Bezpečnost webových aplikací

Security of web applications

Lukáš Zemek

Bakalářská práce
2012

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2011/2012

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš ZEMEK**
Osobní číslo:	**A09795**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Bezpečnostní technologie, systémy a management**
Téma práce:	**Bezpečnost webových aplikací**

Zásady pro vypracování:

1. **Popište bezpečnostní problémy webových aplikací.**
2. **Problematiku prezentujte na příkladech.**
3. **Demonstrujte jednotlivá bezpečnostní rizika.**
4. **Navrhněte vhodnou obranu proti bezpečnostním rizikům.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. HOWARD, Michael a David LEBLANC. Bezpečný kód: [techniky a strategie tvorby bezpečných webových aplikací]. Vyd. 1. Brno: Computer Press, 2008, 895 s. ISBN 978-802-5120-507.
2. OWASP FOUNDATION. OWASP: The Open Web Application Security Project [online]. 2012 [cit. 2012-01-28]. Dostupné z: https://www.owasp.org/index.php/Main_Page
3. VEČEŘA, Zdeněk. Zdeněk Večeřa. Zdeněk Večeřa [online]. 2009 [cit. 2012-01-28]. Dostupné z: http://blog.zdenekvecera.cz/item/jak-na-to-sql-injection-magic_quotes_gpc-addslashes-a-stripslashes
4. Nette Framework: Zabezpečení před zranitelnostmi. NETTE FOUNDATION. Nette Framework [online]. 2012 [cit. 2012-01-28]. Dostupné z: http://doc.nette.org/cs/vulnerability-protection
5. VRÁNA, Jakub. Obrana proti SQL Injection. In: PHP triky [online]. 2005 [cit. 2012-01-28]. Dostupné z: http://php.vrana.cz/obrana-proti-sql-injection.php
6. GILMORE, Jason W. Velká kniha PHP 5 MySQL: kompendium znalostí pro začátečníky i profesionály. Vyd. 1. Brno: Zoner Press, 2005, 711 s. ISBN 80-868-1520-X.
7. SCHLOSSNAGLE, George. Pokročilé programování v PHP 5. Brno: Zoner Press, 2004, 640 s. ISBN 80-868-1514-5.

Vedoucí bakalářské práce: **doc. Ing. Martin Sysel, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **24. února 2012**

Termín odevzdání bakalářské práce: **25. května 2012**

Ve Zlíně dne 24. února 2012

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Mgr. Milan Adámek, Ph.D.
ředitel ústavu

ABSTRAKT

Tato práce se zabývá 10 nejčastějšími bezpečnostními hrozbami pro webové aplikace. Používá žebříček Top 10 2010 sestavený nadací OWASP. Každá zranitelnost z tohoto seznamu je ilustrována zranitelnými kódy a metodami útoku na tento kód. Dále jsou představeny adekvátní způsoby ochrany a zabezpečení. Ukázky zdrojových kódů jsou zpracovány v hojně rozšířeném skriptovacím jazyce PHP a technologii ASP.NET společnosti Microsoft.

Klíčová slova: OWASP, bezpečnost, zranitelnost, injection, cross-site scripting, crosite-request forgery, WebGoat, Top 10 2010

ABSTRACT

This paper deals with the 10 most common security threats to web applications. Uses Top 10 2010 rankings compiled by OWASP Foundation. Each vulnerability of this list is illustrated with the vulnerable code and attack methods in this code. In addition, methods are presented for adequate protection and security. Sample source codes are processed in a widely extended scripting language PHP and ASP.NET Microsoft.

Keywords: OWASP, security, vulnerability, injection, cross-site scripting, cross-site-request forgery, WebGoat, Top 10 2010

Tímto chci poděkovat vedoucímu bakalářské práce panu Ing. Martinu Syslovi, Ph.D. za odborné vedení, cenné rady a připomínky, které mi poskytoval při řešení této práce. Dále bych rád poděkoval svému okolí za podporu při psaní práce.

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

Internet a především jeho asi nejvyužívanější služba World Wide Web, služba pro poskytování webových stránek (WWW), prodělala za posledních několik let obrovský rozmach. V době svého vzniku Internet sloužil pouze armádním a akademickým účelům a v jeho prostředí se pohybovali pouze vzdělaní inženýři s čistými úmysly. V dnešní době se Internet stal dostupný masové veřejnosti a přístup k němu má v současnosti přes 60% domácností v České republice [1].

V důsledku masového rozvoje Internetu jako zcela nového média a v důsledku vzniku velkého množství služeb na Internetu je velmi aktuální otázka bezpečnosti. Vzhledem k povaze dat využívanými moderními webovými aplikacemi, by otázka bezpečnosti měla být zcela prioritní, ale ne ve všech případech tomu tak skutečně je.

Bezpečnostní chyby v aplikacích vznikají z různých příčin. Může to být v důsledku malé informovanosti programátora o možných bezpečnostních hrozbách a nezřídka pak také kvůli časovému tlaku na vydání aplikace a s tím souvisejícím nedostatkem prostoru pro testování aplikace z hlediska bezpečnosti. V neposlední řadě také z důvodu co nejnižší možné ceny a s tím opět souvisejícího tlaku na co nejkratší dobu vývoje. Otázka bezpečnosti bývá často řešena až po vývoji aplikace jako ne zcela podstatná. Tento přístup je nevhodný a otázky bezpečnosti by měly být brány v úvahu již při návrhu aplikace. Současně testování bezpečnosti by mělo probíhat paralelně s testováním funkcionality tak, aby zjištěné nedostatky byly odstraněny před finálním vydáním aplikace.

Ačkoliv rozdíl mezi programátorem webových aplikací a útočníkem resp. hackerem, nemusí být na první pohled zřejmý, jelikož oba musí mít výbornou znalost programovacích jazyků, algoritmů apod., je zde úhel pohledu, který tyto dvě „profese“ odlišuje. Programátor musí hlídat a ošetřit maximum možných slabých míst aplikace a věnovat velké úsilí k zamezení jejich vzniku. Útočníkovi pak stačí objevit většinou pouze jediné i drobné opomenutí a zneužít ho k narušení aplikace.

Cílem bakalářské práce je dostatečně popsat nejčastější bezpečnostní prohřešky v oblasti webových aplikacích a poskytnout informace pro adekvátní ochranu proti těmto hrozbám. Práce nepřináší zcela komplexní a ucelený seznam všech bezpečnostních rizik, protože tento výčet by byl prakticky neomezený, ale soustřeďuje se pouze na zranitelnosti identifikované jako nejzávažnější z pohledu svého rozšíření a dopadu na webové aplikace. K identifikaci nejběžnějších zranitelností je využit žebříček Top 10 2010 nadace OWASP.

Vzhledem ke stále stoupající důležitosti webových aplikací ve všech oblastech lidské činnosti je adekvátní zabezpečení dat uložených na webových stránkách a registrech stále důležitější.

Jednotlivé bezpečnostní problémy jsou rozebrány teoreticky a ilustrovány na konkrétních příkladech nebezpečného kódu. Práce nastiňuje postup, jak na tento kód zaútočit a jak bezpečnostní riziko v kódu eliminovat. Ukázky zdrojových kódů jsou zpracovány ve skriptovacím jazyce PHP[1] a v ASP. NET[2]. Databázové dotazy jazyka SQL[3] jsou připraveny pro databázi MySQL.

Práce by měla sloužit jako zdroj minimálních informací pro webové vývojáře, jak se vyvarovat bezpečnostních zranitelností u svých aplikací.

[1] rekurzivní zkratka – PHP: Hypertext Preprocessor – skriptovací jazyk k tvorbě dynamických webových aplikací
[2] součást .NET Frameworku pro tvorbu webových aplikací a služeb
[3] Structured Query Language – dotazovací jazyk k tvorbě dotazů v relačních databázích

1 OPEN WEB APPLICATION SECURITY PROJECT

Open Web Application Security Project (OWASP) je nevýdělečná organizace zabývající se zlepšením bezpečnosti softwaru zejména v oblasti webu. OWASP Foundation vznikla 21. 4. 2004 ve Spojených státech amerických, ale počátek projektu OWASP a první zprávy jsou na webových stránkách www.owasp.org dostupné již od 1. 12. 2001. Iniciátory a zakladateli tohoto projektu jsou Mark Curphey a Dennis Groves. Pod hlavičkou OWASP Foundation je dostupná celá řada projektů, jejichž cílem je zlepšit bezpečnost a zabezpečení webových aplikací. Tyto projekty se dají rozdělit na dokumentační a vývojářské. OWASP Foundation podporuje vznik lokálních poboček, které mají za úkol šířit osvětu v oblasti zabezpečení prostřednictvím seminářů konferencí a diskusí. V České republice jsou oficiálně založeny dvě pobočky, OWASP Czech Republic a OWASP Prague. OWASP Prague je ale v současné době neaktivní a nevyvíjí žádnou činnost. [2]

Příklady dokumentačních projektů

- OWASP Top Ten (Top Ten Most Critical Web Application Vulnerabilities) – žebříček 10 nejzávažnějších bezpečnostních hrozeb pro webové aplikace
- OWASP Appliacation Security Verification Standard (ASVS) – norma pro testování bezpečnostních prvků aplikací
- The Guide – podrobné pokyny pro zabezpečení webových aplikací
- Metrics – definuje metriky zabezpečení webových aplikací
- Legal – pomáhá prodávajícím i kupujícím sjednat odpovídající zabezpečení ve smlouvách
- Testing Guide – průvodce testováním zabezpečení webových aplikací
- ISO 17799 – podklady pro organizaci realizující ISO 17799
- AppSec FAQ – často kladené otázky

Příklady vývojářských projektů

- WebScarab – nástroj pro testování zranitelností webových aplikací
- WebGoat – děravá aplikace, na které je možné v bezpečném právním prostředí zkoušet bezpečnostní nedostatky
- Validation Filters – filtry
- DotNet – různé nástroje pro zabezpečení .NET aplikací

1.1 OWASP Top Ten

Tento žebříček 10 nejvážnějších bezpečnostních hrozeb je výsledkem shody odborníků na bezpečnost po celém světě. Projekt vede Dave Wichers a za dobu svého fungování uveřejnila OWASP Foundation již tři verze tohoto žebříčku. Výsledky a dokumenty projektu jsou vydány pod licencí Creative Commons Attribution Share Alike 3.0, která je umožňuje dále šířit při uvedení autora. Cílem projektu je zvýšení informovanosti webových tvůrců, manažerů a architektů o slabinách v zabezpečení. Poslední vydání OWASP Top Ten 2010 pochází z 19. 4. 2010. [3]

1.1.1 Hrozby webových stránek dle OWASP Top Ten 2010:

- A1: Injection – Injekce škodlivého kódu
- A2: Cross-Site Scripting (XSS) – Vkládání skriptů do stránek
- A3: Broken Authentication and Session Management – Narušená autentifikace a správa relace
- A4: Insecure Direct Object References – Nezabezpečený přímý odkaz na objekt
- A5: Cross-Site Request Forgery (CSRF) – Podvržení požadavku webové aplikace
- A6: Security Misconfiguration – Nebezpečná konfigurace aplikačních serverů
- A7: Insecure Cryptographic Storage – Nezabezpečené kryptografické úložiště
- A8: Failure to Restrict URL Access – Chybné zamezení přístupu ke konkrétním URL[4]
- A9: Insufficient Transport Layer Protection – Nezabezpečená síťová komunikace
- A10: Unvalidated Redirects and Forwards – Neplatná přesměrování a předávání

[4] Uniform Resource Locator – řetězec identifikující umístění dokumentů na Internetu

1.1.2 Změny hrozeb v průběhu let

V průběhu let se vlivem příchodu nových technologií závažnost jednotlivých hrozeb samozřejmě měnila. Tabulka 1 zachycuje změny v hodnocení závažnosti jednotlivých hrozeb v letech 2004, 2007 a 2010. Stupnice nebezpečnosti identifikuje hrozbu označenou A1 jako nejzávažnější, hrozbu s hodnocením A10 jako méně závažnou. Některá rizika byla od prvního vydání v roce 2004 odstraněna nebo definována přesněji a jejich název byl upraven.

2010	2007	2004
A1 – Injection	A1 – Cross Site Scripting (XSS)	A1 – Unvalidated Input
A2 – Cross Site Scripting (XSS)	A2 – Injection Flaws	A2– Broken Access Control
A3 – Broken Authentication and Session Management	A3 – Malicious File Execution	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object References	A4 – Insecure Direct Object Reference	A4 – Cross Site Scripting
A5 – Cross Site Request Forgery (CSRF)	A5 – Cross Site Request Forgery (CSRF)	A5 – Buffer Overflow
A6 – Security Misconfiguration	A6 – Information Leakage and Improper Error Handling	A6 – Injection Flaws
A7 – Insecure Cryptographic Storage	A7 – Broken Authentication and Session Management	A7 – Improper Error Handling
A8 – Failure to Restrict URL Access	A8 – Insecure Cryptographic Storage	A8 – Insecure Storage
A9 – Insufficient Transport Layer Protection	A9 – Insecure Communications	A9 – Application Denial of Service
A10 – Unvalidated Redirects and Forwards	A10 – Failure to Restrict URL Access	A10 – Insecure Configuration Management

Tabulka 1 – Porovnání změn rizik v posledních letech

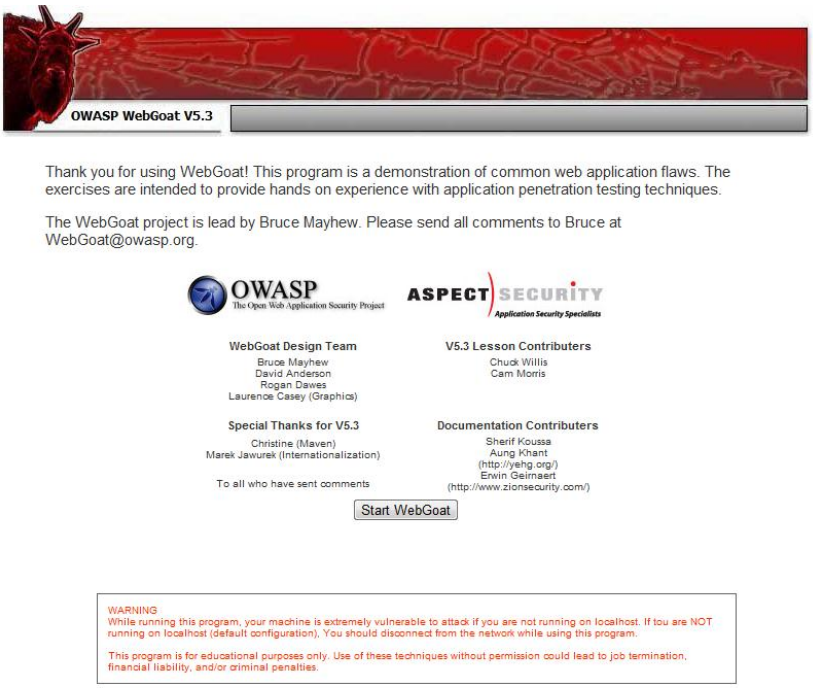
Tato práce využívá zejména informací právě z projektu OWASP Top Ten. Hrozby uvedené v OWASP Top Ten 2010 podrobně představuje a přináší řešení, jak se jim efektivně bránit.

1.2 Projekt WebGoat

Pro důkladné pochopení bezpečnostních zranitelností a poznání metod útoku je vhodné si je v praxi vyzkoušet. V případě pokusů na webových aplikacích třetích stran, ale může dojít ke konfliktu se zákonem, který pokusy o průnik do počítačových systémů postihuje. Trestní zákoník takové pokusy popisuje v § 230 „Neoprávněný přístup k počítačovému

systému a nosiči informací“ a v § 232 „Poškození záznamu v počítačovém systému a na nosiči informací a zásah do vybavení počítače z nedbalosti“. [4]

Projekt WebGoat nadace OWASP přináší výukovou aplikaci právě pro studium různých typů útoků. Jedná se o J2EE[5] webovou aplikaci, která obsahuje řadu bezpečnostních slabin. Poslední verze 5.3 pochází už z roku 2009, ale i tak lze její použití ke studijním účelům doporučit.



Obr. 1 – Úvodní stránka testovací aplikace WebGoat

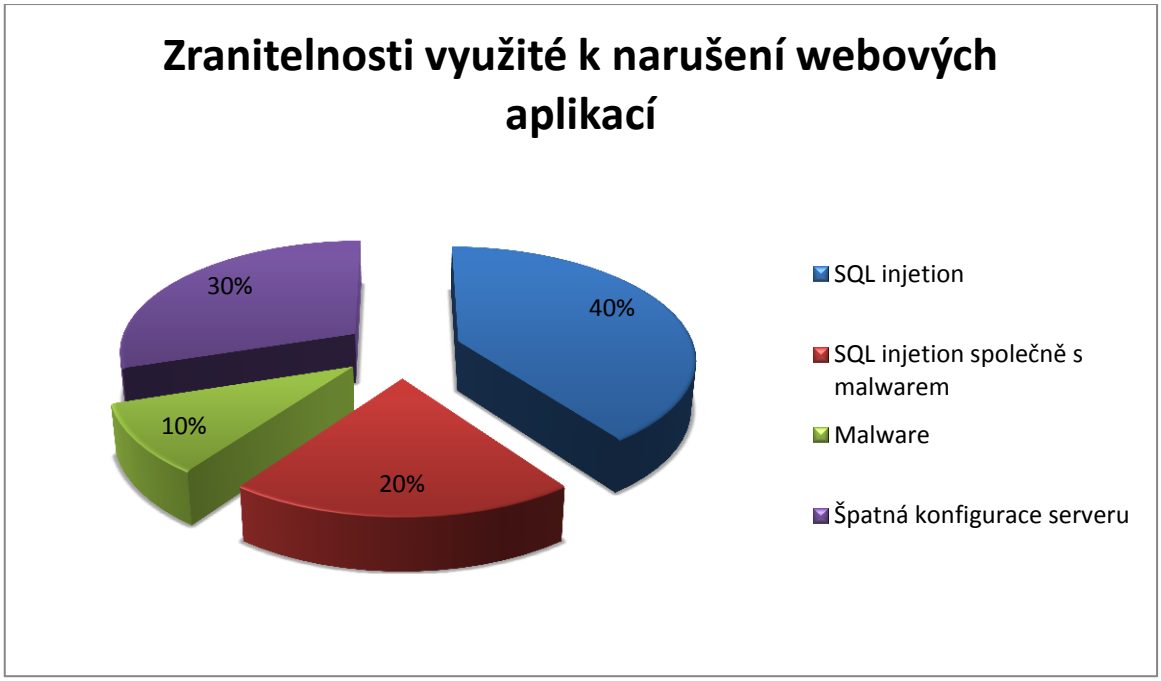
Zprovoznění aplikace je velice snadné a není nutné nic složitého nastavovat. Aplikace je distribuována spolu se serverem Tomcat. Po spuštění souboru *webgoat.bat* dojde k nastartování serveru Tomcat a poté již stačí do webového prohlížeče zadat URL „*http://localhost/webgoat/attack*“. Po zadání této adresy je zobrazena výzva k zadání uživatelského jména a hesla. Oba údaje jsou ve výchozí distribuci aplikace nastavena na „guest“. Po přihlášení do aplikace dojde k zobrazení základní obrazovky, jak ilustruje Obr. 1. Aplikace obsahuje nápovědu k použití i drobné příklady, které mají případně pomoci k odhalení zranitelnosti.

[5] Java Platform, Enterprise Edition – je součást platformy Java určená pro vývoj a provoz podnikových aplikací a informačních systémů

2 INJECTION

Pro uchování dat, která jsou prezentována prostřednictvím webových stránek a aplikací, se používají rozličné relační databáze jako např. Microsoft SQL server, MySQL, Oracle, PostgreSQL atd. Výstup, který webová stránka poskytuje, musí být nějakým způsobem konfigurovatelný. Data jsou vybírána s rozličnými podmínkami, omezeními a jsou různě řazena. Tyto informace je nutné předat do databázového dotazu. Útok SQL injection (SQL injektáž) zneužívá nedostatečné kontroly vstupních dat aplikace. Útočník tak může do aplikace vložit nebezpečný kód, jehož prostřednictvím může přistupovat neoprávněně k datům, upravovat položky v databázi nebo mazat tabulky databáze.

Jedná se v současnosti o nejrozšířenější metodu útoku na webové aplikace, jak ukazuje Graf 1.



Graf 1 - Zranitelnosti využité k narušení webových aplikací

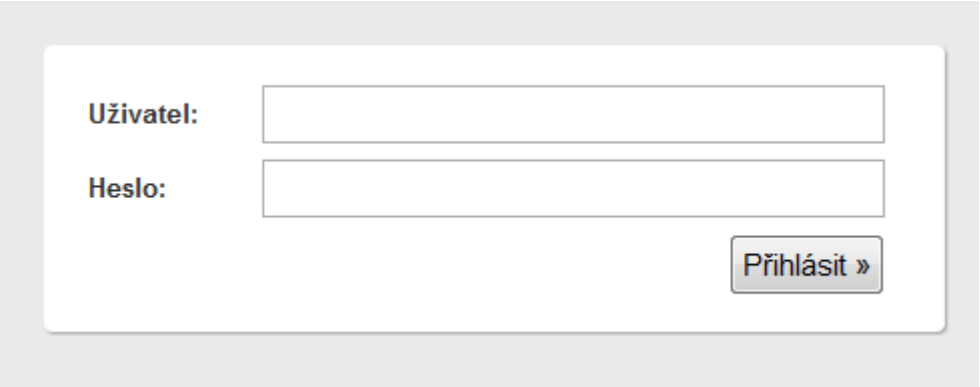
V České republice je známým zneužitím SQL injection napadení portálu libimseti.cz v roce 2008. Tento případ byl poměrně značně medializován, jelikož prostřednictvím tohoto útoku se útočníkům podařilo odcizit značné množství fotografií ze soukromých alb uživatelů [5]. Útoku SQL injection se neubránila ani společnost Sony. V roce 2011 bylo z jejích služeb Playstation Network, Qriocity a Sony Online Entertainment vykradeno téměř 100 milionů uživatelských účtů, včetně čísel kreditních karet. [6]

2.1 Možnosti útoku

Bezpečnostní riziko vzniká neobezřetnou manipulací se vstupními daty pro SQL dotaz a spoléháním se na to, že uživatel nebude zadávat neplatná data. Zadáním znaku apostrofu může útočník docílit změny kontextu SQL parseru[6] a přidávat tak vlastní SQL příkazy.

2.1.1 Podvodné přihlášení

Mnoho stránek obsahuje části, které jsou dostupné pouze vybraným uživatelům. Může jít o administrátorské části stránek pro přidávání článků a dalšího obsahu webu, nebo jen část pro registrované návštěvníky webu. Tito uživatelé mají k přístupu do uzavřených částí webu své uživatelské jméno a heslo.



Obr. 2 – Přihlašovací formulář

Uživatel zadá údaje do formuláře na Obr. 2. Webová stránka data z formuláře přijme a ověří v databázi uživatelů. Následující příklad skriptu v jazyce PHP ilustruje, jak by takové ověření mohlo vypadat.

```
$dotaz = "SELECT * FROM admin WHERE uzivatel = '".$_POST[uzivatel]."' AND
heslo = '".$_POST[heslo]."'";
```

Uvedený SQL dotaz vybere všechny sloupce tabulky *admin* a vrací řádky, které splní podmínku, že *uzivatel* a *heslo* se shodují s daty zadanými do přihlašovacího formuláře, která jsou obsažena v databázi uživatelů. Pokud uživatel zadá platná data, je přihlášen do systému.

[6] syntaktický analyzátor – provádí analýzu formálních prvků zadaných příkazů

V případě, že útočník zadá do pole „Uživatel“ řetězec: „*admin' --*“ dotaz se poskládá do následujícího tvaru:

```
SELECT * FROM admin WHERE uzivatel = 'admin' -- ' AND heslo = ''
```

V tomto případě se útočníkovi podařilo kombinací apostrofu a dvou pomlček vyřadit klíčovou část podmínky v SQL dotazu, která má zajistit porovnání s platným heslem. Podmínka za pomlčkami je chápána již jen jako komentář. SQL dotaz se tedy v podstatě zkrátí pouze na tvar:

```
SELECT * FROM admin WHERE uzivatel = 'admin'
```

Útočník buď přímo věděl, že se v systému nachází uživatel *admin* nebo tento fakt jen předpokládal, protože tento uživatel se vyskytuje v naprosté většině systémů. Navíc uživatel *admin* má v systému zpravidla největší oprávnění. Použitím apostrofu došlo v SQL dotazu k ukončení části řetězce, kde je očekávána hodnota pro zadání uživatelského jména.

Pro označení komentáře se v SQL používá několik znaků (posloupnosti znaků) v závislosti na konkrétní databázi. Nejčastější je již uvedený „--“ dále pak „#“, „//“ a pro více řádkové komentáře „/* */“ a „{ }“. Se všemi těmito potenciálně nebezpečnými konstrukcemi je třeba při návrhu aplikace počítat.

Ne vždy je možné použít tento postup využívající zneplatnění části podmínky označením za komentář. To ovšem neznamená, že se do systému nelze přihlásit bez znalosti hesla. Je však třeba použít trochu jiný postup.

Útočník do pole Uživatel zadá následující řetězec: „*admin' OR 'a' = 'b*“ a SQL dotaz, pozmění do následující podoby:

```
SELECT * FROM admin WHERE uzivatel = 'admin' OR 'a' = 'b' AND heslo = ''
```

Tento dotaz nevypadá na první pohled nijak problematicky. Podmínka dotazu zůstává celá v platnosti a každý z apostrofů je ukončen. Útočník v tomto případě využívá různé priority logických operátorů. Operátor *AND* má prioritu před operátorem *OR*. Při vyhodnocení podmínky je nejprve porovnán výraz *'a' = 'b' AND heslo = ''*. Tato část podmínky není splněna a vrací tedy hodnotu *FALSE*.

Po přepsání dotazu s již vyhodnocenou částí podmínky, vznikne následující dotaz:

```
SELECT * FROM admin WHERE uzivatel = 'admin' OR FALSE
```

Na tomto přepsaném dotazu je problém již zřejmý. Pro zbývající vyhodnocení operátoru *OR* stačí, aby byla pravdivá pouze jedna část výrazu. Což je v případě přítomnosti uživatele *admin* jako v předešlém příkladu splněno. Útočník se tak opět dokázal do systému přihlásit i bez znalosti hesla a bez použití komentáře v podmínce SQL dotazu.

V případě, že útočník nezná žádného uživatele systému, stále se může do systému přihlásit úpravou nebezpečného kódu. V tomto případě využije logických operátorů k vytvoření vždy platné podmínky. Místo jména uživatele stačí vložit řetězec: „*' OR 1=1 --*“ a tím dojde ke složení SQL dotazu:

```
SELECT * FROM admin WHERE uzivatel = '' OR 1=1 -- AND heslo = ''
```

Útočník pomocí výrazu „1=1“ docílil, že podmínka je vždy splněna. Nutnost shody hesla je opět vyřazena pomocí komentáře.

Další kombinace řetězců, kterých může útočník využít k průniku do aplikace: „ OR 1=1--“, „" or 1=1-- “, „" OR "a"="a ') “, „or ('a'='a '“, „or 'a'='a“

2.1.2 Získání dat z databáze

Ve výše uvedených případech je využito faktu, že útočník může zadat znak apostrofu k ukončení řetězce dat v SQL příkazu a za ním doplnit vlastní SQL příkazy. Může se tak do systému přihlásit jako libovolný uživatel. Útočníka ale často zajímají i data v jiných tabulkách. K získání těchto dat lze použít SQL příkaz *UNION*, který slouží ke sjednocení výsledků více dotazů *SELECT*.

Následující příklad ilustruje možný útok na stránku zobrazující různé články. Pomocí parametru *id* v URL adrese je vybrán konkrétní článek. Parametr je předán do skriptu, který vytvoří uvedený SQL dotaz.

URL adresa: *www.nezabezcena-stranka.cz/clanky.php?id=10*

SQL dotaz: `SELECT * FROM clanky WHERE id=10`

Jelikož vstupní proměnná *id* není nijak kontrolována, může útočník jejím prostřednictvím předat do dotazu další SQL příkazy. *UNION* připojí k původnímu příkazu *SELECT* další,

kterým si útočník vybírá data z *tajnaTabulka.* Jediným úskalím této metody je, že druhý příkaz *SELECT* musí vybírat stejný počet sloupců. V tomto případě má původní tabulka 8 sloupců. Tento fakt ale může útočník po pár pokusech zjistit. Obsah *tajnaTabulka* je pak vypsán místo jednotlivých sloupců tabulky *clanky.* [7]

> URL adresa: *www.nezabezcena-stranka.cz/clanky.php?id=10 UNION ALL SELECT 1,2,3,4,5,6,7,8 FROM tajnaTabulka*
>
> SQL dotaz: `SELECT * FROM clanky WHERE id=10 UNION ALL SELECT 1,2,3,4,5,6,7,8 FROM tajnaTabulka`

2.1.3 Ostatní typy útoku

K dalšímu narušení aplikace může útočník využít řadu dalších konstrukcí jazyka SQL. Zde již záleží na typu použité databáze a její verzi. Příklady funkcí používaných k narušení aplikace jsou následující:

- Použití funkce *LOAD_FILE* – umožní do SQL dotazu načíst externí soubor
- Použití funkce *INTO OUTFILE* – umožní zápis do souboru případně samotného skriptu
- Použití databáze INFORMATION_SCHEMA – databáze dostupná v MySQL, obsahuje informace o dostupných databázích na serveru
- Použití funkce *CHAR* – umožní maskování zadávaných dat jejich ASCII[7] hodnotou

2.2 Způsoby obrany

Vzhledem k tomu, že problém SQL injection je známý již řadu let, existuje také několik různých řešení, které tuto slabinu odstraňují. Programátor si tak může vybrat variantu, která je pro konkrétní aplikaci nejvhodnější.

- Ošetření vstupů escapováním[8]
- Kontrola datového typu
- Speciální databázová vrstva

---

[7] American Standard Code for Information Interchange – definuje kódovou tabulku se znaky abecedy a dalšími znaky

[8] doslovně lze přeložit jako „unikáním“, tento výraz se ale nepoužívá.

- Použití frameworku[9]
- Konfigurační direktiva PHP
- Omezení výstupu chybových hlášení

2.2.1 Ošetření vstupů escapováním

Jde o odstranění znaků, které v daném kontextu mají speciální význam a používají se např. k uvození textových řetězců. Typicky jde právě o zmíněné apostrofy, uvozovky nebo lomítka. Použitím tzv. escape znaků sdělíme parseru informaci, že se nejedná o změnu kontextu příkazu, ale o zadávanou hodnotu.

V PHP jsou pro doplnění escape znaků a bezpečnou práci s MySQL databází k dispozici funkce *mysql_escape_string* a *mysql_real_escape_string.* Jejich funkce se liší pouze ve faktu, že *mysql_real_escape_string* bere v potaz aktuálně nastavenou znakovou sadu. Od verze PHP 5.3.0 je ovšem funkce *mysql_escape_string* označena jako zastaralá a její použití se **nedoporučuje**. Náhradou je druhá zmíněná funkce *mysql_real_escape_string.*

Funkce *mysql_real_escape_string* má následující parametry:

```
string mysql_real_escape_string(string $neosetreny_retezec [,resource
$odkaz_spojeni = NULL])
```

Použití:

```
<?php
$uzivatel = mysql_real_escape_string($_POST[uzivatel]);
$heslo = mysql_real_escape_string($_POST[heslo]);
$dotaz = "SELECT * FROM admin WHERE uzivatel = '".$uzivatel."' AND heslo
= '".$heslo."'";
?>
```

SQL dotaz v případě pokusu o napadení řetězcem „*admin' --*“ vypadá takto:

```
SELECT * FROM admin WHERE uzivatel = 'admin\' --' AND heslo = ''
```

V případě použití databáze PostgreSQL je k escapování dat k dispozici funkce *pg_escape_string*. Další již obecnou funkcí bez zaměření na úpravu dat pro konkrétní

[9] sada knihoven, funkcí a dalších podpůrných nástrojů pro snadnější a efektivnější vývoj aplikací

databázi je *addslahes*. Funkce doplní zpětná lomítka před znaky: apostrof, uvozovka, zpětné lomítko. [8]

2.2.2 Kontrola datového typu

Pokud webová aplikace předpokládá, že vstupní parametry budou vždy numerické, je nutné také tuto skutečnost ověřit a předejít tak útoku s využitím klausule *UNION*.

```
$id=(int)$_GET[id];
$dotaz="SELECT * FROM clanky WHERE id='$id'";
```

Proměnná *id* je před vložením do databázového dotazu přetypována pomocí funkce *(int)*. Tím je zajištěno, že *id* bude vždy číslo a nebude možné vložit další příkazy jazyka SQL.

2.2.3 Speciální databázová vrstva

Použití databázové vrstvy je jakýmsi mezikrokem mezi používáním přímých databázových funkcí jako např. *mysql_query* a použitím celého frameworku. Databázová vrstva vytváří rozhraní mezi aplikační logikou aplikace a samotnou databází. Programátorovi tak ulehčuje práci s databází, stará se o ošetření dat escape znaky a umožňuje snadnější přenositelnost mezi různými databázemi. Aplikace je napojena na databázovou vrstvu a nikoliv na konkrétní funkce spojené s určitou databází. Běžně dostupné databázové vrstvy:

- Dibi – vyvinuto v Čechách, velmi aktivní česká komunita
- ADOdb
- PDO - PHP Data Objects, součást PHP

2.2.4 Použití frameworku

Použití frameworku při vývoji aplikace není ochranou v pravém slova smyslu. Samotný framework pochopitelně může jako takový obsahovat bezpečnostní chyby. Většinou se nejedná o produkt jednoho vývojáře, ale je zpravidla vyvíjen celou komunitou programátorů. Tento přístup do značné míry minimalizuje chyby a bezpečnostní rizika v kódu. Použití frameworku zcela nevylučuje opomenutí bezpečnostních zásad při vývoji samotné aplikace, ale ve většině případů toto riziko velmi významně minimalizuje. Vývojář se tak může plně soustředit na vývoj požadované funkcionality aplikace a problémy s ošetřením vstupu dat přenechat na frameworku. Framework se většinou automaticky již postará o doplnění dat escape znaky apod.

2.2.4.1 PHP framework Nette

Velmi oblíbeným frameworkem pro PHP je Nette. Jedná se o framework s českými kořeny, jehož autorem je David Grudl. V současnosti je framework již vyvíjen celou komunitou vývojářů sdružujících se na webu frameworku Nette (http://nette.org). Hlavní verze frameworku je 2.0 a byla publikována v roce 2012. Jedná se o framework s otevřeným kódem, tzv. open-source. Jeho použití je umožněno pod licencí New BSD[10] nebo GNU[11]. [9]

2.2.4.2 PHP framework Zend

Dalším velmi známým a oblíbeným PHP framworkem je Zend. Framework je také vyvíjen jako open-source a je distribuován s licencí New BSD.

Pro práci s databází v Zend framworku slouží modul *Zend_Db_Adapter*. Celá aplikace by měla komunikovat s databází prostřednictvím tohoto modulu. Pro předcházení riziku SQL injection obsahuje tento modul metodu *quote(),* která přidá do vstupního řetězce escape znaky. [10]

2.2.5 Konfigurační direktiva PHP

Konfigurační direktiva jazyka PHP není sice určena primárně jako ochrana před útokem SQL injection, ale její funkčnost může napadení zabránit. Funkce způsobí, že veškerá data dostupná ze super globálních polí $_POST a $_GET jsou doplněna o zpětné lomítko před všechny apostrofy. To v některých případech ulehčuje situaci s ošetřováním apostrofů při vstupu dat do SQL dotazu, ale zároveň toto řešení přináší také problémy v okamžiku, kdy tato data mají být zobrazena na výstupu. V tomto případě je zobrazení zpětných lomítek nežádoucí a musí dojít pomocí funkce *stripslashes()* k jejich odstranění. To lze již aplikovat na konkrétní výstupní řetězec, integrita databáze je tedy ochráněna.

Vzhledem k tomu, že se jedná o konfigurační direktivu přímo skriptovacího jazyka PHP, nemusí být vždy možné zajistit její potřebné nastavení. Zda je funkce v daném prostředí

---

[10] Berkeley Software Distribution Licence – umožňuje volné šíření licencovaného obsahu, přičemž vyžaduje pouze uvedení autora a informace o licenci, spolu s upozorněním na zřeknutí se odpovědnosti za dílo.
[11] GNU General Public License – licence pro svobodný software, vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí

aktivní, můžeme zjistit před zahájením vývoje pomocí funkce *phpinfo()*, která poskytuje výstup jako na Obr. 3.

magic_quotes_gpc	Off	Off
magic_quotes_runtime	Off	Off
magic_quotes_sybase	Off	Off

Obr. 3 – Výstup funkce *phpinfo()*

Ve skriptu aplikace pak lze zjistit stav direktivy pomocí funkce *get_magic_quotes_gpc()*. Od verze PHP 5.3 je tato funkce standardně deaktivovaná. Je označená jako zastaralá a od verze PHP 6 bude vyřazena. Její použití se v současnosti nedoporučuje. [11]

2.2.6 Omezení výstupu chybových hlášení

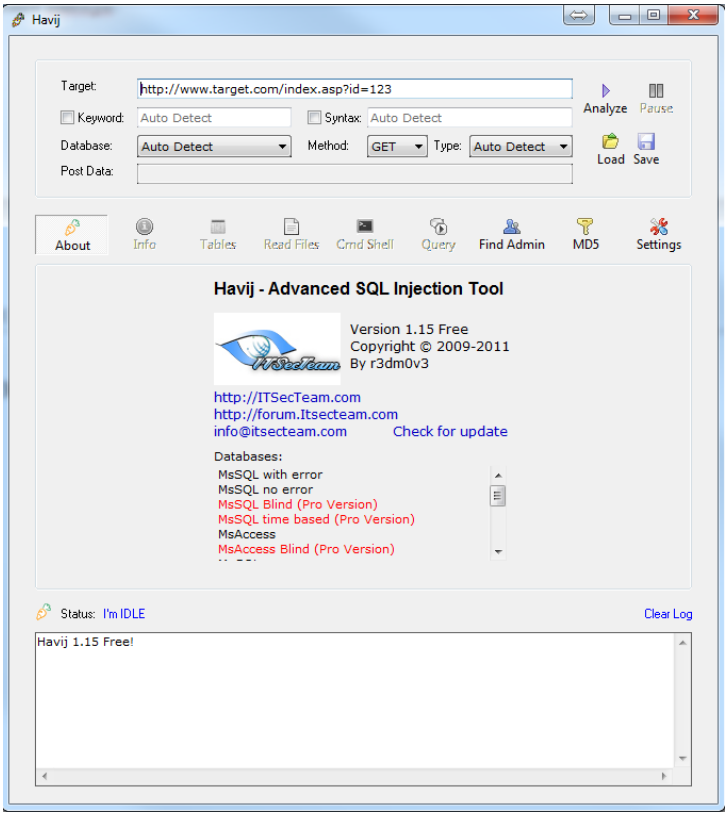
Informace obsažené v chybových zprávách mohou útočníci využít ke zpřesnění svého útoku. Je tedy nutné, aby hlášení neobsahovala žádné detailní informace o verzi databáze, názvech tabulek apod. Hlášení na produkčním serveru by měla být pouze obecného informačního charakteru. Pro běžného návštěvníka navíc nemají podrobné hlášení žádný význam. Podrobné informace o chybě je vhodné ukládat do interního logu a zaslat hlášení o chybě administrátorovi webové aplikace.

2.3 Nástroje pro testování

Pro testování náchylnosti webových aplikací k útoku typu SQL injection lze použít řadu nástrojů. Může se jednat o desktopové aplikace nebo i řadu online scannerů.

2.3.1 Havij

Havij je nástroj k odhalování zranitelností SQL Injection. Pomocí automatických penetračních testů webových stránek dokáže velmi rychle prověřit nejčastější konstrukce SQL Injection útoku, které byly popsány výše. Hlavní okno programu je velmi intuitivní, jak je vidět na Obr. 4.



Obr. 4 – Hlavní okno programu Havij

Do programu stačí zadat URL testované aplikace spolu s parametrem, který má být použit pro provedení injektáže kódu. Stiskem tlačítka „Analyze“ dojde k zahájení testování. Jeho průběh je vidět ve spodním okně. Pokud je stránka zranitelná, dojde ke zpřístupnění dalších prvků programu: „Info“ a „Tables“. V „Tables“ je k dispozici přehled databázových tabulek, které program útokem odhalil. Je tak možné z tabulek získat konkrétní data.

Program disponuje i některými funkcemi, které jsou pro testování zranitelnosti již nadbytečné, jako na např. možnost prolamování MD5 hashů pomocí předpočítaných databází. Může tak také sloužit nejen jako prostředek k testování, ale i jako jednoduchý nástroj k průniku do webových aplikací, přičemž útočník nemusí mít téměř žádné podrobnější znalosti o využívané zranitelnosti. Takové použití je samozřejmě v rozporu se zákonem.

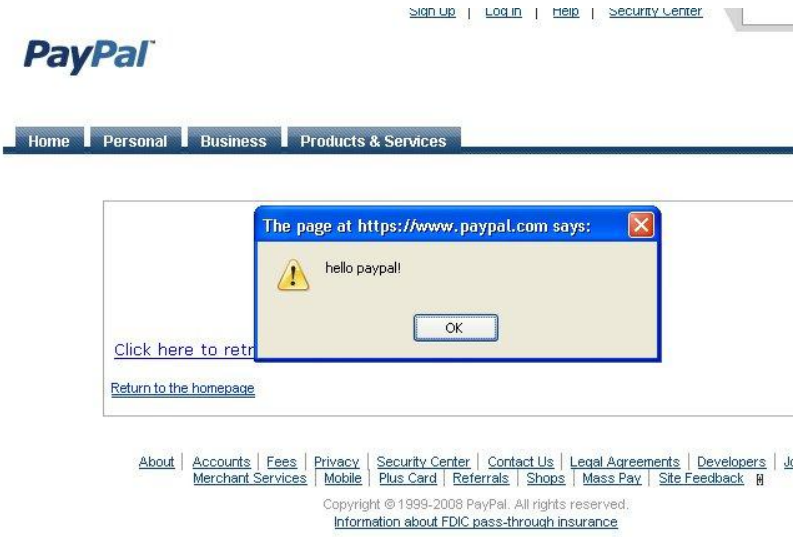
3 CROSS-SITE SCRIPTING (XSS)

Útok typu Cross-site scripting (skriptování napříč weby) využívá podobně jako SQL injection nedostatečné kontroly vstupu dat do webové aplikace a následně jejich nekontrolovanou reprezentaci. Běžně se označuje jako XSS. To je z důvodu, že zkratka, která by vznikla z počátečních písmen názvu útoku, by mohla být zaměňována s kaskádovými styly, které se označují zkratkou CSS[12]. Jako vstup je zadán řetězec znaků obsahující HTML značky, případně je využito jazyka JavaScript. K vložení skriptu ve většině případů dochází skrze formulář nebo parametr v URL adrese. Pokud je pak tento kód zobrazen návštěvníkovi stránky, může jeho prostřednictvím dojít k úpravě obsahu stránky, k přesměrování na stránky jiné nebo k odcizení cookies[13] uživatele. Nezáleží přitom na technologii resp. jazyce, ve kterém je stránka napsána, neboť útok je platformě nezávislý. Pokud nedojde ke korektnímu ošetření vstupních dat, jsou tímto útokem zranitelné weby v PHP, ASP.NET, Javě a další.

Útoky toho typu se nevyhýbají ani webovým stránkám významných společností a vládních institucí. Z poslední doby je znám útok na web španělského předsednictví EU, kde útočníci prostřednictví XSS útoku upravili fotografie politiků. Útoku se nevyhnula ani společnost PayPal zabývající se platbami na internetu viz Obr. 5. Její uživatelé byli přesměrováni na podvodný web, který se využitím prvků sociálního inženýrství[14] snažil uživatele donutit zadat přihlašovací údaje ke službě PayPal a údaje o kreditní kartě [12]. Hrozbu XSS v sobě obsahovala i populární sociální síť Twitter.

Za potencionálně nebezpečný lze obecně považovat **jakýkoliv vstup dat** do webové aplikace, zejména pak superglobální pole $_GET, $_POST, $_REQUEST, $_COOKIE, $_SERVER. Prostřednictvím těchto polí může útočník do aplikace dostat potencionálně nebezpečný kód, jak je uvedeno na příkladech níže. [13]

[12] Cascading Style Sheets – jazyk pro definování vzhledu webových stránek
[13] v předkladu sušenky nebo oplatky, jedná se o data webové stránky, které jsou uložena lokálně u uživatele. Používají se k zapamatování přihlášeného uživatele, obsahu nákupního košíku apod.
[14] označuje pokusy o manipulaci návštěvníků webu vedoucí k provedení požadované akce – např. zadání přístupových údajů, čísel kreditních karet apod.



Obr. 5 – Ukázka zranitelnosti XSS na webových stránkách společnosti PayPal

3.1 Reflected Cross Site Scripting

Reflected Cross Site Scripting, také označován jako Non-Persistent[15] XSS, je nejběžnějším typem XSS útoku, zároveň nejméně nebezpečným. Dochází k němu u dat, které webová stránka neukládá, ale rovnou je prezentuje. To jsou většinou údaje zadané do vyhledávacích polí, průvodců pro výběr produktů, nebo jsou předány jako parametr v adrese URL. V případě úspěšného vložení škodlivého kódu prostřednictvím těchto polí dojde k vykonání kódu pouze v prohlížeči návštěvníka. Tento typ útoku může způsobit problémy zejména ve spojení s technikami sociálního inženýrství. Útočník donutí oběť, aby stránky navštívila pomocí odkazu, který do stránky vloží škodlivý kód. Útočník může vložit do vyhledávacího pole na stránce řetězec:

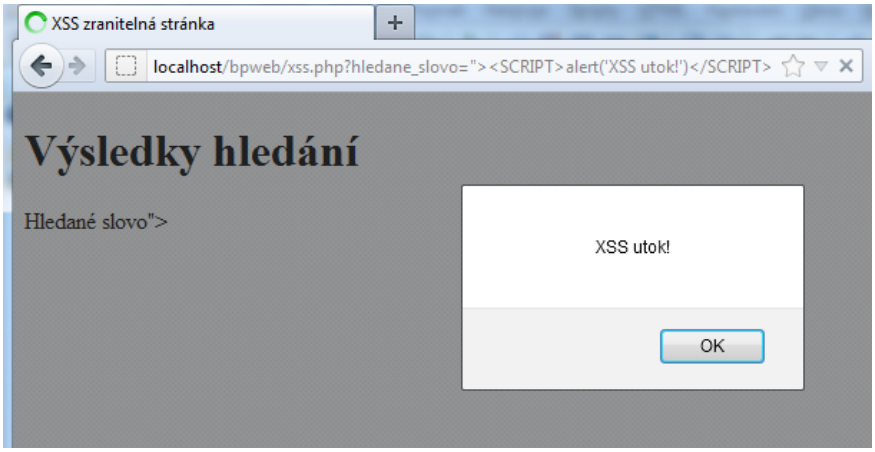
```
"><SCRIPT>alert('XSS utok!')</SCRIPT>
```

Stránka vyhledávání pak zadanou hodnotu zpracuje a zobrazí pomocí kódu:

```
<h1>Výsledky hledání</h1>
<p>Hledané slovo: <?php echo $_GET['hledane_slovo']?></p>
```

[15] neperzistentní, dočasný

PHP doplní místo proměnné *hledane_slovo* skript, který útočník zadal. Tento skript vyvolá na stránce zranitelné na XSS vyskakovací okno s hláškou „XSS útok“. Úvodní dva znaky „">“ ukončí HTML značku, do které se standardně vypisuje obsah vyhledávacího pole. Následně již je vložen JavaScript pro vyvolání vyskakovacího okna. Výsledek útoku je vidět na Obr. 6.

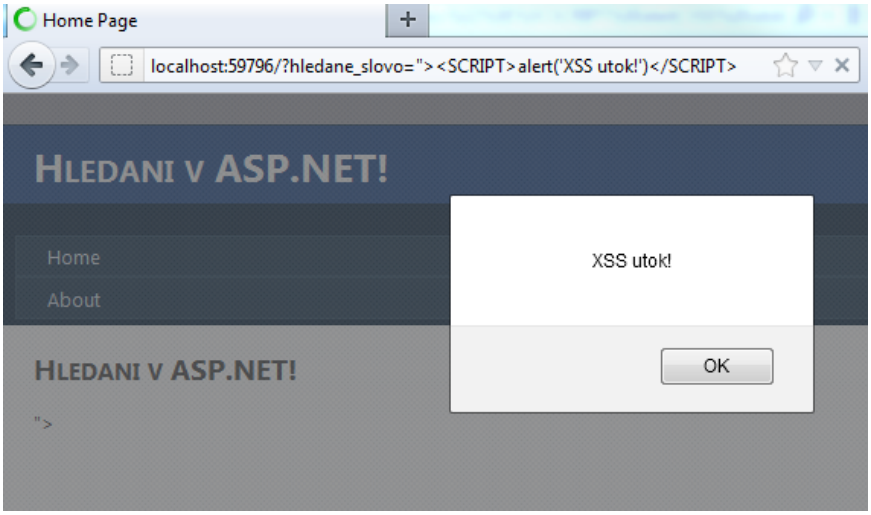


Obr. 6 – Příklad XSS útoku na stránku v PHP

Stejně tak lze postupovat i v ASP.NET

```
<p>
  <%=Request.QueryString["hledane_slovo"]%>
</p>
```

Uvedený příklad opět bez ošetření použije data zadaná uživatelem k vypsání proměnné *hledane_slovo* předané metodou GET. Výsledkem je opět provedení škodlivého kódu v prohlížeči návštěvníka viz Obr. 7.



Obr. 7 – Příklad XSS útoku na stránku v ASP.NET

3.2 Stored Cross Site Scripting

Stored Cross Site Scripting je v podstatě opakem neperzistentnímu XSS. Bývá označován jako Persistent[16] XSS. Jeho výskyt je běžný v aplikacích, kde může uživatel vkládat do aplikace data, která jsou ukládána a dále prezentována. Jako příklad lze uvést různá diskusní fóra, návštěvní knihy apod. Tento typ útoku je již podstatně nebezpečnější vzhledem k faktu, že postižen není pouze jeden návštěvník webu. Nebezpečný kód vložený např. do diskusního příspěvku je vykonán na počítačích všech návštěvníků daného fóra.

```
<div>Diskusní příspěvek: <?php echo $text_prispevku_z_databaze?></div>
```

Uvedený příklad zobrazí text diskusního příspěvku z databáze. Pokud text příspěvku bude obsahovat injektovaný nebezpečný kód, dojde v tomto okamžiku ke XSS útoku, jelikož vypisovaný text není nijak ošetřován. I zde lze aplikovat nebezpečné kódy představené v předešlém příkladu.

3.3 DOM based Cross Site Scripting

Je obdobou Reflected XSS s tím rozdílem, že injektovaný kód se stává součástí původního JavaScriptu. Tento útok lze tedy použít i na statických stránkách využívajících JavaScript. Škodlivý kód je opět do skriptu stránky injektován prostřednictví parametru v adrese webové stránky.

```
<SCRIPT>
 var pos=document.URL.indexOf("hledat=")+7;
 document.write("Hledali jste:
"+document.URL.substring(pos,document.URL.length));
</SCRIPT>
```

Uvedený JavaScriptový kód standardně vypisuje hledanou frázi, které je do stránky předána prostřednictvím parametru „*hledat*“. Útočník může tento parametr využít k vložení nebezpečného kódu prostou úpravou URL.

```
http://nebezpecna-stranka.cz/stranka.html?hledat=<SCRIPT>alert('XSS
utok!')</SCRIPT>
```

[16] persistentní, trvalý

Útočník tímto odkazem docílil XSS útoku, který na stránce otevře pop-up okno s textem „XSS utok“. Při navštívení webu přes neupravený odkaz se bude stránka chovat korektně. [14]

3.4 Testování náchylnosti ke XSS

Prvním krokem testování zranitelnosti webové stránky vůči útoku XSS může být vyzkoušení výše uvedených příkladů. Další možností je využití online scanerů. Jedná se o webové aplikace, které se na základě popsaných metod snaží o průnik do webové stránky. Základní přehled online scanerů XSS zranitelnosti:

- DOM XSS Scanner - http://www.domxssscanner.com/
- Cross Site Scripting Scanner - http://xss-scanner.com/

Uvedené aplikace jsou zdarma a lze je použít bez nutnosti instalace. Na trhu existuje dále řada jak bezplatných, tak komerčních desktopových aplikací určených k detekci XSS.

3.5 Ochrana před XSS

Podobně jako v případě předchozí zranitelnosti i v případě útoku XSS, ochrana spočívá v dostatečném ošetření vstupních dat před jejich zpracováním, což je zpravidla výstup do kódu webové aplikace. Ochrana spočívá v ošetření znaků, které by mohly změnit kontext zpracování dat jako např. HTML tagy, CSS tagy apod.

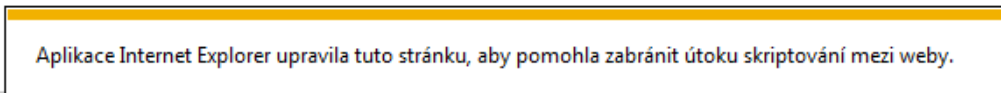
Pravidla pro předcházení XSS útoku [14]:

1. Nevkládat neověřená data do aplikace
2. Provést escapování HTML tagů před vložením do HTML kódu aplikace
3. Provést escapování atributů před vložením do HTML kódu aplikace
4. Provést escapování neověřeného JavaSript kódu
5. Provést escapování neověřených dat před vložením do URL

3.5.1 Ochrana z pohledu uživatele

I samotný návštěvník webu zranitelného útokem XSS může podniknout několik kroků ke své ochraně před tímto útokem. V první řadě by měl být samozřejmostí aktualizovaný operační systém včetně posledních bezpečnostních záplat. Stejně tak používaný internetový prohlížeč by měl být v poslední stabilní verzi a aktualizovaný. Internetový

prohlížeč Internet Explorer od verze 8 disponuje vestavěným XSS filtrem, jehož cílem je XSS útoky znemožnit. Tímto filtrem disponuje i prohlížeč Chrome.



Obr. 8 – Informační zpráva prohlížeče při zabránění XSS útoku

Uživatelé by se také měli vyvarovat otvírání webových stránek pomocí odkazů z nedůvěryhodných zdrojů. Uživatel také může v nastavení webového prohlížeče zakázat spouštění veškerého aktivního obsahu webových stránek. Tento krok přináší uživateli sice zabezpečení před možným útokem, ale značně snižuje uživatelský komfort práce s webovými stránkami. Špatně navržené weby nemusí bez podpory aktivního skriptování na straně klienta vůbec fungovat.

3.5.2 Ochrana z pohledu programátora

V závislosti na použité platformě má programátor k dispozici funkce k ošetření výstupních dat. Základní přehled dostupných funkcí přináší Tabulka 2.

Platforma	Funkce
PHP	Htmlspecialchars, strip_tags
ASP.NET	AntiXssLibrary, HtmlEncode, UrlEncode, XmlEncode, XmlAttributeEncode

Tabulka 2 – Přehled funkcí k ochraně před XSS

Veškerý vstup dat do aplikace by měl být validován za použití **seznamu povolených hodnot** tzv. whitelistu. Použití seznamu zakázaných hodnot (blacklistu) není vhodné zejména z důvodů, že nebezpečný kód lze různými způsoby maskovat a seznam tak prakticky nikdy nebude kompletní.

Použití níže uvedených funkcí je předpokládáno až na **výstupu dat** z aplikace. Funkce modifikující zdrojová data by neměly být používány např. při ukládání dat do databáze. Tímto postupem by se v aplikaci vytvořila závislost mezi datovou a aplikační vrstvou, která může způsobit problémy při další interpretaci dat. [13]

3.5.2.1 Ochrana v PHP

Pomocí funkce *strip_tags* je možné z řetězce odstranit všechny HTML a PHP tagy. Pomocí druhého nepovinného parametru lze definovat tagy, které mají zůstat zachovány. Funkce odstraní i HTML a PHP komentáře, přičemž toto chování nelze změnit.

```
string strip_tags(string $nebezpecnyRetezec [, string $povoleneTagy ]);
```

Použití:

```
$vstup='"><SCRIPT>alert(\'XSS utok!\')</SCRIPT>';
echo strip_tags($vstup);
```

Výstupem výše uvedeného kódu je následující řetězec: „">alert('XSS utok!')“. Výstupní řetězec již neobsahuje žádnou HTML značku a je tedy pro stránku neškodný.

Další možností je využití funkce *htmlspecialchars*, která provede konverzi speciálních znaků na HTML entity.

```
string htmlspecialchars(string $nebezpecnyRetezec [, int $flags =
ENT_COMPAT | ENT_HTML401 [, string $kodovani = 'UTF-8' [, bool
$double_encode = true ]]] )
```

Použití:

```
$vstup='"><SCRIPT>alert(\'XSS utok!\')</SCRIPT>';
echo htmlspecialchars($vstup);
```

Uvedený kód vypíše potencionálně nebezpečný vstup, který byl popsán výše. Rozdíl je ve zdrojovém kódu. Znaky „<“ a „>“ jsou nahrazeny HTML entitami a text mezi nimi není zpracován jako HTML značka, ale jen jako obyčejný text.

HTML zdrojový kód:

```
&quot;&gt;&lt;SCRIPT&gt;alert('XSS utok!')&lt;/SCRIPT&gt;
```

V případě nutnosti zpětného převodu je k dispozici funkce *htmlspecialchars_decode*. Tabulka 3 uvádí základní přehled, jak lze vybrané znaky zakódovat jako HTML entity.

Znak	Zakódování
<	< nebo <
>	> nebo >
&	& nebo &
"	" nebo "
'	' nebo '
(	&#40;
)	&#41;
#	&#35;
%	&#37;
;	&#59;
+	&#43;
-	&#45;

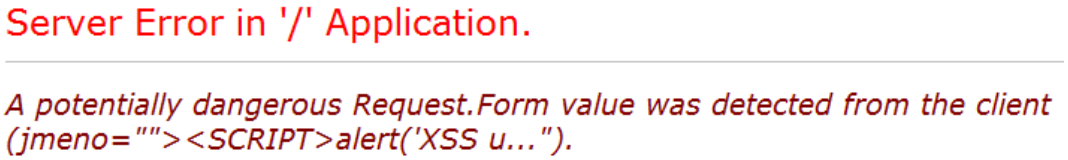
Tabulka 3 – Vyjádření vybraných znaků HTML entitami

3.5.2.2 Ochrana v ASP.NET

V ASP.NET lze jednoduše útoku předcházet nastavením vlastnosti ValidateRequest, jak ukazuje příklad.

```
<%@ Page ValidateRequest="true"%>
```

Všechna data z potencionálně nebezpečných vstupů jsou validována a v případě detekce XSS útoku je generována chyba, jak ilustruje Obr. 9.



Obr. 9 – Chyba pří detekování XSS útoku

Zapnutí kontroly požadavků pro všechny stránky lze učinit i prostřednictvím konfiguračního souboru *Web.config*. Do souboru stačí přidat následující nastavení [15]:

```
<system.web>
  <pages buffer="true" validateRequest="true" />
</system.web>
```

3.5.2.3 Použití frameworku

Stejně jako u rizika SQL injection je možné k tvorbě aplikace odolné vůči XSS využít frameworku.

Nette

Nette resp. jeho šablonovací systém Latte, ošetřuje veškeré potencionálně nebezpečné znaky zcela automaticky. To odstraňuje možnost opomenutí. Latte disponuje unikátní technologií Context-Aware Escaping, která rozezná, ve které části dokumentu se makro nachází a podle toho zvolí správné escapování.

Následující kód vyvolá varovné okno a nastaví titulek stránky, do kterého vypíše text z proměnné *$zprava*.

```
<p onclick="alert({$zprava})">{$zprava}</p>

<script>
document.title = {$zprava};
</script>
```

Pokud do proměnné nastavíme text „Potencionálně nebezpečný text & 8 1/2“ framework ho do kódu upraví takto:

```
<p onclick="alert(&quot; Potencionálně nebezpečný text &amp; 8
1\/2&quot;)"> Potencionálně nebezpečný text &amp; 8 1/2</p>

<script>var movies = " Potencionálně nebezpečný text & 8 1\/2";</script>
```

„Díky Context-Aware Escaping je šablona jednoduchá a aplikace přitom bude perfektně zabezpečená proti Cross Site Scripting. Dokonce je možné zcela nativně používat PHP proměnné uvnitř JavaScriptu.“ [16]

Zend

Aktuálně Zend framework sice nabízí základní ochranu před XSS útokem, ale bohužel ne zcela spolehlivou. Z tohoto důvodu jej **nelze doporučit** jako nástroj sloužící k ochraně před XSS. Kontrolu výstupu je nutné volat manuálně, jak ukazuje následující kód.

```
echo $this->escape($this->uzivatelskyVstup);
```

Toto řešení s sebou samozřejmě přináší riziko opomenutí funkci před výstupem dat zavolat. Navíc ošetření dat není zcela spolehlivé, jak ilustruje následující kód:

```
<span title='<?php echo $this->escape($this->uzivatelskyVstup); ?>'>Test</span>
```

Při vložení „*' onmouseover='alert(/XSS/)*“ dojde ke XSS útoku. [17]

4 NARUŠENÁ AUTENTIFIKACE A SPRÁVA RELACE

Většina moderních webových aplikací potřebuje pro svou správnou funkčnost jednoznačně identifikovat konkrétního uživatele, který ke stránce přistupuje. Aplikace pak přizpůsobuje obsah, který návštěvníkovi poskytuje. Uživatelé mají v aplikacích vytvořeny své uživatelské účty, pomocí nichž se přihlašují. Uživatelé si tak podle svých preferencí mohou nastavit chování webové aplikace. Všechny webové aplikace, ale pracují s HTTP[17] protokolem resp. s jeho šifrovanou variantou HTTPS[18]. Protokol HTTP je, ale navržen jako bezstavový. Vzdálený server vygeneruje svou odpověď na základě požadavku webového prohlížeče návštěvníka. Odpověď serveru je zpracována internetovým prohlížečem a je zobrazena konkrétní webová stránka. Jakákoliv další uživatelská akce vyvolá další požadavek. Relace neboli *sessions* slouží jako identifikace pro webový server, že se nejedná o nezávislé požadavky, ale že jde např. o požadavek již přihlášeného návštěvníka. Webový server si do souboru uloží informace o přihlášeném uživateli a s webovým prohlížečem si navzájem vyměňují session ID dále jen SID. V případě uhodnutí nebo odcizení SID může útočník manipulovat se session uživatele. Je proto velmi důležité tento identifikátor chránit před odcizením.

4.1 Možné problémy

Při práci se session je nutné zkontrolovat následující možné problémy:

- Identifikátor relace je dostupný v URL adrese jako parametr
  eshop.cz/prodej/nakupnikosik;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2
  Uživatel v tomto případě může rozesláním odkazu svůj účet kompromitovat, jelikož při přistoupení přes tento odkaz bude moci kdokoliv dokončit objednávku místo uživatele.
- Relace se neukončí po uzavření prohlížeče
  Uživatel se v internetové kavárně přihlásí ke svému účtu v aplikaci, ale nepoužije funkci odhlásit, případně tato funkce v aplikaci zcela chybí a uživatel pouze zavře

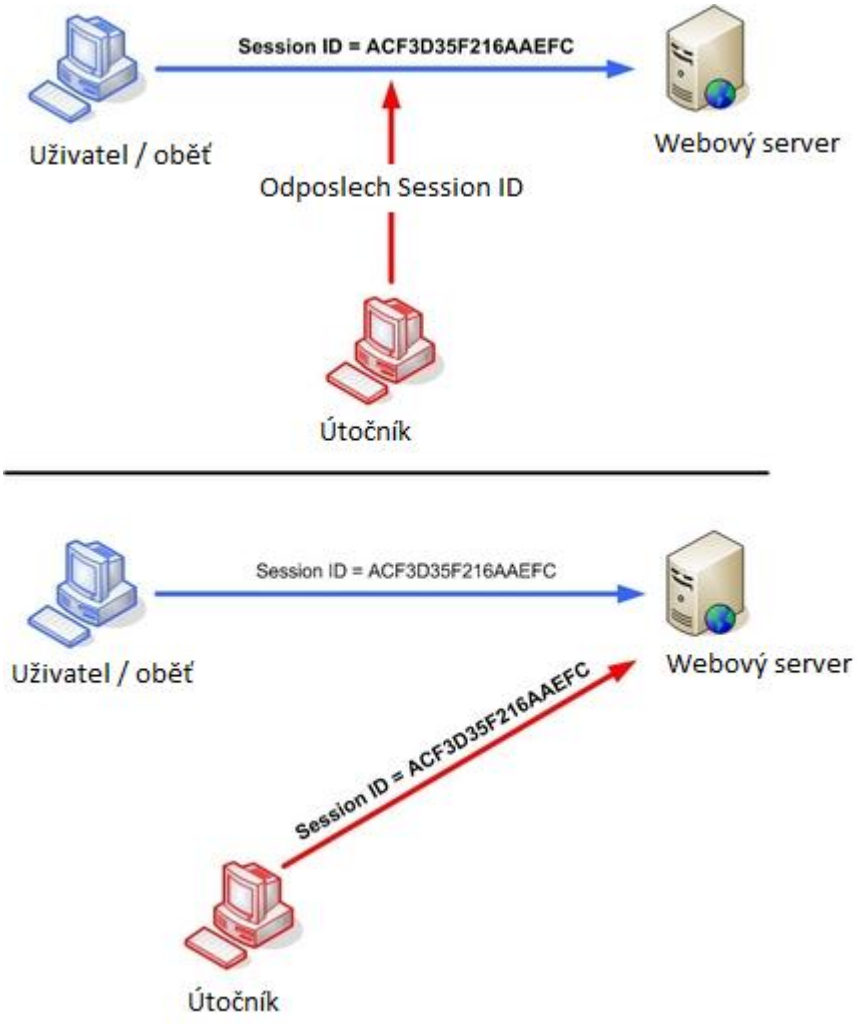
[17] HyperText Transfer Protocol – protokol určený k výměně dokumentů HTML.
[18] HyperText Transfer Protocol Secure – jedná se o nadstavbu původního HTTP protokolu. Umožnuje data pomocí http přenášet zašifrovaná a tím předcházet odposlechu citlivých údajů.

internetový prohlížeč. Při otevření prohlížeče dalším návštěvníkem kavárny, má tento uživatel plný přístup k dříve navštívené webové aplikaci.

- Identifikátor je odcizen např. XSS útokem nebo jiným způsobem
  Webová aplikace je náchylná na některé z uvedených typů útoku XSS, jehož prostřednictvím dojde k odcizení SID.
- Identifikátor relace lze uhádnout

  Jako identifikátor je použito uživatelské jméno, IP adresa nebo jiná informace, kterou může útočník odhadnout.

Následující Obr. 10 ilustruje útok označovaný jako Session Hijacking. Tento výraz označuje právě odcizení existujícího SID oprávněnému uživateli aplikace.



Obr. 10 – Schéma Session Hijacking

4.2 Ochrana před narušením autentifikace a správy relace

V první řadě je nutné zabránit vystavení SID do URL. V nastavení PHP je proto určená direktiva *session.use_trans_sid*. Nastavení se provádí v konfiguračním souboru *php.ini* a v aplikaci ji lze přenastavit pomocí souboru *.htaccess* (pokud je povoleno přepisování nastavení). Hodnota direktivy pro zamezení vystavení SID do URL je "0". Negativem tohoto řešení je fakt, že uživatelům, kteří mají zakázané cookies, přestane aplikace fungovat, jelikož hodnota SID se do aplikace již nedostane. [18]

5 NEZABEZPEČENÝ PŘÍMÝ ODKAZ NA OBJEKT

Webové aplikace velmi často používají k navigaci jména souborů nebo primární klíče z databází, které jsou dále předávané jako parametry v URL stránky. To vytváří možnost útočníkovi tyto jména či klíče měnit a dostat se tak do aplikace bez potřebné autorizace popřípadě ji jinak zneužít.

Příkladem zneužití této zranitelnosti je případ z roku 2000, kdy byly hackerem napadeny webové stránky australského finančního úřadu. Hacker změnil DIČ obsažené v URL a získal tak přístup k údajům 17 tisíc společností. [12]

5.1 Příklady zranitelnosti

Velmi často jsou na webu k dispozici aplikace k prezentování fotografií. Tato webová alba nezřídka používají v URL adrese jméno fotografie, která se má zobrazit. Následující URL adresa ilustruje typické použití přímého odkazu na soubor ve webové galerii.

```
http://nebezpecny-web.cz/galerie.php?file=obrazek.jpg
```

Pokud webová aplikace dostatečně neošetří vstup převzatý z parametru URL adresy, může úpravou parametru útočník proniknout do aplikace.

```
http://nebezpecny-web.cz/galerie.php?file=./../../etc/passwd
```

V uvedeném příkladu nahradil útočník platné jméno obrázku za řetězec „./../../etc/passwd“, kterým se snaží dostat k heslům pro přístup serveru.

Dalším příkladem může být zobrazení informací o bankovním účtu. Uživatel je oprávněný k zobrazení informací pouze o svém účtu „123456“. K aplikaci internetového bankovnictví se přihlásí pomocí svých přihlašovacích údajů a systém ho přesměruje na stránku s informacemi o jeho účtu s následující URL:

```
http://nebezpecna-banka.cz/app/ucet.php?cislo=123456
```

Pokud ovšem uživatel pozmění parametr *cislo* na hodnotu „654321“

```
http://nebezpecna-banka.cz/app/ucet.php?cislo=654321
```

aplikace již nezkontroluje oprávnění k zobrazení tohoto účtu a uživatel tak dokáže zobrazit informace o cizích účtech. [18]

5.2 Obrana před zranitelností

Nejspolehlivější cestou jak zjistit, zda je aplikace náchylná k této zranitelnosti, je otestování všech parametrů, kde je využit přímý odkaz na objekt. Aplikace musí konkrétnímu uživateli zobrazit jen data, které korespondují s jeho oprávněním. Aplikace nesmí dovolit zobrazení souborů s konfigurací, zdrojovými kódy a další.

5.2.1 Použití nepřímé reference

Nejlepší cestou je zcela se vyhnout použití přímé reference a nahradit ji za referenci nepřímou. Ta by měla být snadno ověřitelná.

5.2.2 Validace přímého odkazu

V případě nutnosti lze přímý odkaz v aplikaci použít, je však nezbytné, aby aplikace kontrolovala jeho platnost. Webová galerie by měla otestovat existenci požadovaného souboru ve složce s fotografiemi.

5.2.3 Kontrola přístupu

Při použití přímého odkazu na objekt musí být zkontrolována práva pro přístup k danému objektu. Nelze spoléhat na předchozí ověření uživatele. V případě neoprávněného přístupu musí aplikace odmítnout zobrazení a uživatele přesměrovat na stránku s chybovým hlášením. [18]

6 CROSS-SITE REQUEST FORGERY (CSRF)

Velmi nebezpečný typ útoku představuje Cross-site request forgery označovaný jako CSRF a znamená podvržení požadavku mezi různými stránkami. V některých případech se lze setkat s označeními XSRF, Cross-Site Reference Forgery, Session Riding nebo Confused Deputy attacks. Nebezpečnost útoku spočívá v tom, že k napadení aplikace je využit její legitimní uživatel. S jeho autorizací je pak proveden podvodný požadavek v systému. Může se jednat o smazání dat, jejich odeslání útočníkovi, v případě bankovní aplikace odeslání peněz apod. Podvodná akce v systému je většinou skrytá a uživatel systému tak vůbec nemusí vědět, že právě došlo k jeho narušení. [19]

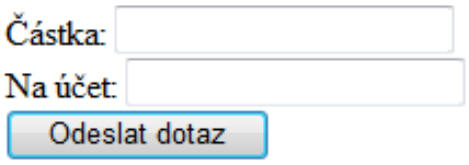
Ochranu proti CSRF je nutné brát v potaz již při návrhu aplikace tak, aby bezpečnostní mechanizmy byly implementovány přímo v jádru aplikace. Jakékoliv další dodatečné úpravy mohou být vzhledem ke značnému dopadu na zpracování požadavků a ověřování jejich platnosti velmi náročné, ne-li zcela nemožné.

Zmíněnou zranitelností byl napadnutelný také vyhledávač Google. Ačkoliv se nejednalo v tomto případě o žádnou destruktivní chybu, bylo umožněno načtením skrytého obrázku na podvodné stránce změnit jazykové nastavení vyhledávače. [19]

6.1 Příklady zneužití

6.1.1 Zneužití dat odesílaných metou GET

Fiktivní bankovní aplikace používá ve formuláři k odesílání platby na cizí účet metodu GET. Zjednodušený formulář pro odeslání platby je na Obr. 11.



Obr. 11 – Část formuláře fiktivní bankovní aplikace

Kód tohoto formuláře je velmi jednoduchý:

```
<form action="preved.php" method="get">
  Částka:
  <input type="text" name="kolik"><br />
  Na účet:
  <input type="text" name="kam"><br />
  <input type="submit" name="PŘEVEĎ">
</form>
```

Díky použití metody GET jsou veškerá odesílaná data předávána v URL. Útočník pak může využít toho, že například načtení obrázku generuje také GET požadavek a do parametru *src* vloží URL, která způsobí podvodné převedení peněz.

```
<img src="http://nabezpecna-banka.cz/preved.php?kolik=10000&naUcet=123456789" />
```

Tento podvodný obrázek umístí útočník na svůj web, nebo v kombinaci s již zmíněným XSS útokem ho umístí např. do široce navštěvovaného fóra. Obrázek může mít nastavenou minimální velikost, případně je pomocí CSS zcela skryt a na stránce tedy nepůsobí rušivě. V okamžiku, kdy je uživatel přihlášen ke svému bankovnímu účtu a načte tento obrázek, dojde k převodu peněz na účet útočníka. Prohlížeč totiž spolu s požadavkem na obrázek odešle do bankovní aplikace i autorizační cookies.

6.1.2 Zneužití dat odesílaných metodou POST

V případě, že aplikace zpracovává data metodou POST je situace mírně obtížnější a je nutné, aby oběť měla aktivovaný ve svém prohlížeči JavaScript, který je použit k odeslání dat. Útočník umístí na svůj web následující formulář:

```
<body onload="document.forms[0].submit();">
<form action=" http://nebezpecna-banka.cz/preved.php" method="post">
  Částka: <input type="text" name="kolik" value="10000">
  Na účet: <input type="text" name="kam" value="123456789">
  <input type="submit" name="PŘEVEĎ">
</form>
```

Jde v podstatě o kopii originálního formuláře s drobnými rozdíly. JavaScript na prvním řádku způsobí okamžité odeslání po načtení stránky bez vědomí uživatele. Dále pak formulář obsahuje již vyplněná data v atributech *value* v jednotlivých textových polích. Útočník opět může formulář na stránce skrýt, aby stránka nebyla ničím podezřelá.

6.2 Obrana proti CSRF

Ve skriptovacím jazyce PHP jsou data přijata webovou stránkou dostupná v superglobálních polích $_COOKIE, $_GET, $_POST a dalších. Všechny hodnoty jsou pak navíc dostupné v poli $_REQUEST. V případě, že aplikace použije data z tohoto pole, mohou být metodou GET podvržena data i přestože formulář je odesílán metodou POST. V poli $_REQUEST již není možné rozlišit, jakou metodou byla data do aplikace předána. Pokud tedy aplikace odesílá data metodou POST, měla by k jejich zpracování používat přímo pole $_POST.

6.2.1 Kontrola referreru

Nejjednodušší, ale také nejméně spolehlivou obranou, je kontrola tzv. referreru, tedy zdroje, odkud je stránka vykonávající požadovanou akci ve webové aplikaci navštívena. Položka *Referrer* je volitelnou položkou HTTP hlavičky.

Tato obrana bohužel není zcela spolehlivá, jelikož údaj o předchozí navštívené stránce lze podvrhnout. Jde tedy spíše o ztížení útoku. Odesílání hlavičky s referrem může být také zachyceno např. proxy serverem a tím znemožněno korektní přihlášení uživatelům připojujícím se tímto způsobem.

6.2.2 Tokeny požadavků

Nejspolehlivější ochranou před tímto útokem je generování náhodných hodnot serverem tzv. tokenů. Server vygeneruje náhodný řetězec, který je umístěn do formuláře nebo odkazu. Při odeslání formuláře dojde k odeslání tokenu, porovnání s vygenerovaným tokenem a v případě shody jsou data uložena. Pokud se tokeny neshodují je detekován CSRF útok a data zahozena.

Server musí nějakým způsobem uchovávat vygenerované tokeny pro účely ověření. Je několik možností, kam tokeny ukládat.

- Do databáze
- Do session

Následující příklad ilustruje, jak je možné implementovat ochranu tokeny uloženými v databázi. Tuto techniku publikoval Jakub Vrána. [20]

```
<?php
$token = rand_chars();

// při posílání odkazu e-mailem
mysql_query("INSERT INTO auth_tokens (token, validity) VALUES ('$token',
NOW() + INTERVAL 1 DAY)");
$url =
"http://$_SERVER[SERVER_NAME]/admin/detail.php?id=$id&token=$token";

// při výpisu formuláře pro editaci nebo smazání
mysql_query("INSERT INTO auth_tokens (token, validity) VALUES ('$token',
NOW() + INTERVAL 1 HOUR)");
echo "<input type='hidden' name='token' value='$token' />\n";

// při provedení akce
mysql_query("DELETE FROM auth_tokens WHERE validity < NOW()"); // smazání
zastaralých
mysql_query("DELETE FROM auth_tokens WHERE token = '" .
mysql_real_escape_string($_REQUEST["token"]) . "'"); // smazání použitého
if (mysql_affected_rows()) {
    // aktualizace záznamu
}
?>
```

6.2.3 Ochrana z pohledu uživatele

I uživatel může částečně přispět ke své vlastní bezpečnosti v obraně před tímto útokem. V případě použití aplikace, u které není známo, zda je proti CSRF útoku zabezpečena, je nejjednodušší použít novou instanci webového prohlížeče. Pokud se používají autorizační cookies platné do konce práce s prohlížečem, nejsou již tyto cookies v nové instanci aplikaci odesílány. Aplikace uchovávající platnost přihlášení mezi jednotlivými spuštěními prohlížeče jsou ovšem zranitelnější. Je vhodné se tedy od nepoužívané aplikace odhlásit, aby nebylo možné přihlášený účet zneužít. [20]

6.2.4 Nette framework

I v případě tohoto útoku je možné k jednoduchému zabezpečení použít framework Nette. Framework obsahuje funkci *addProtection([string $message = NULL], [int $timeout = NULL])*. Funkce generuje autorizační token pro ověření platnosti požadavku. Lze nastavit chybovou zprávu, která bude zobrazena v případě, že autorizační tokeny nebudou souhlasit. Dalším parametrem je platnost tokenu. Možné použití je následující:

```
$form = new Form;
$form->addProtection([string $message = NULL], [int $timeout = NULL]);
```

Token chránící před CSRF útokem má platnost po dobu existence session. Díky tomu nebrání použití ve více oknech najednou (v rámci jedné session). Obrana by měla být aktivována pokaždé, kdy formulář mění nějaká citlivá data v aplikaci. [9]

6.2.5 Další způsoby ochrany

Ochrana před CSRF útokem je možná ještě několika dalšími metodami, které ale nejsou aplikovatelné na všechny situace. Je to zejména z důvodu snížení uživatelského pohodlí.

První možností je použití CAPTCHA[19]. Jedná se o generovaný obrázek s textem a čísly, který je nutno přepsat do textového pole. Aplikace pak ověří shodu vygenerovaného řetězce se zadaným, a pokud jsou shodné, je akce autorizována. Běžně se používá pro odlišení akcí automatizovaných robotů. Použití je vidět na Obr. 12.

Obr. 12 – Formulář s ochranou CAPTCHA

Před provedením důležité akce v aplikaci je také možné nechat uživatele znovu přihlásit pomocí jména a hesla k aplikaci. Tím dojde k dostatečnému ověření platnosti požadavku. Toto řešení je ale značně uživatelsky nepříjemné a komplikuje práci se systémem.

Další možností je vygenerování autorizačního hesla a zaslání prostřednictvím SMS zprávy uživateli. Tento postup je extrémně bezpečný, ale také komplikovaný na implementaci a zvyšuje náklady na provoz aplikace. Využívá se zejména v aplikacích, kde je vyžadovaná vysoká úroveň bezpečnosti, jako například u internetového bankovnictví při potvrzování příkazů k převodu.

[19] **c**ompletely **a**utomated **p**ublic **T**uring test to tell **c**omputers and **h**umans **a**part - plně automatický veřejný Turingův test k odlišení počítačů a lidí

7 NEBEZPEČNÁ KONFIGURACE

Bezpečnost webových aplikací ovlivňuje i úroveň zabezpečení služeb, které slouží k jejich fungování. Provoz webových aplikací se neobejde bez webového serveru. Nejpoužívanější jsou servery Internet Information Services (IIS) od společnosti Microsoft a Apache od Apache Software Foundation. Dalšími nezbytnými službami jsou databázové servery, které jsou nejčastěji zastoupeny produkty Microsoft SQL Server a MySQL. Ke svému běhu dále webové aplikace využívají další desítky služeb, které mohou obsahovat zranitelná místa a narušit tak bezpečnost celé webové aplikace.

I samotná konfigurace serverů může způsobit bezpečnostní problém. Například pokud jsou ponechány pro přístup k serverům přednastavená hesla. U webhostingu může chybnou konfigurací dojít k nedostatečnému oddělení uživatelů sdílející jeden server a tím ohrožení provozované webové aplikace.

Konfigurace webových, databázových, případně i dalších serverů mají v kompetenci většinou administrátoři těchto služeb. Jejich úkolem je zajistit bezpečné prostředí pro běh aplikací. Tato práce se především zaměřuje na doporučení a postupy určené vývojářům aplikací a následující postupy jsou určeny zejména programátorům. Detailní konfigurace serverů jsou mimo rozsah této práce.

7.1 Příklady zneužití

7.1.1 Používání aktuálního softwaru

V kompetencích programátora je z pohledu této zranitelnosti například nasazení aktuální verze frameworku, knihovny a dalších součástí, které jsou použity pro běh webové aplikace. Pokud dojde ve frameworku k nalezení chyby a z tohoto důvodu vydání nové verze, je na programátorovi aplikace využívající tento framework, aby ho co nejdříve do aplikace nasadil.

7.1.2 Konfigurace PHP

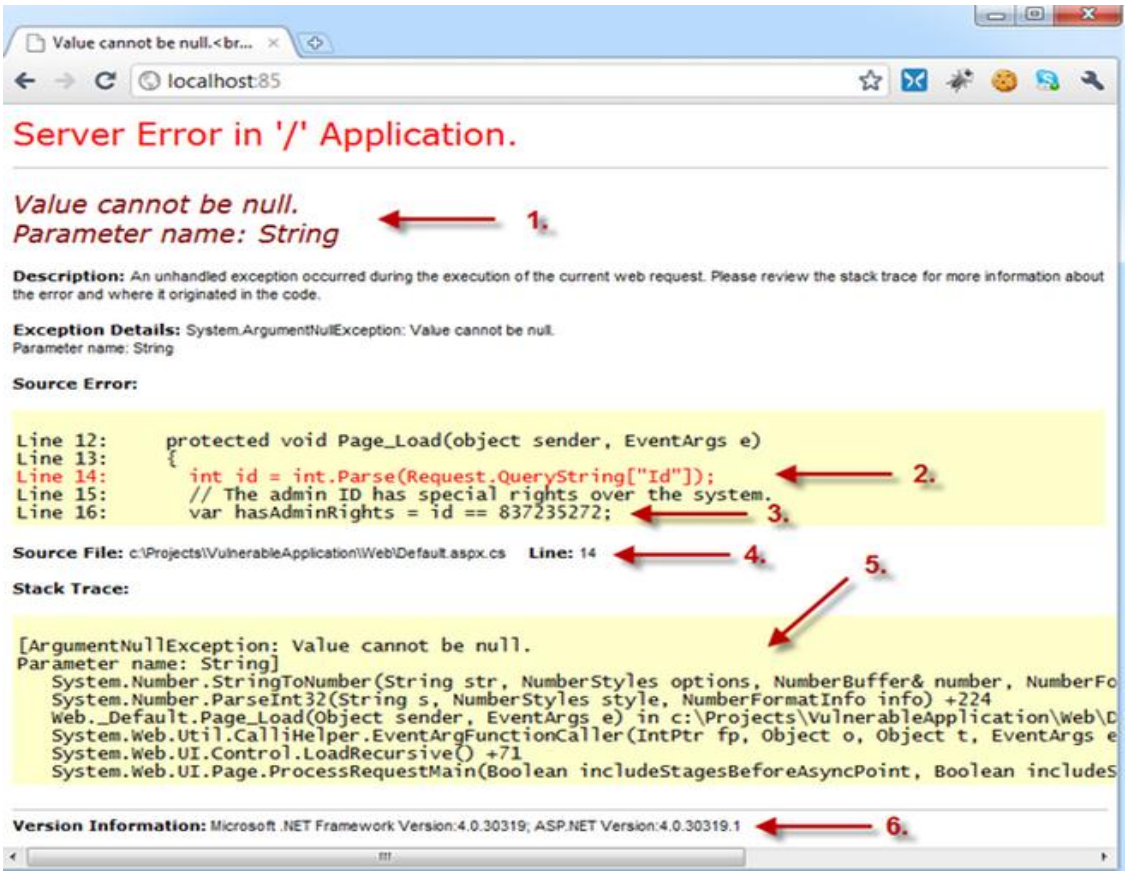
Tabulka 4 uvádí základní přehled konfiguračních direktiv jazyka PHP. Jejich nastavení může programátor ověřit pomocí funkce *phpinfo().*

Direktiva	Popis	Doporučená hodnota
register_globals	Vytvoří globální proměnné ze všech vstupních dat uživatele přicházející z polí GET, POST a COOKIE	Vypnuto
disable_functions	Umožnuje definovat zakázané PHP funkce	exec, passthru, shell_exec, system, proc_open, popen, curl_exec, curl_multi_exec, parse_ini_file, show_source
open_basedir	Direktiva slouží k zamezení přístupu mimo zadaný adresář	Adresář aplikace
allow_url_fopen		Vypnuto
error_reporting	Úroveň podrobností zobrazovaných chyb.	E_ALL
log_errors	Nastavení logování chyb.	Zapnuto
display_errors	Nastavení zda se mají zobrazovat chybové hlášky.	Vypnuto – produkční prostředí Zapnuto – vývojové prostředí

Tabulka 4 – Přehled konfiguračních direktiv PHP

7.1.3 Omezení chybových výstupů

Chybové hlášky, které jsou aplikací produkovány, mohou útočníkovi přinést velmi podrobné informace o použité platformě, verzi softwaru ale i samotném zdrojovém kódu, jak je vidět na Obr. 13 [21]



Obr. 13 – Ukázka nebezpečného chybového výstupu

1. Informace o hodnotě a názvu parametru
2. Odhalení zpracování parametru ID
3. Citlivé údaje pro možný průnik do aplikace
4. Číslo řádku chyby
5. Seznam volání metod před chybou
6. Verze použité platformy

Všechny výše uvedené informace jsou pro návštěvníka zcela zbytečné a neposkytnou mu žádnou informaci o tom, co se právě v aplikaci stalo a jak má pokračovat. Naopak útočník zde najde celou řadu důležitých informací pro útok. Na produkčním serveru se takto podrobné informace nesmí objevit. Framework Nette dokáže podle typu prostředí měnit způsob práce s chybovými hlášeními. Na vývojové prostředí jsou všechny informace

zobrazeny, aby mohl programátor aplikaci odladit a v případě výskytu chyby v produkčním prostředí jsou detailní informace uloženy do logu. Návštěvník dostává pouze obecnou informaci a nedostane se k podrobným zněním chyb jako na Obr. 13.

8 NEZABEZPEČENÉ KRYPTOGRAFICKÉ ÚLOŽIŠTĚ

Webové aplikace již pokrývají velmi široký okruh nabízených služeb a obsahují tak v řadě případů velmi citlivé informace. Téměř každá aplikace musí nějakým způsobem ukládat hesla, e-maily a další kontaktní informace svých uživatelů. Internetové obchody často uchovávají čísla kreditních karet svých zákazníků, aby jim mohly nabídnout komfortní služby jako předplatné online časopisů, nákupy hudby a aplikací do mobilních zařízení a další. Webové aplikace spravují i lékařské záznamy pacientů, čímž usnadňují dostupnost informací mezi lékaři a zároveň poskytují pacientovi ucelené diagnózy a lékařské zprávy. Všechny tyto údaje jsou velmi citlivého charakteru a je třeba věnovat maximální možné úsilí k jejich ochraně před možným únikem a zneužitím.

Ukázkou nezabezpečených citlivých dat z poslední doby je odcizení až 10 milionů čísel kreditních karet ze společnosti Global Payments. V reakci na tento útok zablokovaly i české banky několik desítek platebních karet uživatelů. Na tomto případu je vidět závažnost této zranitelnosti.

8.1 Příklady nechráněných dat

Typickým příkladem špatné ochrany dat je ukládání hesel uživatelů aplikace v textově čitelné formě. Útočník se např. využitím zranitelnosti SQL Injection zmocní exportu dat z databáze aplikace a již má k dispozici velmi citlivá a nešifrovaná data. Na Obr. 14 je vidět výstup z nešifrované databáze.

	id	jmeno	email	heslo	telefon
	1	Karel Novák	knovak@seznam.cz	123456	606852741
	2	Jan Omáčka	jan.omacka@gmail.com	mojeheslo	723123369
	3	Marcela Nová	nova@email.cz	marcelka	776456147
	4	Karolína Slabá	karolinka@seznam.cz	777123789	777123789

Obr. 14 – Ukázka dat v nešifrované databázi

8.2 Šifrování dat jako ochrana

Zašifrováním dat do nečitelné podoby může útočníkovi znemožnit využití získaných informací nebo ho alespoň značně ztížit. Je nutné volit vzhledem k povaze dat pokud možno co nejsilnější šifrovací algoritmy.

Doporučení dle OWASPu [21]:

- Nevytvářet vlastní šifrovací algoritmy
- Nepoužívat slabé nebo již prolomené algoritmy
- Generování klíčů offline a uložení privátních klíčů na bezpečném místě
- Zajištění ochrany přístupů k úložištím a databázím
- Zašifrovaná data nelze jednoduše dešifrovat
- Neukládat nepotřebná data

Hesla k uživatelským účtům lze zabezpečit uchováním jejich hashů. Hash neboli otisk je algoritmus, který převede data libovolné vstupní délky na řetězec s konstantní délkou resp. velikostí. Jedná se o jednosměrnou operaci a z vypočteného otisku nelze rekonstruovat původní vstupní data. Hlavní vlastnosti hashovacích funkcí jsou:

- Konstantní délka
- Nemožnost rekonstrukce vstupních dat
- Malá pravděpodobnost shody otisku pro různá vstupní data
- Malá změna na vstupu (jeden bit) způsobí velkou změnou výstupu (několik bitů)

Mezi běžně používané hashovací funkce patří: MD5, SHA1 a SHA2.

8.2.1 MD5

Hojně využívaným hashovacím algoritmem je MD5. Je popsán v RFC 1321. Ze všech zadaných dat vytváří otisk o konstantní délce 128 bitů. V letech 1996 a 2004 byly v tomto algoritmu nalezeny závažné chyby a jeho použití se tedy **nedoporučuje**.

Dalším argumentem proti použití MD5 k zašifrování uživatelských hesel a dalších údajů je i existence tzv. rainbow tables. Jedná se o databáze s předpočítanými řetězci dat a jejich hashů. V případě, že uživatel použije v systému slabé heslo typu „1234“ je možné jeho hash ve velmi krátkém čase prolomit resp. nalézt hash v rainbow tabulce a získat původní heslo.

8.2.2 SHA1

Funkce SHA1 vytváří ze vstupních dat řetězec o konstantní délce 160 bitů. Byla vydána v roce 1995 jako úprava SHA0. Její použití je doporučené oproti dříve uvedené funkci MD5. V současnosti je k dispozici i SHA2, která i přes lepší zabezpečení není příliš rozšířena. Jako důvod bývá uváděna i horší podpora v systémech Windows XP.

8.2.3 Zvýšení bezpečnosti hashe

Nedostatek spočívající v jednoduchosti hesla je možné samozřejmě řešit nastavením patřičných požadavků na složitost hesla již při registraci do systému. Další možností je k otisku hesla přidat tzv. sůl. Jedná se o náhodný řetězec znaků, který se libovolným způsobem spojí s uživatelským heslem. Až ze spojení tohoto řetězce je spočítán otisk. To i z jednoduchého hesla „1234“ činí heslo řádově složitější a prakticky vylučuje použití rainbow tables.

```
<?php
$heslo = $_POST[heslo];
$sul = microtime();
$hash = hash('sha256', $heslo . $sul);
?>
```

Uvedený skript uživatelské heslo spojí s aktuálním časem v mikrosekundách a z tohoto řetězce je funkcí *hash* spočítán otisk.

Při přihlášení do systému se pak opět spojí uživatelské heslo a „sůl“, spočítá se otisk a ten se následně porovná s otiskem v databázi.

```
<?php
$dotaz = mysql_query("SELECT * FROM uzivatele WHERE login = '" .
mysql_real_escape_string($_POST["login"]) . "' AND heslo_sha2 =
SHA2(CONCAT('" . mysql_real_escape_string($_POST["heslo"]) . "',
salt))");
?>
```

I přesto, že se útočník s odcizenými daty dostane i k „soli“, nemůže předpočítaná hesla použít, jelikož neobsahují unikátní „sůl“. Toto řešení ovšem neochrání aplikaci před útokem hrubou silou. [22], [23]

9 CHYBNÉ ZAMEZENÍ PŘÍSTUPU KE KONKRÉTNÍM URL

Jedná se v podstatě o poměrně banální zranitelnost, kdy nezabezpečená webová aplikace žádným způsobem nechrání stránky, které mají být přístupné pouze vybraným uživatelům, jako jsou administrátoři stránek nebo registrovaní uživatelé. Znepřístupnění stránek je realizováno pouze nezveřejněním jejich URL adres, přes které jsou stránky přístupné. Útočník může URL adresu „privátní“ stránky odhadnout podle její funkcionality, případně ji odhalí některý z oprávněných uživatelů systému. Útočník se po zjištění URL adresy dané stránky nemusí přihlašovat a je mu tak umožněno velmi snadno se dostat neoprávněně k privátním datům.

Tento problém se týká ale také přístupu k nezabezpečeným API[20] rozhraním aplikací. Moderní webové aplikace ve velké míře využívají pro zlepšení uživatelského komfortu technologie AJAX[21] pro okamžitou prezentaci dat. Data jsou často poskytována ve formátu JSON[22] z rozhraní aplikace. Opomenutí zabezpečení v těchto místech aplikace může být podpořeno faktem, že se nemusí jednat o přímou prezentaci citlivých dat prostřednictvím grafického rozhraní v prohlížeči.

S touto bezpečnostní chybou se potýkal web konference Macworld Conference & Expo, který v roce 2008 umožnil díky absenci autorizace uživatele na serveru získání vstupenek za 1700 dolarů zcela zdarma [12].

9.1 Příklady zneužití

Webová aplikace je rozdělena na veřejnou a neveřejnou část. Do neveřejné části se lze přihlásit prostřednictvím URL:

```
http://nebezpecna-stranka.cz/admin
```

Tato stránka zobrazí přihlašovací formulář a po zadání platných přístupových údajů přesměruje uživatele na URL:

```
http://nebezpecna-stranka.cz/prehled_webu.php
```

[20] Application Programming Interface – obecné rozhraní pro přístup k datům a funkcím aplikace
[21] Asynchronous JavaScript and XML – označení technologie využívající jazyka JavaSript k tvorbě interaktivních aplikací
[22] JavaScript Object Notation – způsob zápisu dat, který je platformě nezávislý a čitelný

Útočník zjistí URL, na kterou je uživatel přesměrován a tuto adresu zadá přímo do prohlížeče. Bez nutnosti přihlášení jsou útočníkovi zobrazeny veškerá citlivá data aplikace. Tento postup je velmi banální a využívá toho, že aplikace nepředpokládá přístup jinou cestou než skrze přihlašovací okno.

Dalším příkladem může být internetový obchod poskytující programové rozhraní pro své obchodní partnery. Tato rozhraní mohou obsahovat citlivé informace určené právě jen pro obchodní partnery. Může se jednat o výstupy dat ve formátu JSON, XML, webové služby SOAP[23] a další. Obchodní partneři využívají tato data pro automatický přenos do svých interních systémů. Odhalení citlivých informací může mít pro internetový obchod značný negativní dopad.

9.2 Obrana proti útoku

Každá z „privátních“ stránek musí při pokusu o prohlížení nepřihlášeným uživatelem přesměrovat návštěvníka na přihlašovací formulář. Bez tohoto ověření nesmí zobrazit žádná data ani provést žádnou akci. Ověření musí proběhnout na serveru, nikoliv pouze klientským skriptem v JavaScriptu, jehož vykonání může uživatel ovlivnit např. jeho zakázáním.

V případě, že se jedná o webovou službu poskytující data, je nutné, aby pro jejich zobrazení bylo vyžadováno zadání privátního kódu.

9.3 Testování náchylnosti

Testování je v tomto případě poměrně jednoduché, ačkoliv závisí na rozsahu webové aplikace. Spočívá v kontrole všech odkazů, které mají zůstat přístupné pouze vybraným uživatelům po přihlášení. Pro snadnější kontrolu lze použít roboty, kteří web projdou automaticky a zaznamenají všechny stránky, ke kterým má běžný návštěvník webu přístup. Pokud se na výsledném seznamu budou nacházet i stránky privátního charakteru je nutné doplnit jejich ochranu.

[23] Simple Object Access Protocol - protokol pro výměnu zpráv založených na XML

10 NEZABEZPEČENÁ SÍŤOVÁ KOMUNIKACE

Pro zajištění bezpečnosti webových aplikací je také velmi důležité brát v úvahu bezpečnost informace při přenosu mezi webovým serverem a webovým prohlížečem návštěvníka. Nezašifrovaná data může útočník na síti poměrně snadnou odposlechnout a následně zneužít. Citlivá data by měla webová aplikace zasílat v zašifrované podobě, aby byla pro útočníka nečitelná.

Příkladem ilustrující možnost zneužití může být případ firmy TJX. V tomto případě šlo o krádež dat z nedostatečně zabezpečené bezdrátové WiFi[24] sítě, která sloužila k propojení mezi přenosnými zařízení, pokladnami a počítači obchodu. Síť používala nedostatečné šifrování WEP. [12]

10.1 Příklady zneužití

Webová aplikace nepoužívá ke komunikaci SSL na stránkách vyžadující autorizaci. Útočník pak může monitoringem síťového provozu odchytit přihlašovací údaje, cookies nebo jiná data a ta pak zneužít.

Dalším možným problémem je samotné připojení uživatele. Velmi rozšířené jsou v současnosti domácí WiFi sítě. Zabezpečení těchto sítí je často nedostačující. Útočník se tak může připojit k nedostatečně šifrované WiFi síti a na ní pak odposlouchávat komunikaci.

10.2 Metody ochrany

10.2.1 Ochrana z pohledu uživatele

Z pohledu bezpečnosti síťové komunikace je možné, aby některé kroky pro svou ochranu podnikl i samotný návštěvník webových stránek. Zejména se jedná o adekvátní zajištění bezpečnosti své domácí WiFi sítě. Je zcela vyloučené, aby WiFi síť nepoužívala žádné šifrování a nebyla chráněna heslem. Rizikové je i použití slabých šifrovacích algoritmů

[24] označení pro několik standardů IEEE 802.11 popisující bezdrátová zařízení

např. WEP[25]. Tento typ šifrování je již prolomen a útočníkovi by nezabránil v průniku do WiFi sítě. Doporučuje se tedy používat šifrování WPA2[26].

10.2.2 Ochrana z pohledu programátora

Programátor by pro zajištění bezpečnosti aplikace měl dle nadace OWASP dodržet následující pravidla:

- Používat šifrování pro všechny stránky s přihlášením a všechny privátní stránky
- Používat šifrování pro všechny sítě odesílající citlivá data
- Nepoužívat nešifrované stránky pro zabezpečený obsah

[25] Wired Equivalent Privacy - soukromí ekvivalentní drátovým sítím
[26] Wi-Fi Protected Access - chráněný přístup k Wi-Fi

11 NEPLATNÉ PŘESMĚROVÁVÁNÍ A PŘEDÁVÁNÍ PARAMETRŮ

Poslední zranitelností ze seznamu nadace OWASP je neplatné přesměrování a předávání parametrů. Její umístění na samém konci žebříčku je pochopitelné z důvodu, že škody, které je možné napáchat využitím této zranitelnosti, byly ohodnoceny jako mírné. Zranitelnost využívá již dříve popsaných metod útoku. Jako v již zmíněných slabinách Injection, XSS a dalších, i zde je zneužito nekontrolovaného vstupu dat od uživatele. Tento neověřený vstup je pak zneužit specifickým způsobem, a to k přesměrování na podvodnou stránku.

Touto zranitelností byly dlouhý čas napadnutelné weby významných společností. Jako příklad lze uvést online aukční síň eBay.com. [24]

11.1 Ukázky útoku

Útočník v tomto případě využívá známosti a důvěryhodnosti webu, který danou zranitelnost obsahuje. Do parametru URL vloží odkaz na svoji podvodnou stránku. Odkaz pak s využitím metod sociálního inženýrství distribuuje svým obětem.

```
http://www.ebay.com/presmerovani.php?url=podvodny-web.cz
```

Oběť pak v domnění, že se nachází na důvěryhodném serveru, může útočníkovi sdělit citlivé údaje. Grafická podoba webu bývá v tomto případě naprosto identická s původní důvěryhodnou webovou stránkou. Uživatel tak nemá žádné podezření, že se nachází na podvodném webu.

Dalším možným scénářem zneužití je varianta, kdy aplikace používá skript pro přesměrování na vlastní stránky a dále již neověřuje oprávnění konkrétních uživatelů pro jejich zobrazení. Opět se ale v podstatě jedná i o již popsanou zranitelnost „Chybné zamezení přístupu ke konkrétním URL“.

```
http://www.nebezpeceny-web.com/presmerovani.php?fwd=admin.php
```

Uvedený příklad přesměruje útočníka do administrátorské části aplikace, přestože k tomuto přístupu nemá příslušná oprávnění. Aplikace v tomto případě opět nedostatečně ověřuje uživatelský vstup. [25]

11.2 **Metody obrany**

Jelikož technika provedení útoku je do značné míry totožná s již zmíněnými útoky i metody obrany proti napadení jsou podobné. Základním předpokladem je vždy **kontrola vstupních dat** od uživatele a jejich validace. I v tomto případě je nejvhodnější použít pozitivní validaci neboli „whitelist“. Nadace OWASP uvádí následující kroky jako vhodnou obranu před tímto typem útoku. [25]

- Nepoužívat ve webové aplikaci přesměrování
- Pokud je přesměrování použito, nevkládat uživatelský vstup do parametru přesměrování
- Pokud je nutné, aby parametr přesměrování vzešel z uživatelského vstupu, zajistit kontrolu platnosti a oprávnění přesměrování pro daného uživatele.

ZÁVĚR

Otázka bezpečnosti webových aplikací je vzhledem k jejich rozšíření velmi aktuální a zároveň do jisté míry opomíjena. Zásadními bezpečnostními nedostatky trpí i řada důvěryhodných aplikací velkých společností, což bylo ilustrováno příklady z praxe. V řadě případů se nejedná o extrémně sofistikované útoky, ale naopak většinou je využito některé z výše popsaných zranitelností. Dopady útoku přesto mohou být pro postižený subjekt zcela fatální, jako v případě úniku informací o platebních kartách.

Bezpečnost webové aplikace se velmi často neřeší již v jejím návrhu, ale až dodatečně. Je to zejména z důvodu minimalizace potřebného času k vývoji a tím i snížením nákladů. Tento postup paradoxně může náklady na aplikaci rapidně zvýšit. Bezpečnostní mechanismy nejsou dostatečně provázané s jádrem aplikace a celkovou filozofií funkčnosti. Dodatečné doplňování zabezpečení může být velmi náročné a v případě velmi špatného návrhu může znamenat nutnost kompletního přepsání aplikace.

Tato bakalářská práce popsala deset nejzávažnějších a nejrozšířenějších zranitelností v oblasti webových aplikací. Na ukázkách zdrojových kódů jsou jednotlivé zranitelnosti ilustrovány a je uveden postup, jak zranitelnosti odstranit. Tento seznam by měl sloužit jako minimální přehled pro webové vývojáře a přinést jim informace o bezpečnosti webových aplikací. Kontrola webové aplikace na zranitelnosti uvedené v tomto seznamu by měla být naprostá samozřejmost pro každého tvůrce webu.

Dalším aspektem bezpečnosti webových aplikací je vhodná konfigurace veškerých služeb a platforem nezbytných pro provoz aplikací. Pokud bude bezpečná webová aplikace provozována na serveru, jehož zabezpečení je nízké, bude i tato aplikace velmi náchylná k napadení útočníkem. Popis konfigurace služeb nezbytných k provozu webových aplikací jako jsou webové servery, databázové servery a další, je nad rámec této práce. V této oblasti jsou zmíněny pouze některé obecné předpoklady.

ZÁVĚR V ANGLIČTINĚ

The matter of security of web applications is due to their expansion very accurate and at the same time neglected to some extent. Number of trusted applications, created by large companies, still suffers from major safety deficiencies, which was illustrated using examples. In many cases they are not extremely sophisticated attacks, but in most cases some of the vulnerabilities described above are used. Impacts of the attack may still be entirely fatal, as in the case of an information leak of credit cards.

Security of web applications is very often not addressed in its design, but as an addition to the final product. It's essential in order to minimize the development time and thus reducing cost, but at the end the application's costs can increase rapidly. Safety mechanisms are not sufficiently linked with the core application functionality and overall philosophy. Implementing additional security can be very difficult and in the case of very poor design may lead to a complete redesign of the application.

This thesis described the ten most serious and most common vulnerabilities in web applications. The source code examples are illustrating individual vulnerabilities and instructions on how to remove or avoid them. This list should serve as a minimum list for web developers and provide them information about web application security. Test of the vulnerability of web applications on this list should be a crucial point for each webdeveloper.

Another aspect of web application security is appropriate configuration of all platforms and services necessary for running those applications. If the secure web application runs on a server with low security, this application will be very vulnerable to attack. Configuration of services necessary to run Web applications such as Web servers, database servers and other is beyond the scope of this work. In this area are only some general assumptions mentioned such as the strength of passwords and more.

SEZNAM POUŽITÉ LITERATURY

1. **Český statistický úřad.** INFORMAČNÍ TECHNOLOGIE DOMÁCNOSTI. *Český statistický úřad.* [Online] 2010. [Citace: 24. 03 2012.] http://www.czso.cz/csu/redakce.nsf/i/informacni_technologie_domacnosti_letacek/$File/IT_domacnosti_2011.pdf.

2. **OWASP Foundation.** Main Page. *The Open Web Application Security Project.* [Online] OWASP Foundation. [Citace: 22. 03 2012.] https://www.owasp.org/index.php/Main_Page.

3. —. Top 10 2010-Main. *The Open Web Application Security Project.* [Online] 27. 04 2010. [Citace: 21. 03 2012.]

4. **Závodský, Petr.** OWASP: za webové aplikace bezpečnější. *Root.cz.* [Online] Internet Info, s.r.o., 16. 06 2010. [Citace: 21. 03 2012.] http://www.root.cz/clanky/owasp-za-webove-aplikace-bezpecnejsi/.

5. **Dočekal, Daniel.** Kauza Libimseti.cz ukázala, že nejen král je nahý. *Lupa.cz.* [Online] Internet Info, s.r.o., 30. 10 2008. [Citace: 21. 03 2012.] http://www.lupa.cz/clanky/kauza-libimseti-cz-ukazala-ze-nejen-kral-je-nahy/.

6. —. Sony vs. hackeři? Hackeři stále vedou, další hacknutý web Sony. *JustIT.cz.* [Online] 03. 06 2011. [Citace: 21. 03 2012.] http://www.justit.cz/wordpress/2011/06/03/sony-vs-hackeri-hackeri-stale-vedou-dalsi-hacknuty-web-sony/.

7. **Security-portal.cz.** SQL Injection (Full Paper). *Security-portal.cz.* [Online] 05. 11 2009. [Citace: 21. 03 2012.] http://www.security-portal.cz/clanky/sql-injection-full-paper.

8. **Grudl, David.** Escapování - definitivní příručka. *phpFashion.* [Online] 19. 05 2009. [Citace: 22. 03 2012.] http://phpfashion.com/escapovani-definitivni-prirucka.

9. **Nette Foundation.** Zabezpečení před zranitelnostmi. *Nette framework.* [Online] 17. 06 2011. [Citace: 21. 03 2012.] http://doc.nette.org/cs/vulnerability-protection.

10. **Zend Technologies Ltd.** Zend_Db_Adapter. *Zend Framework.* [Online] [Citace: 24. 04 2012.] http://framework.zend.com/manual/en/zend.db.adapter.html.

11. **The PHP Group.** Magic Quotes. *PHP.* [Online] 20. 04 2012. [Citace: 24. 04 2012.] http://php.net/manual/en/security.magicquotes.php.

12. **Brodkin, Jon.** Hlavní hrozby pro firemní web. *Computerworld.* [Online] IDG Czech Republic, a. s., 09. 02 2009. [Citace: 21. 03 2012.] http://computerworld.cz/internet-a-komunikace/hlavni-hrozby-pro-firemni-web-3400.

13. **Tichý, Jan.** Cross-site scripting. *PHP Guru.* [Online] 22. 02 2008. [Citace: 25. 04 2012.] http://www.phpguru.cz/clanky/cross-site-scripting.

14. **OWASP Foundation.** XSS (Cross Site Scripting) Prevention Cheat Sheet. *OWASP.* [Online] 23. 02 2012. [Citace: 25. 04 2012.] https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.

15. **Microsoft.** How To: Prevent Cross-Site Scripting in ASP.NET. *MSDN.* [Online] 05 2005. [Citace: 25. 04 2012.] http://msdn.microsoft.com/en-us/library/ms998274.aspx.

16. **Nette Foundation.** Šablony. *Nette framework.* [Online] 27. 06 2011. [Citace: 24. 04 2012.] http://doc.nette.org/cs/templating#toc-context-aware-escaping.

17. **Vrána, Jakub.** Defense against XSS in Zend Framework. *PHP triky.* [Online] 13. 02 2012. [Citace: 25. 04 2012.] http://php.vrana.cz/defense-against-xss-in-zend-framework.php.

18. **OWASP Foundation.** Top 10 2010-A4-Insecure Direct Object References. [Online] 02. 05 2010. [Citace: 07. 05 2012.] https://www.owasp.org/index.php/Top_10_2010-A4.

19. **Pejša, Jan.** Co je Cross-Site Request Forgery a jak se mu bránit. *Zdrojak.cz.* [Online] 24. 11 2008.

20. **Vrána, Jakub.** Cross-Site Request Forgery. *PHP triky.* [Online] 24. 04 2006. [Citace: 21. 03 2012.] http://php.vrana.cz/cross-site-request-forgery.php.

21. **OWASP Foundation.** Top 10 2007-Insecure Cryptographic Storage. *OWASP.* [Online] 04 19, 2010. [Cited: 04 24, 2012.] https://www.owasp.org/index.php/Top_10_2007-Insecure_Cryptographic_Storage.

22. **Vrána, Jakub.** Ukládání hesel. *PHP triky.* [Online] 13. 04 2005. [Citace: 21. 03 2012.] http://php.vrana.cz/ukladani-hesel.php.

23. **Tichý, Jan.** Solení hesel aneb Sůl nad zlato. *PHP Guru.* [Online] 30. 10 2007. [Citace: 24. 04 2012.] http://www.phpguru.cz/clanky/soleni-hesel.

24. **Hunt, Troy.** OWASP Top 10 for .NET developers part 10: Unvalidated Redirects and Forwards. [Online] 12. 12 2011. [Citace: 08. 05 2012.] http://www.troyhunt.com/2011/12/owasp-top-10-for-net-developers-part-10.html.

25. **OWASP Foundation.** Top 10 2010-A10-Unvalidated Redirects and Forwards. [Online] 24. 10 2011. [Citace: 08. 05 2012.] https://www.owasp.org/index.php/Top_10_2010-A10.

26. **Vrána, Jakub.** Vypnutí magic_quotes_gpc. *PHP triky.* [Online] 13. 01 2006. [Citace: 21. 03 2012.] Vypnutí magic_quotes_gpc.

27. **Večeřa, Zdeněk.** Jak na to: SQL injection, magic_quotes_gpc, addslashes() a stripslashes(). *Zdeněk Večeřa.* [Online] 21. 05 2009. [Citace: 21. 03 2012.] http://blog.zdenekvecera.cz/item/jak-na-to-sql-injection-magic_quotes_gpc-addslashes-a-stripslashes.

28. **Security-portal.cz.** SQL Injection. *Security-portal.cz.* [Online] 21. 01 2005. [Citace: 21. 03 2012.] https://www.security-portal.cz/clanky/sql-injection.

29. —. Zabezpečení serveru Apache a PHP. *security-portal.cz.* [Online] 11. 11 2004. [Citace: 21. 03 2012.] http://www.security-portal.cz/clanky/zabezpe%C4%8Den%C3%AD-serveru-apache-php.

30. **Závodský, Petr.** Deset nejběžnějších bezpečnostních chyb na webu. *Root.cz.* [Online] Internet Info, s.r.o., 10. 08 2010. [Citace: 21. 03 2012.] http://www.root.cz/clanky/deset-nejbeznejsich-bezpecnostnich-chyb-na-webu/.

31. **OWASP Foundation.** Cross-site Scripting (XSS). *OWASP Foundation.* [Online] OWASP Foundation, 08. 12 2011. [Citace: 21. 03 2012.] https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

32. **Vrána, Jakub.** Cross Site Scripting. *PHP triky.* [Online] 23. 11 2005. [Citace: 21. 03 2012.] http://php.vrana.cz/cross-site-scripting.php.

33. **Security-portal.cz.** Script Injection (PHP remote exploit). *Security-portal.cz.* [Online] 01. 10 2004. [Citace: 21. 03 2012.] http://www.security-portal.cz/clanky/script-injection-php-remote-exploit.

34. **The PHP Group.** mysql_real_escape_string. *PHP.* [Online] The PHP Group, 16. 03 2012. [Citace: 21. 03 2012.] http://php.net/manual/en/function.mysql-real-escape-string.php.

35. **Security-portal.cz.** XSS (Cross-Site Scripting) hacking. *Security-portal.cz.* [Online] 17. 02 2008. [Citace: 10. 05 2012.] http://www.security-portal.cz/clanky/xss-cross-site-scripting-hacking.

36. **Tichý, Jan.** Session ID do URL nepatří. *PHP Guru.* [Online] 04. 09 2009. [Citace: 10. 05 2012.] http://www.phpguru.cz/clanky/sid-do-url-nepatri.

37. **Hunt, Troy.** OWASP Top 10 for .NET developers part 6: Security Misconfiguration. [Online] 10. 12 2010. [Citace: 10. 05 2012.] http://www.troyhunt.com/2010/12/owasp-top-10-for-net-developers-part-6.html.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AJAX	Asynchronous JavaScript and XML – označení technologie využívající jazyka JavaSript k tvorbě interaktivních aplikací
API	Application Programming Interface – obecné rozhraní pro přístup k datům a funkcím aplikace
ASCII	American Standard Code for Information Interchange – definuje kódovou tabulku se znaky abecedy a dalšími znaky
ASP.NET	součást .NET Frameworku pro tvorbu webových aplikací a služeb
BSD	Berkeley Software Distribution
CAPTCHA	**c**ompletely **a**utomated **p**ublic **T**uring test to tell **c**omputers and **h**umans **a**part - plně automatický veřejný Turingův test k odlišení počítačů a lidí
CSRF	Cross-site Request Forgery (nebo také XSRF) je jedna z metod útoku do internetových aplikací
CSS	Cascading Style Sheets – kaskádové styly. Jazyk k popisu formátování webových stránek.
GET	dotazovací metoda v rámci HTTP protokolu.
GNU GPL	licence pro svobodný software, vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí
HTML	HyperText Markup Language – značkovací jazyk pro vytváření webových stránek
HTTP	Hypertext Transfer Protocol – protokol k výměně HTML dokumentů
HTTPS	Hypertext Transfer Protocol Secure – nadstavba protokolu HTTP umožňující šifrovat přenášená data
J2EE	Java Platform, Enterprise Edition – součást platformy Java určená pro vývoj a provoz podnikových aplikací a informačních systémů.
JSON	JavaScript Object Notation – způsob zápisu dat, který je platformě nezávislý a čitelný
MySQL	Databázový systém velmi hojně využívaný při vývoji webových aplikací

OWASP	Open Web Application Security Project – nevýdělečná organizace zabývající se zlepšením bezpečnosti softwaru zejména v oblasti webu
PDO	PHP Data Objects – rozhraní pro přístup k databázím ve skriptovacím jazyce PHP
PHP	rekurzivní zkratka – PHP: Hypertext Preprocesor – skriptovací jazyk k tvorbě dynamických webových aplikací
POST	dotazovací metoda v rámci HTTP protokolu.
SID	Session ID – unikátní identifikátor relace
SQL	Structured Query Language – dotazovací jazyk k tvorbě dotazů v relačních databázích
SSL	Vrstva, která poskytuje zabezpečení komunikace šifrováním a autentizaci komunikujících stran.
URL	Uniform Resource Locator – řetězec identifikující umístění dokumentů na Internetu
WWW	World Wide Web – služba pro poskytování dokumentů HTML v rámci sítě Internet.
XSRF	Cross-site Request Forgery (nebo také CSRF) je jedna z metod útoku do internetových aplikací
XSS	Cross-site scripting – metoda útoku na webové stránky.
AJAX	Asynchronous JavaScript and XML – označení technologie využívající jazyka JavaSript k tvorbě interaktivních aplikací
API	Application Programming Interface – obecné rozhraní pro přístup k datům a funkcím aplikace
ASCII	American Standard Code for Information Interchange – definuje kódovou tabulku se znaky abecedy a dalšími znaky
ASP.NET	součást .NET Frameworku pro tvorbu webových aplikací a služeb
BSD	Berkeley Software Distribution

CAPTCHA	**c**ompletely **a**utomated **p**ublic **T**uring test to tell **c**omputers and **h**umans **a**part - plně automatický veřejný Turingův test k odlišení počítačů a lidí
CSRF	Cross-site Request Forgery (nebo také XSRF) je jedna z metod útoku do internetových aplikací
CSS	Cascading Style Sheets – kaskádové styly. Jazyk k popisu formátování webových stránek.
GET	dotazovací metoda v rámci HTTP protokolu.
GNU GPL	licence pro svobodný software, vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí
HTML	HyperText Markup Language – značkovací jazyk pro vytváření webových stránek
HTTP	Hypertext Transfer Protocol – protokol k výměně HTML dokumentů
HTTPS	Hypertext Transfer Protocol Secure – nadstavba protokolu HTTP umožňující šifrovat přenášená data
J2EE	Java Platform, Enterprise Edition – součást platformy Java určená pro vývoj a provoz podnikových aplikací a informačních systémů.
JSON	JavaScript Object Notation – způsob zápisu dat, který je platformě nezávislý a čitelný
MySQL	Databázový systém velmi hojně využívaný při vývoji webových aplikací
OWASP	Open Web Application Security Project – nevýdělečná organizace zabývající se zlepšením bezpečnosti softwaru zejména v oblasti webu
PDO	PHP Data Objects – rozhraní pro přístup k databázím ve skriptovacím jazyce PHP
PHP	rekurzivní zkratka – PHP: Hypertext Preprocesor – skriptovací jazyk k tvorbě dynamických webových aplikací
POST	dotazovací metoda v rámci HTTP protokolu.
SID	Session ID – unikátní identifikátor relace

SOAP	Simple Object Access Protocol - protokol pro výměnu zpráv založených na XML
SQL	Structured Query Language – dotazovací jazyk k tvorbě dotazů v relačních databázích
SSL	Vrstva, která poskytuje zabezpečení komunikace šifrováním a autentizaci komunikujících stran.
URL	Uniform Resource Locator – řetězec identifikující umístění dokumentů na Internetu
WEP	Wired Equivalent Privacy - soukromí ekvivalentní drátovým sítím
WiFi	označení pro několik standardů IEEE 802.11 popisující bezdrátová zařízení
WPA2	Wi-Fi Protected Access - chráněný přístup k Wi-Fi
WWW	World Wide Web – služba pro poskytování dokumentů HTML v rámci sítě Internet.
XSRF	Cross-site Request Forgery (nebo také CSRF) je jedna z metod útoku do internetových aplikací
XSS	Cross-site scripting – metoda útoku na webové stránky.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK