErrors mode="On" defaultRedirect="/Home/Error">
```

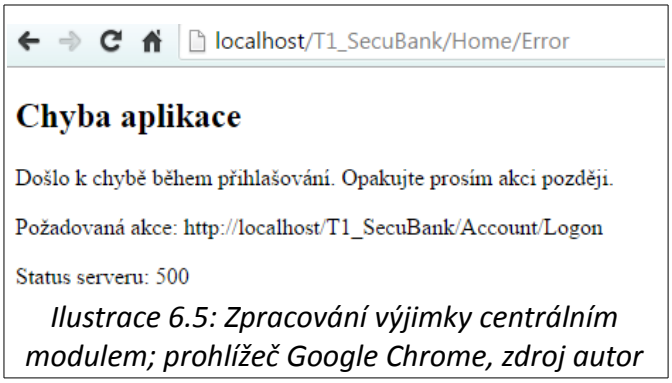
Modul pro zpracování výjimek je zařazen jako filtr globálně na úrovni aplikace ve zdroji `T1_SecuBank[App_Start].FilterConfig`, metoda `RegisterGlobalFilters`. Následující úryvek obsahuje část kódu tohoto filtru:

```
public class LogAndHandleExceptionFilterAttribute : HandleErrorAttribute
{...
 public override void OnException(ExceptionContext filterContext)
 {
  if (filterContext.ExceptionHandled) return;
  if (!filterContext.HttpContext.IsCustomErrorEnabled) return;
  Uri reqUrl = filterContext.RequestContext.HttpContext.Request.Url;
  int statusCode = new HttpException(null, filterContext.Ex...).Get...
  filterContext.Result = new RedirectToRouteResult(...
                    new { action = "Error", controller = "Home"...

  filterContext.Controller.TempData[...] =
      new LogAndHandleExceptionFilterModel
      {
       RequestUrl =..., StatusCode = ..., ExceptionMessage = ...
      };
  filterContext.HttpContext.Response.TrySkipIisCustomErrors = true;
  filterContext.ExceptionHandled = true;
  filterContext.HttpContext.Response.Clear();
```

Filtr zpracuje chybu a přesměruje požadavek na připravenou akci. Navíc uchová v kolekci `TempData` model, který obsahuje dodatečné informace o chybě, které je možné využít při omluvě uživateli. Ukázka také upozorňuje na vhodná opatření, jako například nastavení hodnoty atributu `TrySkipIisCustomErrors`, který umožňuje v produkčním prostředí lépe modulu pracovat.

Výsledný výstup při zadání ukázkového vstupu podsunutí SQL je zobrazen níže:




*Ilustrace 6.5: Zpracování výjimky centrálním modulem; prohlížeč Google Chrome, zdroj autor*

# 7 . Úroveň 4 – Elitní

Tato kapitola se věnuje popisu Elitní úrovně zabezpečení webových aplikací a zranitelností do ní spadajících.

Elitní úroveň předpokládá splnění Standardní úrovně zabezpečení. Jedná se o profesionální úroveň zabezpečení splňovanou aplikacemi, při jejichž tvorbě byla bezpečnost a její zařazena do standardního vývojového procesu. Elitní úroveň by měly splňovat všechny netriviální aplikace státní správy, aplikace většího rozsahu a veškeré aplikace mající přímý vliv na finance a hmotné statky společnosti nebo jednotlivců. Tvůrci takto zabezpečených aplikací využívají auditovaných komponent třetích stran, metodik pro ověřování bezpečnosti, penetračních testů a nástrojů pro statickou analýzu kódu i validaci aplikace. Úroveň zahrnuje zabezpečení proti sofistikovaným a specializovaným útokům, typicky prováděným cílenými odbornými útočníky. Dále se věnuje principům dobré architektury aplikací z pohledu bezpečnosti a způsobům zmírňování dopadů úspěšných útoků. Aplikace této úrovně zabezpečení předpokládají cílené útoky zainteresovaných skupin, předcházejí útokům jak vnějšího tak vnitřního charakteru. Věnují se také možnosti úspěšných útoků a způsobům rekonvalescence, odhalování probíhajícího útoku, obnovám ztrát a sledování útoku pro poučení. Útoky standardní úrovně běžně nejsou prováděny náhodnými amatéry, během plošných útoků v Internetu, ani běžnými jednotlivci profesionály. Hlavní skupinou útočníků jsou profesionální a placené specializované skupiny.

Tuto úroveň by měla splňovat každá aplikace zasahující velmi široké publikum uživatelů (v řádech od stovek tisíců) – jako aplikace státní správy, kritické bankovní aplikace, firemní aplikace obsahující kritické informace konkurenční výhody a především aplikace ohrožující zdraví a život osob.

Aplikace nesplňující plnohodnotně tuto úroveň (a všechny předcházející), s výjimkou zranitelností nerelevantních pro typ aplikace nebo platformu, je nutné považovat za nebezpečné pro použití ve: kritické aplikace státní správy – aplikace ohrožující zdraví a život osob, aplikace pracující s utajovanými skutečnostmi, vojenské a další, nebo kritické firemní aplikace finančních institucí.

Mezi typické zranitelnosti na této úrovni patří napadení administrátorů postranními kanály nebo časovanými útoky, sociální inženýrství, útoky masivními sítěmi a útoky hrubou silou velmi výkonným distribuovaným zdrojem aj.

## 7.1 Zranitelnost: Podsunutí do logů

Podsunutí do logů (z angl. „Log injection“) je variací zranitelností podsunutí, která cílí na interpret používaný při zpracování logů aplikace. Zranitelnost je irelevantní pro aplikace, které zobrazují logy v kontextu, ve kterém není jejich interpretace možná.

### 7.1.1 Princip útoku a obrany

První podmínkou pro úspěšný útok podsunutí logů je možnost útočníka vložit vstup do aplikace tak, aby byl uchován v záznamu chyby (resp. logu chyby). Druhou nutnou podmínkou je, aby logy byly zobrazeny administrátorem, analytikem nebo jiným uživatelem v kontextu, ve kterém je možné zneužít některý interpret – typicky ve webové aplikaci. Třetí podmínkou je pak příležitost připravit výstup do kontextu tak, aby byl interpretován cíleným interpretem a spuštěn jako kód.

Optimální, avšak pouze teoretickou možností je zamezení tisku veškerých uživatelských vstupů do výstupu logů. Obvykle je potřeba alespoň část vstupu uživatele zapsat a posléze zobrazit. Možnost nezobrazovat logy v kontextu jakéhokoliv interpretu je komplikovaná. Takový kontext je obtížné zajistit, i obyčejný textový soubor může být teoreticky nebezpečný. Navíc omezení z toho vyplývající přináší negativa, jako nedostatečná analýza logů. Hlavní obranou pak zůstává ošetření výstupu do vybraného kontextu tak, aby nemohl být interpretován.

### 7.1.2 Kategorizace

Kategorie: [A], [C], [D], [F]

#### *7.1.2.1 Severita, dopad*

| **Severita** | **A+, Kritická** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | - | - [není proveditelný] |
| Odborný náhodný | - | - [není proveditelný] |
| Odborný cílený | A | Data, Systém, Služba, Účet, Osoba |
| Profesionální cílený | A+ | Data, Systém, Služba, Účet, Osoba |

*Tabulka 7.1: Severita a dopad podsunutí do logů, zdroj autor*

#### *7.1.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizené informace, zneužívají a obchodují napadené privilegované účty, vyhrožují kompromitací aplikace, znehodnocují celý ekonomicko-společenský účel aplikace za úplatu, zneužívají uživatelů k napadání třetích stran, šíření virů, obsahu.

Vlastníci: ztrácí postavení na trhu, platí smluvní pokuty a penalizace za ztrátu dat, kompromitaci klientů, přicházejí o klientelu v důsledku poškození účelu aplikace, jsou spojováni s dopady útoku (z pohledu uživatele útok provádí aplikace), jsou žalováni za napadení třetích stran, ukončují svou činnost

Třetí strany: platí za cílené poškození (reputace) konkurence, odcizení dat s cílem průmyslové špionáže, napadení uživatelů, zneužití prostředků prohlížečů, platí za šíření virů, dat, převzetí kontroly nad infrastrukturou a její poškození, znehodnocení aplikace.

### 7.1.3 Způsob prověření (provedení)

Příkladem nejblíže kontextu této práce je napadení zobrazení logů ve webové aplikaci a cílení na interpret jazyka javascript. Jedná se tedy prakticky o modifikovanou zranitelnost podsunutí skriptu z kapitoly 5.1. Stejný nebo ještě kritičtější útok je však možné provést i na jiné aplikace pro prohlížení logů (například na silného klienta a logy v databázi).

Útočník se bez úspěchu pokusí o sadu útoků standardním podsunutím skriptů. Pokusí se tedy ještě o podsunutí skriptu do logu. Aplikace zpracovává číselný vstup uživatele tak, že se jej pokusí přeložit jako číslo pomocí `Int32.Parse(id)`. Pokud parsování selže (odchycení), je zalogována chyba o parsování včetně originálního vstupu. Do položky, která má obsahovat číslo je

útočníkem vložen řetězec z vycházející z principu z kapitoly 5.1.3:

```
<%tag onmouseover="alert('xss')">hover here
```

Tento řetězec prochází nativní obranou frameworku a zároveň i standardní ochranou proti podsunutí skriptů, jelikož neútočí přímo na běžný uživatelský výstup. Následně administrátor navštíví správcovské rozhraní pro zobrazení logů chyb. Chyba parsování je mezi nimi. Správcovská rozhraní jsou často méně zabezpečena, jelikož je nepoužívá „běžný uživatel“ a z toho důvodu jsou také vhodným cílem útoku.

V administračním rozhraní je použit stejný mechanizmus načtení obsahu do stránky jako ten, popsaný v kapitole 5.1.5.1. Kód je v takovém okamžiku interpretován jako HTML, respektive javascript. Administrátor může být pak typicky napaden útoky zaměřenými na relaci, padělání požadavků nebo únos kliknutí popsanými v předchozích úrovních. Útok primárně spoléhá na často méně zabezpečované administrační rozhraní a privilegovaný účet jeho uživatelů.

Při identifikaci potenciálních míst pro „podsunutí do logů“ je vhodné hledat: standardní místa napadení aplikace podsunutím skriptů s tím rozdílem, že cílem vložení není samotný výstup, ale zpracování výjimky a její uchování v logu aplikace, v kódu pak způsob zpracování výjimek (primárně centrální mechanizmus ukládání logů), ukládání autentizačních a autorizačních rozhodnutí do logů a další body, kde je vstup uživatele nutné v logu uchovat.

### 7.1.4 Způsob obrany

Obrana vychází ze stejného principu jako 5.1.4, přičemž je vhodné administrátorské rozhraní a zobrazování vstupů od uživatele bránit mechanizmy, které nejsou pro běžné uživatele akceptovatelné, jako například:

- Zákaz veškerého skriptování ve správcovském rozhraní,
- nejstriktnější možná politika CSP,
- omezení grafických úprav výstupu,
- validace výstupů pomocí kontrolních součtů (hash výstupu),
- poučení administrátorů a privilegovaných uživatelů o možných nebezpečích a konfigurace jejich pracovního prostředí nejstriktnějším způsobem (zákaz skriptů, zákaz interpretace zdrojů třetích stran, zákaz rámů, zákaz doplňků, flash, zákaz doplňků zobrazujících PDF a další), instalace bezpečnostních doplňků do prohlížečů,
- využití speciálních prohlížečů pro administrační rozhraní, například textového režimu (jako prohlížeč linx), alternativních operačních systémů a další opatření.

Takto zabezpečený klientský stroj administrátora nemůže být snadno napaden žádným z dostupných způsobů podsunutí skriptů nebo jiných libovolných prostředků.

### 7.1.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.AccountController metoda EditUserPhone<br>T1_SecuBank.Controllers.AdminController metoda GetLogs |

| | |
|---|---|
| | T1_SecuBank.Views.Admin.DisplayLogs |
| **Oprava** | T1_SecuBank.Controllers.AdminControllerSafe metoda GetLogs<br>T1_SecuBank.Views.AdminSafe.DisplayLogs |

#### *7.1.5.1 Ukázkové použití: zranitelné*

Aplikace se během změny telefonního čísla pokouší provést parsování prvních tří znaků čísla do číslice. Pokud se parsování nepodaří, je uložena výjimka do logů dle následujícího úryvku:

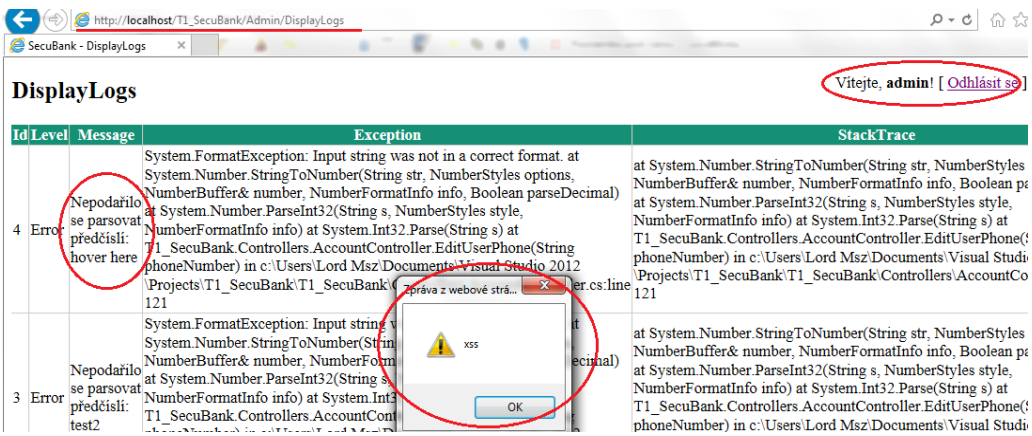
```
try
{
 string pre = phoneNumber.Substring(0, 3);
 int preNum = Int32.Parse(pre);
...
 catch (Exception e)
 {
  ILogger logger = new DefaultLogger();
  logger.Error("Nepodařilo se parsovat předčíslí: {0}", e, phoneNumber);
```

*Poznámka autora: z logického hlediska se nejedná o praktický příklad. To nemění nic na faktu, že jakákoliv takto uložená výjimka z reálné praxe je dostatečně vhodná.*

Pokud útočník vloží kód z předchozí kapitoly, projde nativní validací frameworku, je vyvolána výjimka a hodnota je uložena do úložiště logů (v tomto případě databáze). V administračním rozhraní má pak uživatel k dispozici funkci pro zobrazení nejnovějších logů v tabulce. Data tabulky jsou načítána asynchronně následujícím javascriptem:

```
$.ajax({ type: "POST", url: "...", success:
function (data) {...
 for (var i = 0; i < data.length; i++) {...
  container.append("<tr>" +
      "<td>" + d.Id + "</td>" +...
      "<td>" + d.Message + "</td>" +...
```

Následující příklad zobrazuje obsah stránky úspěšně napadeného administrátora:




*Ilustrace 7.1: Úspěšný útok na administrátora aplikace pomocí podsunutí do logů; prohlížeč Internet Explorer 9 (emulace), zdroj autor*

#### *7.1.5.2 Ukázkové použití: bezpečné*

Zabezpečená verze stránky nevyužívá k vykreslení tabulky logů javascriptové asynchronní volání. Namísto něj vykresluje obsah na serveru – je tedy možné vypnout při přístupu na výpis javascript v prohlížeči. Jelikož je obsah vykreslován `helperem Html`, je výstup zakódován. Navíc je akce opatřena atributem `ContentSecurityPolicyFilter`, který omezuje spouštění javascriptů obsažených ve stránce, přičemž administrátoři aplikace mohou být dostatečně poučeni o nutnosti instalace a užití nejnovějších prohlížečů.

## 7.2 Zranitelnost: Nedostatečná anti-automatizace

Nedostatečná anti-automatizace (z angl. „insufficient anti-automation“) je zranitelnost orientovaná na využití hrubé síly k prolomení zabezpečení aplikace. Zranitelnost je irelevantní pro aplikace, které nevyužívají žádný obranný prvek, který lze napadnout útokem hrubou silou.

### 7.2.1 Princip útoku a obrany

Pro útok na nezabezpečenou aplikaci není nutné splnit žádné předpoklady. Pokud je možné napadnout jakýkoliv mechanizmus aplikace hrubou silou, ať už jde o hledání identifikátorů dat nebo hesel uživatelů, útočník může útok provést.

Obrana je zaměřena na zpomalování útoků tak, aby nebylo jejich provedení praktické. Obranné mechanizmy jsou také často umístěny mimo hranice aplikace – na úrovni platformy. Zde dochází k heuristickému rozhodování a vyřazování podezřelých nebo příliš častých požadavků.

### 7.2.2 Kategorizace

Kategorie: [D], [B.1], [A.3]

#### *7.2.2.1 Severita, dopad*

| **Severita** | **A+, Kritická** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | C | Data, Služba |
| Odborný náhodný | B | Data, Služba, Účet |
| Odborný cílený | A | Data, Služba, Účet, Osoba |
| Profesionální cílený | A+ | Data, Služba, Účet, Osoba |

*Tabulka 7.2: Severita a dopad nedostatečné anti-automatizace, zdroj autor*

#### *7.2.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: obchodují odcizená data (především databáze hesel a uživatelů), uživatelské účty, vyhrožují kompromitací aplikace, zpomalují, poškozují nebo zcela znemožňují funkci aplikace, znehodnocují celý ekonomicko-společenský účel aplikace za úplatu. Dále riskují finanční postihy, žaloby a trestní řízení.

Vlastníci: ztrácí postavení na trhu, platí smluvní pokuty a penalizace za ztrátu dat, kompromitaci klientů, přicházejí o klientelu v důsledku poškození účelu aplikace, ukončují svou činnost.

Třetí strany: platí za cílené poškození (reputace) konkurence, odcizení dat, automatické extrakce, napadení administračních účtů.

### 7.2.3 Způsob prověření (provedení)

Základním příkladem nedostatečné obrany je možnost útočníků ověřovat velké množství uživatelských jmen a hesel, vytvářet tisíce nových záznamů, odesílat velké množství notifikací, nebo objednávat tisíce kusů zboží. Typicky se jedná o dva druhy útoků (zdroj [29]):

- Útok do výšky (základní útok hrubou silou),
  - je otestování velkého množství hodnot (hesel) na jednom vstupu (uživateli).
- útok do šířky (distribuovaný útok hrubou silou),
  - je otestování jedné hodnoty (např. hesla) na širokém spektru vstupů (např. uživatelů).

Ačkoliv provedení působí obdobně, je nutné aplikaci zabezpečit proti oběma. Pokud aplikace brání otestování všech hesel na jednom uživatelském účtu, ještě nebrání otestování jednoho hesla na všech uživatelích a naopak. Útočník tedy může použít seznam například 50 000 nejpoužívanějších hesel a pokusit se je aplikovat na jeden známý uživatelský účet. Nebo se může pokusit vzít první nejpoužívanější heslo a pokusit se jej aplikovat na 50 000 uživatelských účtů. S každým dalším pokusem v obou případech stoupá pravděpodobnost úspěchu.

Výše uvedené případy pak platí stejně i u funkcí, které neznamenají odcizení účtu uživatele. Typicky jde o založení nového záznamu (jeden uživatel vloží milion nových produktů do košíku za minutu), vyhledání položky, nebo zobrazení detailu (jeden doktor zobrazí tisíce pacientů za den).

Při identifikaci potenciálních míst pro „nedostatečnou anti-automatizaci“ je vhodné hledat: libovolné formuláře nebo akce napadnutelné hrubou silou (přihlašování, vyhledávání, vytvoření uživatelského účtu), identifikátory dat odhalitelné hrubou silou (zobrazení detailu), v kódu pak mechanizmus pro omezení počtu požadavků jedním uživatelem nebo na jednu datovou položku.

### 7.2.4 Způsob obrany

Následující seznam popisuje mechanizmy obrany od základních:

- Exponenciálně se zvyšující zpoždění při přístupu k jednomu bodu,
  - úspěšný útok hrubou silou zcela vyřadí už například jednosekundové zpomalení po zadání prvního chybného hesla,
  - pokud se zpoždění navyšuje s přibývajícím počtem pokusů (exponenciálně, lineárně, s horním limitem), je útok hrubou silou na jednoho uživatele zcela nepraktický,
- zvyšující se zpoždění při distribuovaném přístupu k aplikaci,
  - distribuovaný útok je nutné sledovat nikoliv pro každého uživatele zvlášť, ale na úrovni celé aplikace,
  - pokud je globálně v aplikaci sledován průměrný počet pokusů o přihlášení a adekvátně

s ním je navyšováno globální zpoždění pro všechny uživatele, je distribuovaný útok zcela nepraktický,

- zvyšující se zpoždění pro individuální požadavky a skupiny,
  - v případě komplexního zabezpečení je možné sledovat požadované prostředky a zamezit jednotlivým uživatelům (relacím) požadovat velké množství prostředků,
  - stejně tak je možné sledovat globálně v aplikaci užití jednoho libovolného prostředku a korelaci mezi uživateli (resp. skupinami) a hodnotami požadavku,
  - tuto situaci typicky řeší systémy mimo aplikaci na úrovni platformy.

### 7.2.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.AccountController metoda Logon |
| **Oprava** | T1_SecuBank.Controllers.AccountController metoda LogonSafe<br>T1_SecuBank.Infrastructure.MVC.Filters.<br>ResourceGovernorFilterAttribute<br>T1_SecuBank.Infrastructure.Security.ResourceGovernor |

#### *7.2.5.1 Ukázkové použití: zranitelné*

Aplikace během standardního přihlášení neověřuje, zda se uživatel pokouší přihlásit chybným heslem vícekrát. Pokud tedy útočník vytvoří jednoduchý skript, může pro vybraného uživatele ověřit velké množství hesel a případně uspět útokem hrubou silou.

V ukázkové aplikaci lze tento fakt ověřit několika násobným odesláním formuláře za sebou v rychlém sledu. Aplikace tedy neobsahuje žádný ochranný prvek.

#### *7.2.5.2 Ukázkové použití: bezpečné*

Do aplikace je doplněn mechanizmus pro sledování a zabránění jednomu z typů útoku hrubou silou – konkrétně obrana proti útoku do výšky. Tento mechanizmus se snaží zabránit ozkoušení velkého množství požadavků v řadě shodujících se ve vybraném parametru. Akce pro bezpečné přihlášení je označena následujícím atributem:

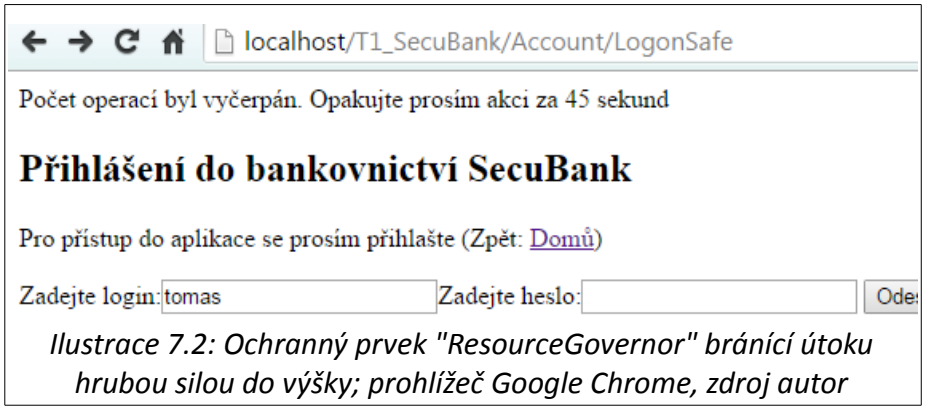
```
[ResourceGovernorFilter("login")]
public ActionResult LogonSafe(string login, string pass...)
```

Tento atribut je základní (jednoduchou) implementací pro ověření daného požadavku pomocí správce prostředků (tzv. „ResourceGovernor“). Na základě zadaných parametrů a kontextu požadavku sestaví identifikátor vybrané akce a odešle požadavek na vyhodnocení správci. Ten na základě uchovaných dat vyhodnotí, zda je tento požadavek validní nebo není a vrací případné zpoždění, tedy čas, který musí uplynout, než je povolena další shodná akce.

```
public bool ExceedsBudget(object paramValue, ...){...
  ... cutDates = dates.Where(d => d > DateTime.Now.AddSeconds(-1*
MAX_SECOND_TRESHOLD)).OrderByDescending(d => d).ToList();
 if (cutDates.Any()) {...
  delay = CalculateDelay(...lastHit).TotalSeconds, cutDates.Count());
  if (delay > 0) {...
   return true;
```

Pokud má tedy správce prostředků uložen vybraný požadavek mezi posledními shodnými požadavky, které proběhly během posledních sekund (`MAX_SECOND_TRESHOLD`), je vyhodnocen čas, o který má být požadavek zpožděn a je vrácen. Filtr uvedený výše pak odpověď přesměruje na stejnou adresu a zapíše čekací lhůtu. Aplikace tedy dotaz vůbec nezačne provádět.

*Poznámka autora: tato implementace je pouze ukázková a není zdaleka optimální. Jako vstupní bod však může posloužit.*




*Ilustrace 7.2: Ochranný prvek "ResourceGovernor" bránící útoku hrubou silou do výšky; prohlížeč Google Chrome, zdroj autor*

Obranu proti distribuovanému typu útoku je pak možné snadno doplnit rozšířením sledovaných atributů o heslo (což není příliš bezpečné) nebo přidáním globálního sledování požadavků přímo ve třídě `ResourceGovernor`.

**Ověření**

Následujícím javascriptovým kódem je možné ověřit funkčnost řešení při zátěži. Jelikož akce přihlášení není omezena na formuláře odeslané uživatelem, je možné se pokoušet přihlásit asynchronním dorazem javascriptu – v praxi je vhodné tuto variantu zakázat:

```
setInterval(function(){
    $.ajax({
        type: "POST",
        url: "/T1_SecuBank/Account/LogonSafe/",
        data: {login: '[vybrany_login]', password: '[heslo]'},
        success: function (data) { $("body").html(data); } });
}, 500);
```

## 7.3 Zranitelnost: Nedostatečné logování

Nedostatečné logování (z angl. „Insufficient logging“) je skupina nedostatků umožňující mimo jiné skrývání útoků a omezení sledování útočníků. Zranitelnost je relevantní pro všechny aplikace. Tato zranitelnost je primárně orientována na dopady jiných úspěšných útoků a jejich zmírňování a předcházení.

### 7.3.1 Princip útoku a obrany

Pro úspěšné zneužití této slabiny útočník nemusí plnit žádná kritéria. Na akci, kterou hodlá napadnout jedním z možných útoků není korektně aplikováno uchování záznamů o aktivitě, chybách a podezřelém chování uživatelů – tzv. „logování“. Záznam není ukládán vůbec, je ukládán jen ve výjimečných případech, nebo je možné se jeho uchování vyhnout.

Způsobů, jak napravit tuto chybu je velké množství. Centrální ukládání logů v případě všech aplikačních výjimek (viz 6.4), centrální ukládání detailních logů napříč aplikací v případě podezření z napadení (manuální spuštění), speciální logování všech kritických funkcí aplikace a další. Specifika má pak také způsob uchování a distribuce logů, jako uchování v databázi (kritické a citlivé), duplikování do textových souborů (základní), nebo zápis do systémových událostí. Posledním faktorem pro úspěch obrany je pak vhodné prostředí pro vyhledání a analýzu uložených logů.

### 7.3.2 Kategorizace

Kategorie: [D], [B], [F], [A.3], [C.2]

#### *7.3.2.1 Severita, dopad*

| **Severita** | **A, Vysoká** | |
|---|---|---|
| | | |
| *Úroveň útoku* | *Severita* | *Dopady* |
| Základní/amatérský | - | - [není proveditelný] |
| Odborný náhodný | C | Služba |
| Odborný cílený | C | Služba |
| Profesionální cílený | A | Systém, Služba |

*Tabulka 7.3: Severita a dopad nedostatečného logování, zdroj autor*

#### *7.3.2.2 Ekonomické aspekty úspěšného útoku*

Útočníci: skrývají svou aktivitu, zneužívají nemožnosti odhalení správci, přetěžují a zpomalují systém beze stopy.

Vlastníci: ztrácí důvěru uživatelů, ztrácí kontrolu nad aplikací, vykazují ztrátu z důvodu vysokých nákladů na nápravu této systémové chyby.

Třetí strany: platí za cílené poškození (reputace) konkurence, za zpomalování a vyřazení aplikace v konkurenčně kritické okamžiky, za cílené průběžné poškozování aplikace.

### 7.3.3 Způsob prověření (provedení)

Příkladem je libovolná akce významná z pohledu zabezpečení aplikace, která není uložena

v libovolné podobě ke zpětnému dohledání. *U všech uvedených bezpečnostních problémů od Náležité až po Elitní úroveň je vysoce důležité uchovávat záznam.* Ať už se jedná o jakýkoliv drobný pokus o podsunutí skriptů nebo SQL, o napadení relací nebo obcházení autorizace, či snad masivní automatizovaný distribuovaný útok na aplikaci, uchované záznamy jsou následně to jediné, co mohou tvůrci a správci aplikace využít, jako zdroj informací.

Uchovávání logů má pochopitelně význam i mimo bezpečnostní kontext. Především je díky nim možné dohledat chyby aplikační logiky a řešit problémy, které uživatelé s aplikací mají. Jejich význam mnohdy převyšuje i nejlepší zabezpečovací mechanizmy.

Při identifikaci potenciálních míst pro „nedostatečné logování“ je vhodné hledat: místa v aplikaci, která slouží jako vstupní body pro libovolný útok (tzv. „vektory“), formulářové akce, asynchronní dotazy, zpracování cookies, parametrů dotazu, v kódu pak způsob uchování záznamu o chování uživatelů v takových místech, tedy záznamy o neúspěšném přihlášení, o pokusu o přístup k zabezpečenému zdroji aj.

### 7.3.4 Způsob obrany

Následující seznam popisuje vhodné mechanizmy logování od základních:

- Při zpracování důležité akce je využit blok `try-catch-finally`,
- při zpracování každé akce je využit uvedený blok,
- logy jsou uchovány v paměti,
- při průchodu požadavku aplikačními vrstvami je uchován záznam o výjimkách každé úrovně,
- při vyvolání výjimky, kterou aplikace neošetřuje je použit centrální mechanizmus jejího zpracování a uchování,
- logy jsou uchovány v textových a datových souborech, nebo specifickém formátu pro zpracování vybranou aplikací,
- při provádění všech bezpečnostních rozhodnutí je uveden log,
- při spuštění kteréhokoliv obranného prvku aplikace je uchován log,
- při požadavku na libovolnou akci je možné manuálně přepnout aplikaci do režimu detailního logování, kdy správce může precizně analyzovat podezření z provedeného útoku,
- logy jsou uchovány v databázi nebo obdobných analyzovatelných úložištích,
- logy jsou uchovány na více místech, jsou distribuovány podle své povahy a kritičnosti, různé správcovské role mohou analyzovat různé úrovně logů (síťové, serverové, aplikační logické, aplikační bezpečnostní), provádět opakovatelné korelační a další statistické analýzy.

### 7.3.5 Ukázka

| | |
|---|---|
| **Projekt** | T1_SecuBank |
| **Hlavní zdroj problému** | T1_SecuBank.Controllers.AccountController metoda Logon<br>T1_SecuBank.Controllers.MessageController metoda SendMessage<br>T1_SecuBank.Infrastructure.Security.ResourceGovernor |

| **Oprava** | T1_SecuBank.Infrastructure.MVC.Filters.<br>LogAndHandleExceptionFilterAttribute |
|---|---|

#### *7.3.5.1 Ukázkové použití: zranitelné*

Prakticky celá aplikace T1_SecuBank je primárně vytvořena bez uchování logů. Při pokusu o jednotlivé útoky není uchován na serveru, krom výchozích záznamů serveru IIS, žádné stopy. Pokud tedy útočník využije podsunutí SQL neúspěšně, nikdo se o jeho činnosti nedozví (akce `Logon`, `AccountController`). V případě pokusu o podsunutí skriptů není krom samotné uložené zprávy uchován žádný záznam o tomto útoku (akce `SendMessage`, `MessageController`). V případě útoku hrubou silou na aplikaci není uchován seznam IP adres útočníků, napadených uživatelů, ani úspěšnost nebo neúspěšnost útoku (třída `ResourceGovernor`).

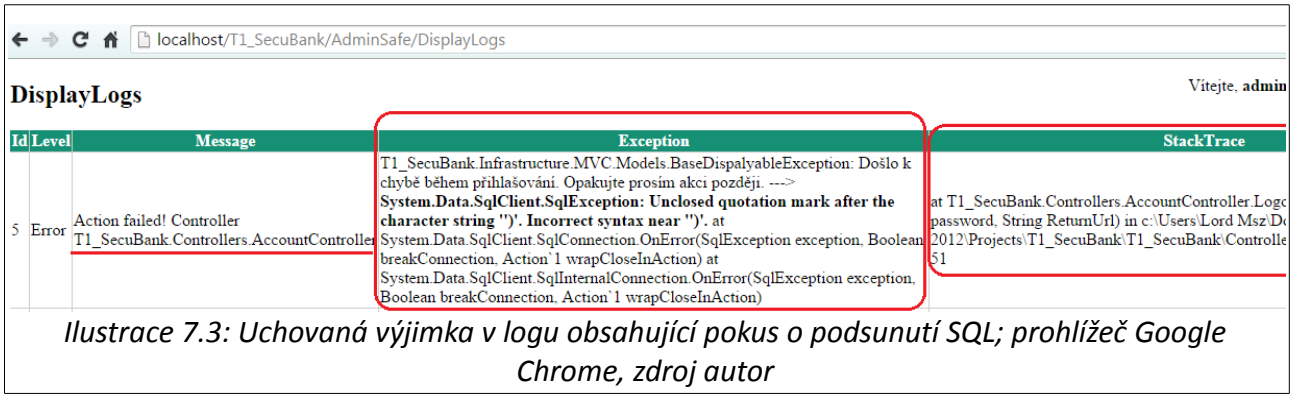
Z výše uvedených příkladů je zřejmé, že správci takové aplikace nemají dostupné informace pro zmírnění útoků nebo obnovu.

#### *7.3.5.2 Ukázkové použití: bezpečné*

Aplikace doplňuje výchozí mechanizmus zpracování výjimek o logování. Každá nezpracovaná výjimka aplikace je uchována v logu společně s dalšími informacemi:

```
public override void OnException(ExceptionContext filterContext)
{...
 _logger.Error("Action failed! ...", filterContext.Exception, ...);
```

Pokud se útočník pokusí například o podsunutí SQL, je pravděpodobné, že bude alespoň jednou neúspěšný a dojde k výjimce zpracování jeho dotazu. Ta je uložena a při analýze logů je možné dohledat minimálně bezpečnostní tuto zranitelnost. Následující obrázek obsahuje ukázku administračního rozhraní se zobrazením logů a vyznačeným pokusem o útok:




*Ilustrace 7.3: Uchovaná výjimka v logu obsahující pokus o podsunutí SQL; prohlížeč Google Chrome, zdroj autor*

# 8 . Standardy bezpečnosti webových aplikací

Tato kapitola se zabývá vazbou metodiky na jiné dostupné standardy a dokumenty v oboru, primárně uvádí mapování na zdroje popsané v kapitole 3.2.

## 8.1 Mapování

Kapitola uvádí odkazy na mapování mezi touto metodikou a dostupnými rozšířenými metodikami a zdroji. Jelikož jsou mapovací tabulky velmi rozsáhlé, jsou umístěny na přiloženém CD práce. Mapování jsou uložena ve formátu „.ods“ (program „Libre Office Calc“) a „.pdf“.

Následující tabulka popisuje umístění mapování:

| Mapování | Umístění |
|---|---|
| Mapování s projektem OWASP TOP 10 | CD/prilohy/mapovani/owasp_top_10.ods (.pdf) |
| Mapování s projektem OWASP ASVS | CD/prilohy/mapovani/owasp_asvs_10.ods (.pdf) |
| Mapování s CWE | CD/prilohy/mapovani/cwe.ods (.pdf) |
| Mapování s CWE/SANS TOP 25 | CD/prilohy/mapovani/cwe_sans_top25.ods (.pdf) |

*Tabulka 8.1: Umístění mapování metodiky na zdroje, zdroj autor*

## 8.2 Závěry mapování

Každá z uvedených dostupných metodik pokrývá relativně široké spektrum bezpečnostních otázek. Jednotlivé v nich popsané zranitelné body jsou navíc často velmi objemné a zasahují do daleko širší oblasti, než je zde vymezený obor zranitelností běžných webových aplikací. Výsledkem mapování je částečné pokrytí těchto zdrojů, ale zároveň jejich výrazné doplnění na poli praktických ukázek a přesných a konkrétních postupů prokázání slabiny.

Projekt OWASP ASVS je dostatečně relevantně zpracován ve většině popisovaných témat. Nedostatečné vazby se dostává především čistě nízkoúrovňovým problémům webových formulářů (jako konkrétní doporučené HTML prvky stránky), nebo zranitelné aplikační logice, typicky zabezpečení formulářů pro obnovu hesel. Naopak OWASP TOP 10 je metodikou ve většině případů rozšířen na příkladech pro vybranou platformu. K největším zúžením dochází u zranitelností, které zasahují velké množství různých technologií – například „A1 – Injection“ zabývající se mj. „LDAP injection“, „XML injection“ a mnoha dalšími.

Z taxonomie CWE jsou vhodně vybrány klíčové kategorie zranitelností a popsány z velké části typicky webové problémy. Ty jsou navíc výrazně doplněny o konkrétní příklady a kód. Naopak u obecných zranitelností, týkajících se například logiky aplikace, je prostor pro zlepšení. Pokrytí této masivní kolekce je zdánlivě malé, nicméně po vyřazení kategorií, které nesouvisí přímo s webovou platformou (jako jsou problémy s interpretací ukazatelů) je míra pokrytí výrazně vyšší. Zranitelnosti z CWE/SANS TOP 25 jsou ve velké míře obsaženy v této metodice. Jelikož se jedná prakticky o podmnožinu CWE a vzhledem k jejímu zaměření, je vazba mezi touto prací a tímto zdrojem silná.

# 9 . Závěr

Webové aplikace za dobu své existence změnily celý dnešní svět. Je však důležité uvědomit si, že ne všechny změny jsou vždy pro všechny přínosné. Bezpečnost na webu je jedním z velkých témat dnešního světa a je otázkou, zda v budoucnu bude inklinovat spíše ke zvyšování povědomí, zkvalitňování platforem a aplikací, nebo povede k větším obavám, obezřetnosti a celkově k uvržení tohoto realizovaného snu, sítě znalostí, informací, zábavy a lidí, do temného stínu nebezpečí.

Hlavní cíl práce, vytvoření ucelené metodiky pro ověření bezpečnosti webové aplikace, plní především kapitoly 4 až 7. Ty popisují nejen vytyčený obsah, tedy zranitelnosti a jejich ukázky, ale navíc obsahují velké množství nových informací a poznatků, nedokumentovaných způsobů prolomení obrany i zamezení útoku, které nejsou uvedeny v žádném z autorovi dostupných zdrojů. Některé útoky nemají na cílené platformě vůbec dostupný odpovídající obranný mechanizmus a jsou tedy velmi nebezpečné. Všechny útoky jsou navíc provedeny na konkrétním příkladu a existující ukázkové aplikaci a jsou tedy ověřeny proti uvedeným verzím platformy.

Druhotný cíl, tedy shrnutí maximálního možného počtu zranitelností z vícero zdrojů popisuje a jeho plnění specifikuje kapitola 3. Využité zdroje patří k nejobsáhlejším a nejnovějším poznatkům v oboru a značná část jejich obsahu je do metodiky zahrnuta. Přesto metodika neobsahuje zdaleka všechny zranitelnosti, které do ní teoreticky patří, a vzniká tedy prostor pro další navázání na tuto práci.

Za hlavní přínos práce považuje autor některé vybrané útoky, které jsou dokumentovány v dostupných zdrojích pouze teoreticky a proto jsou praxí podceňovány. Mezi ně se řadí například popis přímého útoku na relace v kapitole 5.3 nebo podsunutí skriptů v kapitole 5.1. První z nich popisuje konkrétní analytický postup útoku na nejrozšířenější autentizační mechanizmus dané platformy (ve starší verzi), rozebírá bezpečnostní identifikátor v něm používaný, identifikuje jeho zranitelné části a zneužívá těchto nových poznatků ke změně identity uživatele a kompromitaci aplikace. Tento postup je v práci zcela původní. Kapitola uvádí také možný vývoj této kompromitace i na nejnovější verzi platformy. Druhá zmiňovaná kapitola popisuje podrobně možnosti zneužití skriptů k útoku na klientských strojích. Kromě popisu nejnovějších poznatků v této oblasti se věnuje také klíčové zranitelnosti na cílové platformě. Jedná se o útok, který obchází vestavěné obranné mechanizmy, na které často vývojáři spoléhají. Navíc je aplikován ve zcela běžném použití v aplikaci – jde tedy o kritickou zranitelnost nebezpečnou v praxi.

Práce krom uvedených příkladů a zranitelností obsahuje sadu teoretických konceptů nebo formálních specifikací. Kategorizace a formální popis zranitelností metodiky je systematicky a přesně popisován v kapitole 3. Zde je věnována pozornost systematickému vzniku obsahu metodiky, kvalitě a hodnotě zdrojů a popisu pojmů užitých k zasazení zranitelností do kontextu. Díky popisu ekonomických a technických dopadů zranitelností a rozboru v úvodu každé úrovně metodiky mohou i ekonomicky zaměřené subjekty společnosti vyhodnotit nebezpečí přinášené každou z nich. Kategorie navíc zahrnují popis případných útočníků, jejich sofistikovanost, motivaci a cíle.

Pro usazení znalostí do vhodného teoretického pozadí vybírá kapitola 2 nejdůležitější koncepty a teoretické základy bezpečnosti webové platformy. Tento základ pak kromě porozumění útokům slouží k rozšíření sledovaného okruhu vývojáři, správci, architekty a analytiky infrastruktury během návrhu a realizace běhových prostředí aplikací. Cílem metodiky není pouze zamezit sadě

vybraných zranitelností, ale také zavést „nejlepší praxi“ do samotného průběhu návrhu a tvorby softwarových řešení a do procesů a organizace, ve kterých aplikace vznikají.

Na práci je možné navázat ve smyslu rozšíření metodiky o další zdroje, jako jsou taxonomie CAPEC (viz 13.2), CVE, NVD, HP Enterprise security a další. Tyto zdroje popisují rozsáhlou problematiku bezpečnosti aplikací jako celek. Vývojáři (resp. společnosti) pak mohou takové zdroje využívat pouze za relativně vysokých nákladů. Specializované metodiky, jako je tato, umožní velmi rychlou adaptaci bezpečných způsobů vývoje a jejich rozšíření.

Další možností je doplnění metodiky o systém hodnocení rizik, specifikaci nutných znalostí útočníků a vyhodnocení pravděpodobnosti jednotlivých útoků. Tato variace rozšíření je zaměřena převážně na netechnické subjekty společnosti, které využijí metodiku v rámci vyjednávání, vyhodnocení rizik, při odhadech nákladů a dalších činnostech. Přínos těchto výstupů se stále nejvíce odráží na práci samotných vývojářů, kteří mohou získat více času a prostředků pro kvalifikované a precizní zabezpečení aplikací, které používáme my všichni každý den.

# 10 . Terminologický slovník

| Termín | Zkratka | Význam [zdroj] |
|---|---|---|
| Active server pages | ASP | Serverová technologie pro tvorbu webových aplikací předcházející ASP.NET, autor |
| Adresa Internetového protokolu | IP | Číselný identifikátor počítače v síti používající protokol IP, autor |
| Architektura orientovaná na služby | SOA | Architektura aplikace složená primárně z autonomních služeb (typicky webových), autor |
| Extensible Markup Language | XML | Obecný značkovací jazyk vyznačující se rozšiřitelností, přenositelností a rozšířeností, autor |
| Hypertext Markup Language | HTML | Značkovací jazyk (v některých verzích vycházející z XML) používaný pro tvorbu webových prezentací a aplikací, autor |
| Hypertext Transfer Protocol | HTTP | Protokol pro přenos informací v rámci WWW, autor |
| IT governance | - | Je odpovědnost exekutivy a managementu a sestává z vedení, organizačních struktur a procesů zajišujících, že informační technologie společnosti podpoří strategii a cíle organizace. [1] |
| Java Enterprise Edition | JEE | Aplikační platforma jazyka Java určená pro tvorbu rozsáhlých, integrovatelných a robustních aplikací a webových aplikací |
| Java Server Pages | JSP | Serverová technologie pro tvorbu webových aplikací na platformě JEE, autor |
| Mashupové (webové) aplikace | - | Aplikace propojující více zdrojů a zobrazující data v rámci jednoho grafického rozhraní. [12] |
| Model View Controller | MVC | Architektonický vzor aplikací umožňující oddělení datových struktur, jejich reprezentace a ovládací logiky v rámci aplikace. [31] |
| Objektově relační mapování | ORM | Sada nástrojů pro překlenutí rozdílů mezi objektovým a relačním způsobem uchování dat, autor |
| PHP Hypertext Preprocessor | PHP | Otevřená serverová technologie pro tvorbu webových aplikací |
| Taxonomie | - | Utříděný soubor vědění vybrané oblasti uchovaný ve strukturované podobě, autor |
| Uniform Resource Locator a Identifier | URL a URI | Unikátní textový identifikátor zdroje, zejména v rámci sítě Internet |
| Webová služba | WS | Rozhraní pro komunikaci v heterogenním, autonomním a distribuovaném prostředí, autor |

*Tabulka 10.1: Terminologický slovník*

# 11 . Zdroje


1. ITGI. *Cobit 4.1*. Rolling Meadows, USA: IT Governance Institute, 2007. 196 s. ISBN 1-933284-72-2.
2. KÜMMEL, R. *XSS: Cross-Site Scripting v praxi : o reálných zranitelnostech ve virtuálním světě*. Zlín: Tigris, 2011. 330 s. s. ISBN ISBN 978-80-86062-34-1.
3. PEIRIS, CH.; MULDER, D. *Pro WCF: practical Microsoft SOA implementation*. New York City: APress, 2007. 475 s. ISBN 978-1-59059-702-6.
4. STUTTARD, D.; PINTO M. *The web application hacker's handbook: finding and exploiting security flaws.*. Chichester: John Wiley, 2011. 878 s. ISBN 978-1-118-02647-2.
5. ABOUL-ELA A. *Delete Credit Cards from any Twitter Account in ads.twitter.com* [online]. - : Hackerone, -2015, last modif. 9/2014 [cit. 3/2015]. Dostupný z WWW: <https://hackerone.com/reports/27404>.
6. APACHE SW FOUNDATION *Apache JMeter* [online]. - : Apache Software Foundation, 1999-2015, last modif. 1/2015 [cit. 1/2015]. Dostupný z WWW: <http://jmeter.apache.org/>.
7. AZIZ, A. *elmah - Error Logging Modules and Handlers for ASP.NET, Google Project Hosting* [online]. - : , 2011-2015, last modif. 3/2015 [cit. 4/2015]. Dostupný z WWW: <https://code.google.com/p/elmah/>.
8. BOYD I. *How did harmless crawler bypass WebForms authentication, and hijack a user's session?* [online]. New York : Stack Exchange Inc., 2015, last modif. 10/2013 [cit. 1/2015]. Dostupný z WWW: <http://stackoverflow.com/questions/19188717/how-did-harmless-crawler-bypass-webforms-authentication-and-hijack-a-users-ses>.
9. BROWN M.; CHRISTEY S. *CWE/SANS TOP 25 Most Dangerous Software Errors 2011* [online]. - : SANS Institute, 2011, last modif. 6/2011 [cit. 12/2014]. Dostupný z WWW: <http://www.sans.org/top25-software-errors/>.
10. CODENOMICON LTD. *The Heartbleed Bug* [online]. Oulu, Finland : Codenomicon Ltd., 2001-, last modif. 4/2014 [cit. 2/2015]. Dostupný z WWW: <http://heartbleed.com/>.
11. CRUZ, D. *Dinis Cruz Blog: Bypassing asp.net request validation detection, but it is a vulnerability?* [online]. - : OWASP, 2008-2015, last modif. 6/2014 [cit. 3/2015]. Dostupný z WWW: <http://blog.diniscruz.com/2014/06/bypassing-aspnet-request-validation.html>.
12. FICHTER D. *Library Mashups: Exploring New Ways to Deliver Library Data* [online]. Medford, N.J. : University of Saskatchewan Library, 2009, last modif. 2009 [cit. 12/2014]. Dostupný z WWW: <http://books.infotoday.com/books/Engard/Engard-Sample-Chapter.pdf>.
13. GOOGLE. *Chromium OS - The Chromium Projects* [online]. USA : Google inc., 1998 - , last modif. 2/2015 [cit. 2/2015]. Dostupný z WWW: <http://www.chromium.org/chromium-os>.
14. HICKSON I., BERJON R., W3C. *HTML5* [online]. - : W3C, 1994 - 2015, last modif. 10/2014 [cit. 3/2015]. Dostupný z WWW: <http://www.w3.org/TR/html5/>.
15. ISO *ISO/IEC 27000:2012* [online]. - : ISO, -2015, last modif. 1/2014 [cit. 4/2015]. Dostupný z WWW: <http://www.iso.org/iso/catalogue_detail?csnumber=56891>.
16. KAMKAR, S. *evercookie - virtually irrevocable persistent cookies* [online]. - : -, 2010-2015, last modif. 10/2010 [cit. 3/2015]. Dostupný z WWW: <http://www.samy.pl/evercookie/>.
17. KAZEROONI S., CUTHBERT D. *OWASP Application security verification standard* [online]. - : The

OWASP Foundation, 2014, last modif. 8/2014 [cit. 12/2014]. Dostupný z WWW: <https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project>.

18. LUCAS, I. *Password Recovery Speeds* [online]. UK : -, 1996-2009, last modif. 7/2009 [cit. 1/2015]. Dostupný z WWW: <http://www.lockdown.co.uk/?pg=combi&s=articles>.
19. MAONE, G., HUANG, D. *User Interface Safety Directives for Content Security Policy* [online]. - : W3C, 1994 - 2015, last modif. 11/2012 [cit. 4/2015]. Dostupný z WWW: <http://www.w3.org/TR/UISafety/>.
20. MICROSOFT *Security Considerations (Entity Framework)* [online]. Redmond : Microsoft Corp., 1999-, last modif. 2014 [cit. 1/2015]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/vstudio/cc716760(v=vs.100).aspx>.
21. MICROSOFT OPEN TECHNOLOGIES. *aspnetwebstack/AntiForgeryWorker.cs, GitHub* [online]. - : Microsoft Open Technologies, Inc., 2012 - 2015, last modif. 8/2013 [cit. 4/2015]. Dostupný z WWW: <https://github.com/ASP-NET-MVC/aspnetwebstack/blob/master/src/System.Web.WebPages/Helpers/AntiXsrf/AntiForgeryWorker.cs>.
22. MICROSOFT. *forms Element for authentication (ASP.NET Settings Schema)* [online]. Redmond : Microsoft corp., 1999-, last modif. 2014 [cit. 1/2015]. Dostupný z WWW: <https://msdn.microsoft.com/en-us/library/1d3t3c61(v=vs.100).aspx>.
23. MICROSOFT. *Welcome to TypeScript* [online]. Redmond : Microsoft, 2012-2014, last modif. 2014 [cit. 3/2015]. Dostupný z WWW: <http://www.typescriptlang.org/>.
24. MITRE. *CWE - Common Weakness Enumeration* [online]. Massachusetts : The MITRE Corporation, 2006-2015, last modif. 6/2014 [cit. 3/2015]. Dostupný z WWW: <http://cwe.mitre.org>.
25. MITRE. *CWE - Sources* [online]. Massechusetts : The MITRE Corporation, 2006-2015, last modif. 8/2014 [cit. 4/2015]. Dostupný z WWW: <https://cwe.mitre.org/about/sources.html>.
26. MÜLLER, J. *FreeMind* [online]. - : -, -2015, last modif. 4/2014 [cit. 4/2015]. Dostupný z WWW: <http://freemind.sourceforge.net/wiki/index.php/Main_Page>.
27. NHIBERNATE COMMUNITY. *NHibernate, The object-relational mapper for .NET* [online]. - : NHibernate Community, 2005 - 2015, last modif. 1/2015 [cit. 1/2015]. Dostupný z WWW: <http://nhibernate.info/>.
28. OPENSSL PROJECT. *OpenSSL: The Open Source toolkit for SSL/TLS* [online]. - : The OpenSSL Project, 1999-2014, last modif. 12/2014 [cit. 2/2015]. Dostupný z WWW: <https://www.openssl.org/>.
29. OWASP FOUNDATION. *The Open Web Application Security Project (OWASP)* [online]. US : OWASP Foundation, 2001-, last modif. 12/2014 [cit. 12/2014]. Dostupný z WWW: <http://www.owasp.org>.
30. PCI SECURITY STANDARDS COUNCIL, LLC. *Payment Card Industry (PCI) Data Security Standard, v3.0* [online]. US : PCI Security Standards Council, LLC, 2006-2013, last modif. 11/2013 [cit. 4/2015]. Dostupný z WWW: <https://www.pcisecuritystandards.org/documents/PCI_DSS_v3.pdf>.
31. REENSKAUG T.; COPLIEN J. *The DCI Architecture: A New Vision of Object-Oriented Programming* [online]. - : Artima, 2009, last modif. 3/2009 [cit. 12/2014]. Dostupný z WWW:

<http://www.artima.com/articles/dci_vision.html>.

32. ROSS, D., GONDROM, T. *HTTP Header X-Frame-Options* [online]. - : IETF, 1986-, last modif. 10/2012 [cit. 4/2015]. Dostupný z WWW: <http://tools.ietf.org/html/draft-ietf-websec-x-frame-options-01>.
33. SHAMIR A., GENKIN D., TROMER E. *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis* [online]. Tel Aviv : Tel Aviv University, Weizmann Institute of Science, 2004-2014, last modif. 12/2013 [cit. 2/2015]. Dostupný z WWW: <http://www.cs.tau.ac.il/~tromer/acoustic/>.
34. SOLON, O. *Apple was warned about iCloud vulnerability MONTHS before leaked celeb photos emerged* [online]. London, UK : Trinity Mirror, 2014, last modif. 9/2014 [cit. 11/2014]. Dostupný z WWW: <http://www.mirror.co.uk/news/technology-science/technology/apple-warned-icloud-vulnerability-months-4326913>.
35. U.S. DHHS. *Health Insurance Portability and Accountability Act (HIPAA)* [online]. US : U.S. Department of Health & Human Services, 2007-2013, last modif. 2013 [cit. 4/2015]. Dostupný z WWW: <http://www.hhs.gov/ocr/privacy/>.
36. VANSTECHELMAN, L. *Cookie replay attacks in ASP.NET when using forms authentication* [online]. - : -, 2005-2014, last modif. 9/2014 [cit. 3/2015]. Dostupný z WWW: <https://www.vanstechelman.eu/content/cookie-replay-attacks-in-aspnet-when-using-forms-authentication>.
37. WILLIAMS J.; WICHERS D. *OWASP TOP 10 - 2013: The ten most critical web application security risks* [online]. - : OWASP Foundation, 2013, last modif. 6/2013 [cit. 12/2014]. Dostupný z WWW: <https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project>.
38. ELIÁŠ, M. *Testování bezpečnosti webových aplikací*. Brno, 2008. Dostupný z WWW: <http://is.muni.cz/th/3958/fi_m/diplomova_prace_text.pdf>. Diplomová práce. Masarykova univerzita.
39. MENČÍK, J. *Aplikace e-learningu a bezpečnost dat*. Praha, 2014. Dostupný z WWW: <https://isis.vse.cz/zp/portal_zp.pl?podrobnosti=124697>. Bakalářská práce. Vysoká škola ekonomická v Praze.
40. PAVELEK, M. *Bezpečnost ASP.NET aplikací*. Praha, 2011. Dostupný z WWW: <http://www.unicorncollege.cz/european-it-center/pavelek-marek/attachments/BP_Marek_Pavelek.pdf>. Bakalářská práce. Unicorn College.
41. SŮVA, J. *Hodnocení bezpečnosti PHP aplikace podle standardu OWASP ASVS*. Praha, 2014. Dostupný z WWW: <https://isis.vse.cz/zp/portal_zp.pl?podrobnosti=113038>. Bakalářská práce. Vysoká škola ekonomická v Praze.

# 12 . Seznam obrázků a tabulek

## 12.1 Seznam tabulek



## 12.2 Seznam obrázků

# 13 . Přílohy

## 13.1 Příloha 1: XSLT transformace CWE

Z důvodu rozsahu je příloha umístěna na přiloženém CD:

CD/prilohy/zdrojovy_kod/cwe_xslt.xslt

## 13.2 Příloha 2: XSLT transformace CAPEC

Z důvodu rozsahu je příloha umístěna na přiloženém CD:

CD/prilohy/zdrojovy_kod/capec_xslt.xslt

## 13.3 Příloha 3: projekt T1_SecuBank

Projekt je společně s verzovacím repozitářem umístěn na přiloženém CD:

CD/projekty/T1_SecuBank.zip