Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Bezpečnost webových prohlížečů

Diplomová práce

Autor: Patrik Maisner
Studijní obor: IM2-k

Vedoucí práce: Ing. Zuzana Němcová Ph.D.

Hradec Králové duben 2018

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 25.4.2018

Poděkování:
Děkuji vedoucímu diplomové práce Ing. Zuzaně Němcové Ph.D. za metodické vedení práce.

Anotace

Diplomová práce se zabývá problematikou bezpečnosti webových prohlížečů se zaměřením na v současnosti nejvíce používaný prohlížeč Google Chrome. Práce se věnuje zejména otázce bezpečnosti takzvaných rozšíření včetně jejich architektury. Součástí práce je také analýza atributů typických pro škodlivá rozšíření. Na základě analýzy znaků škodlivých rozšíření je navržena metoda pro jejich detekci spolu s automatizovaným systémem, který tuto metodu implementuje.

Annotation

Title: Web browser security

The thesis deals with the security issue of web browsers with emphasis on the most used browser Google Chrome. The thesis focuses on the issue of the so-called extensions including their architecture and security. An analysis of attributes typical especially for malicious extensions was made. Based on the characteristics of such extensions, a method for their detection is proposed along with the automated system that implements this method.

Obsah

Seznam obrázků

1 Úvod

Webové prohlížeče se v současnosti vyvinuly v software, který spravuje velké množství citlivých dat. Kromě vyhledávání jsou prohlížeče používány při vyřizování plateb, obchodování na e-shopu a mnoha dalších činnostech. Uživatel prostřednictvím prohlížeče zasílá značné množství údajů, které lze zneužít. Tato skutečnost z nich dělá velmi častý a oblíbený cíl pro útočníky.

Prohlížeče nabízejí uživatelům krom jiného možnost rozšířit stávající funkcionalitu pomocí tzv. rozšíření. Jedná se o kompaktní aplikace, které lze nainstalovat prostřednictvím veřejně dostupného repositáře, a které mají přístup k rozhraní prohlížeče. Příkladem jsou rozšíření pro správu hesel, blokování reklam, nastavování vlastního vzhledu nově otevřených oken.

Přístup k vysoce privilegovanému rozhraní prohlížeče ovšem vede útočníky k vytváření vlastních rozšíření. Prostřednictvím škodlivých rozšíření se útočníci snaží například o krádeže citlivých informací, falšování autentizovaných požadavků, úpravu obsahu zobrazovaných stránek a vkládání vlastních reklam. Vzhledem k problematické historii škodlivých rozšíření a opakovaným bezpečnostním incidentům se vývojáři rozhodli povolit instalaci pouze z oficiálního repositáře. Na tomto místě dochází k pravidelnému testování a detekci škodlivých rozšíření. Ovšem ani tento přístup není stoprocentní a útočníkům se i tak může povést škodlivé rozšíření do repositáře propašovat. Tato skutečnost dělá z této komponenty prohlížeče tu nejméně bezpečnou a důvěryhodnou. Přesto popularita rozšíření neustále roste, mnoho uživatelů má ve svém prohlížeči některá rozšíření nainstalovaná. Dalším problémem pro detekci škodlivých rozšíření je možnost měnit chování již publikovaných verzí pomocí systému aktualizací. Zmíněné aspekty problematiky systému rozšíření dělají toto téma značně aktuální.

Vzhledem k aktuálnosti této problematiky je tak v této práci představena otázka bezpečnosti webových prohlížečů a jejich rozšíření u nejpoužívanějšího webového prohlížeče, Google Chrome. První část práce se věnuje teoretickým základům síťování (kapitola 3), webových technologií a technologií, na kterých stojí moderní webové prohlížeče (kapitola 4). V další části je nastolena problematika bezpečnosti webových prohlížečů (kapitola 5) a také jejich časté zranitelnosti

(kapitola 6). Důraz je kladen především na zranitelnosti, které souvisí s problematikou rozšíření. Následně je podrobena popisu samotná architektura rozšíření, jednotlivé prvky a také je ve zkratce zmíněna otázka architektury rozšíření v rámci prohlížeče Firefox (kapitola 7). Kapitola týkající se prohlížeče Firefox je zde zmíněna proto, že tento prohlížeč přijal na konci roku 2017 architekturu rozšíření přítomnou v Chrome. Práce se dále věnuje otázce bezpečnosti rozšíření a útokům proti nim (kapitola 8). Pro úplnost je uvedena problematika pluginů (kapitola 9). V poslední části se práce věnuje útokům za použití rozšíření (kapitola 10).

V praktické části (kapitola 11) je vytvořen systém Tali. Jedná se o systém, neboť v sobě obsahuje několik komponent a aplikací, které spolu navzájem komunikují. Tento systém má za úkol detekovat instance škodlivých rozšíření po jejich nainstalování na straně klienta. Cílem systému Tali je ukázat možný přístup k detekci.

2 Cíl práce

Smyslem práce je popsat bezpečnostní problematiku prohlížečů, jejich rozšíření a také navrhnout vlastní systém detekce škodlivých rozšíření.

Při vymezování problematiky bezpečnosti prohlížečů jsou popsány v současnosti používané webové technologie. S technologiemi je spojena i otázka bezpečnosti. Práce si klade za cíl popsat bezpečnostní principy prohlížečů a použitých technologií s důrazem na ty problémy, které se týkají rozšíření.

Dalším cílem je zmapovat útoky proti této komponentě webových prohlížečů a také útoky s využitím této komponenty.

Posledním cílem je navrhnout možné řešení detekce škodlivých rozšíření a následně tento proces automatizovat pomocí systému Tali.

3 Základy webových technologií

Je všeobecně známo, že síťování je základním kamenem webových technologií. Nebýt počítačových sítí, nebyla by možná existence internetu v současné podobě. Z tohoto důvodu jsou v následující kapitole vysvětleny nejdůležitější pojmy týkající se této problematiky.

3.1 ISO/OSI

ISO/OSI je síťový model vytvořený mezinárodním standardizačním úřadem ISO. V rámci tohoto modelu je komunikace mezi zařízeními rozdělena do sedmi vrstev a každá z těchto vrstev řeší jinou otázku přenosu dat na síti. Jednotlivé vrstvy, počínaje nejnižší, jsou následující:

- fyzická – tato vrstva popisuje elektrické nebo optické signály použité při přenosu,
- linková – zajišťuje výměnu dat mezi sousedními počítači v rámci jedné lokální sítě,
- síťová – zde dochází ke směrování dat mezi jednotlivými sítěmi,
- transportní – na této úrovni se zabezpečuje spojení mezi vzdálenými počítači,
- relační – zde dochází k zabezpečení výměny dat a také synchronizaci mezi aplikacemi na síti,
- prezentační – tato vrstva řeší otázku prezentace dat, šifrování, zabezpečení integrity apod.,
- aplikační – popisuje, jakým způsobem mají být data přebírána od aplikačních programů.

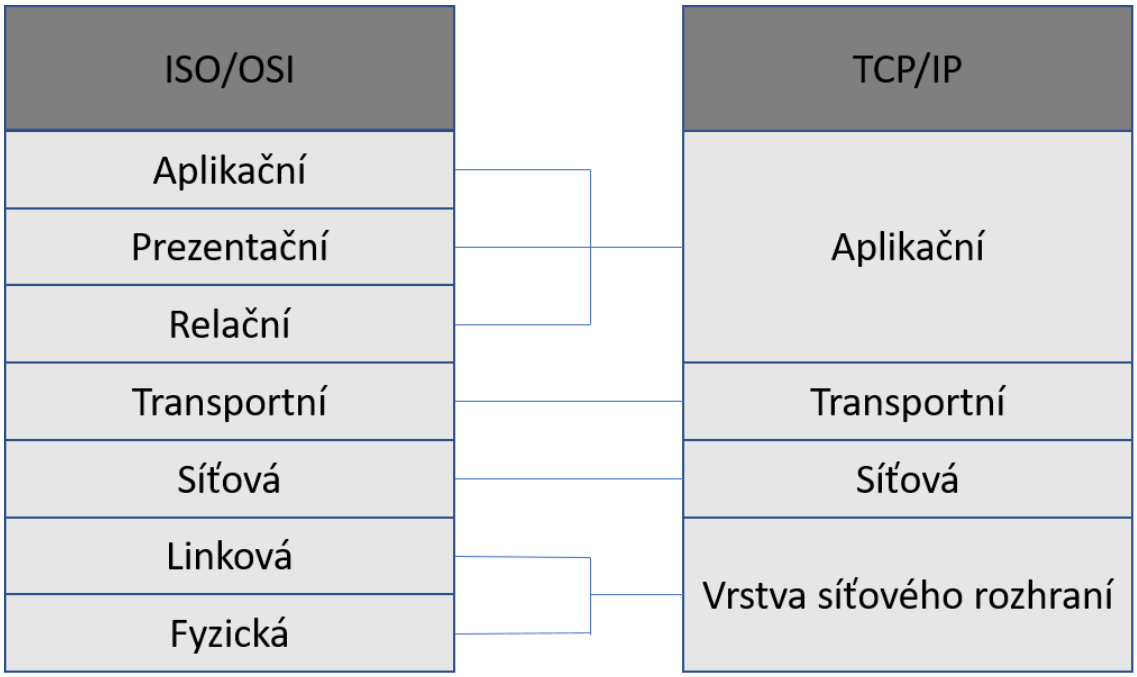
Na každé úrovni jsou definovány protokoly, které řeší úkoly stanovené pro danou vrstvu. (Dostálek a Kabelová, 2000, s. 2-6)

3.2 TCP/IP

Rodina protokolů (a také model) TCP/IP představuje alternativní pohled na uspořádání služeb zabývajících se přenosem dat po síti. Podobně jako v ISO/OSI je i v tomto modelu celý proces přenosu dat řešen více vrstvami. V teoretické rovině

platí, že protokoly v ISO/OSI modelu jsou nesouměřitelné s protokoly TCP/IP. V praxi se ovšem používají protokoly z ISO/OSI modelu pro fyzický přenos dat mezi počítači i v rámci TCP/IP. (Dostálek a Kabelová, 2000, s. 6-8)

Na obrázku číslo 1 je vyobrazen vztah mezi ISO/OSI modelem a rodinou protokolů TCP/IP.



Obrázek 1 - Vztah mezi ISO/OSI a TCP/IP, převzato z (The Network Encyclopedia, 2013)

Z rodiny protokolů TCP/IP jsou zásadní pro tuto práci protokoly IP, TCP a UDP. Pro popis těchto protokolů byla použita publikace (Dostálek a Kabelová, 2000).

IP protokol představuje soubor pravidel, které vymezují způsob komunikace mezi jednotlivými zařízeními v rámci lokální sítě i napříč lokálními sítěmi. Tato schopnost spojovat sítě do větších celků je principem, na kterém stojí současný internet. Tato skutečnost se odráží v samotném názvu tohoto protokolu, neboť se jedná o Internet Protocol. Tento protokol také definuje, že každé zařízení v síti musí mít specifikovanou tzv. IP adresu. Ta tvoří jednoznačný identifikátor.

IP protokol existuje ve dvou verzích. Starší verzí je protokol IP verze 4 (IPv4) a novější je protokol IP verze 6 (IPv6). Novější verze protokolu vznikla jako reakce na omezený rozsah adres nabízený ve starší verzi. Rozsah IPv4 se ukázal být jako nedostatečný při dnešním množství zařízení připojených k internetu.

TCP protokol se zabývá přepravováním dat mezi dvěma aplikacemi (zatímco IP protokol řeší, kam mají data doputovat). TCP pracuje na čtvrté vrstvě OSI modelu (transportní).

TCP je spojovanou službou, což znamená, že mezi dvěma aplikacemi se naváže spojení, které je obousměrné. Při přenosu dochází k číslování jednotlivých bloků dat (záleží tak na pořadí, v jakém jsou přijímány). Poruchám je předcházeno pomocí kontrolního součtu.

Komunikující aplikace na obou stranách připojení jsou určeny tzv. portem. Port představuje dvojbajtové číslo (rozsah 0-65535). K jednoznačnému vymezení aplikace ovšem nestačí pouze číslo, je potřeba specifikovat také protokol. Kromě TCP protokolu používá totiž identifikaci pomocí čísla i UDP protokol. Oba tyto protokoly mají své vlastní rozsahy portů. Například port 53/tcp nemá s portem 53/udp nic společného. Jednoznačně identifikovaná aplikace, kterou může být webový prohlížeč, je identifikována IP adresou, portem a použitým protokolem.

UDP protokol je jednodušší alternativou k TCP. Jedná se o nespojovou službu. Odesílatel pouze odešle data a už se nestará o to, zda proběhl přenos v pořádku. Obdobně jako TCP protokol pracuje UDP na transportní vrstvě ISO/OSI modelu.

3.3 DNS

DNS protokol vznikl za účelem usnadnění adresace jednotlivých prvků sítě. IP adresy jsou pro uživatele těžko zapamatovatelné, a proto se namísto nich používá název síťového rozhraní.

DNS představuje celosvětově distribuovanou databázi mapující jednotlivé doménové názvy na IP adresu s tím, že jedna IP adresa může mít více doménových názvů. Problematika DNS protokolu přesahuje svojí složitostí předmět této práce, proto lze pro bližší informace odkázat na literaturu (Dostálek a Kabelová, 2000, s. 235).

3.4 Architektura sítě

Zařízení v rámci sítě mohou utvořit určitá uspořádání. Z hlediska této práce jsou důležitá dvě dále uvedená. Prvním je centralizovaná architektura klient-server a druhým je decentralizovaná P2P architektura.

Klient-server architektura představuje jeden ze způsobů uspořádání zařízení v rámci dané sítě. Pro tuto architekturu platí, že obvykle jeden počítač disponuje zdroji, které jsou potřebné pro práci více jiných počítačů. Může se jednat o data, službu atd. Počítač nabízející své služby se nazývá server a počítač služby požadující je klient. Pokud chce klient využít zdrojů serveru, musí mu nejprve zaslat požadavek a server mu dle možností odpoví.

V současnosti je značné množství internetových aplikací uspořádáno do této architektury. Obvyklé je uspořádání, v němž je klientem webový prohlížeč, který zasílá požadavek na zdroj – webovou stránku na webovém serveru. (Furht, 2008, s. 61)

Peer-to-Peer (P2P) architektura je alternativou k architektuře klient-server. V rámci tohoto seskupení je spojení počítačů více decentralizované. Jednotlivé počítače mají víceméně ekvivalentní možnosti a zodpovědnosti za zahájení relace a získání či odesílání zdrojů. Jinak řečeno počítače se v tomto zapojení chovají jako klient (zasílají požadavky na zdroje) a zároveň i jako server (řeší požadavky jiných počítačů). Výhodou tohoto spojení je, že není potřeba specializovaného softwaru a také je zajištěna větší spolehlivost (na rozdíl od centralizované Klient-Server architektury, kde selhání serveru je ekvivalentem selhání sítě). Problematika P2P architektury je také nad rámec této práce, lze ovšem odkázat na podrobnější publikaci (Vu, LUPU, OOI, 2010).

4 Principy webových technologií

V této kapitole je popsána problematika webových technologií s důrazem na technologie používané ve webových prohlížečích. Tato kapitola se vychází z publikace (Zalewski, 2012, s. 23-107).

4.1 URL adresa

URL dle RFC 3986 představuje jednoznačný identifikátor, který odkazuje na právě jeden zdroj na serveru. Takový byl alespoň jeho primární účel. V současnosti se tento identifikátor používá i pro zpřístupnění konfigurace prohlížeče, spouštění skriptů, odesílání e-mailu a mnoho dalšího.

Z hlediska identifikace zdroje na internetu se rozlišují dva druhy URL adres; absolutní a relativní. Absolutní odkaz na zdroj v sobě obsahuje všechny informace potřebné pro jednoznačnou identifikaci zdroje na serveru. Na druhou stranu relativní odkaz neobsahuje sám o sobě všechny potřebné informace a je potřeba k takovému odkazu přičíst ještě kontext (obvykle v podobě dokumentu, v němž se relativní odkaz nachází).

Plně specifikovaná absolutní URL adresa, která je validní dle dokumentu RFC 3986, má definovanou následující strukturu:

```
schema://uzivatel.heslo@adresa:port/cesta/ke/zdroji?dotazovaci_retezec#fragment
```

Z tohoto zápisu je zřejmé, že URL adresa je rozdělena do několika částí a každá z nich plní svůj daný účel. Jednotlivé části URL mají své označení a podobně i jejich kombinace mohou mít své pojmenování. Kombinace přihlašovacích údajů, adresy a portu se nazývá autorita.

Relativní adresa by mohla vypadat například následovně:

```
../zdroj.php?dotazovaci_retezec
```

Cesta ke zdroji v rámci serveru má hierarchickou strukturu známou z operačního systému Unix.

Ač každá část výše uvedené absolutní adresy má svůj definovaný význam, tak v rámci specifikace existují i jistá úskalí (viz následující podkapitoly).

4.1.1 Schéma

Schéma je řetězec v rámci URL, který určuje, jaký protokol bude použit pro získání daného zdroje. Mimo alfanumerické znaky může schéma také obsahovat plus, mínus, tečku a zároveň musí vždy končit dvojtečkou. Některá z rozšířených schémat jsou `http:`, `https:`, `ftp:`, `file:`.

Seznam validních schémat eviduje organizace IANA (Internet Assigned Number Authority) na svých stránkách. V seznamu této organizace se nachází značné množství schémat, ale v praxi je často používaná jen jejich malá množina. Seznam zaregistrovaných schémat se nachází na stránkách (Uniform Resource Identifier (URI) Schemes, 2018).

V seznamu organizace IANA se nenachází všechna schémata, která lze v rámci internetového prohlížeče použít. Příkladem je schéma `javascript:`, které slouží k spuštění Javascriptového kódu (dále jen JS). Jedná se o tzv. pseudo-URL adresu. Pseudo-URL adresa obecně neslouží k získání zdroje ze serveru, ale ke zpracování dat na straně klienta.

Rozlišení způsobu získání a zpracování dokumentu prostřednictvím webového prohlížeče se neodvíjí pouze od typu schématu, ale také od absence či přítomnosti dvou lomítek za jeho definicí. V nejjednodušším případě je přítomnost dvojitých lomítek indikátorem absolutní URL adresy a absence pak indikátorem relativní nebo pseudo-URL adresy. Například pseudo-URL `mailto:` způsobí přerušení zpracování adresy a předání dat poštovnímu klientovi v daném počítači. Dvojitá lomítka ale nepředstavují jednoznačně spolehlivý indikátor, neboť jsou mnohdy vynechána i pro jinak korektní absolutní URL adresu. Na webových stránkách se mohou nacházet absolutní URL odkazy o ve formátu `http:stranka.com` .

Zajímavý jev nastane v okamžiku, kdy není URL adresa zadána ve formátu, který by byl očekáván. Může se jednat o URL, které má schéma typické pro absolutní adresu, ale bez dvou lomítek. Nebo se může jednat o schéma typické pro pseudo-URL adresy se znaky lomítek (například `mailto://`). V dokumentu RFC 3986 je tento problém zmíněn, ovšem není nijak konkrétně řešen. Absence konsensu

v rámci specifikace vedla jednotlivé tvůrce webových prohlížečů k hledání vlastních řešení.

Pro prohlížeče (resp. jejich 64 bitové verze) Firefox 57, Chrome 62, Internet Explorer 11 funguje řešení absence lomítek následovně:

- `http:example.com/` adresa je přeložena na `http://example.com` ve všech výše uvedených prohlížečích,
- `javascript://example.com/%0Aalert(1)` adresa vyvolá příkaz `alert()` na výše uvedených prohlížečích. Ke spuštění kódu dojde i přes to, že dvě lomítka by měly uvádět absolutní URL adresu.

4.1.2 Přihlašovací údaje

Jméno uživatele a heslo jsou nepovinné části URL adresy, které umožňují provést autentizaci v okamžiku získávání zdroje ze serveru. Způsob autentizace se odvíjí od použitého protokolu určeného schématem. Pokud ovšem daný protokol nepodporuje autentizaci, pak není chování definováno.

V současnosti je tento způsob autentizace považován za zastaralý. Jako nevhodný k používání byl definován už v dokumentu RFC 3968 z roku 2005. (BERNERS-LEE et al., 2005, s. 17)

S touto částí URL je spojeno úskalí nejednoznačnosti. Pro laika nemusí být zřejmé, že následující adresa:
`http://example.com&gibberish=1234@167772161/`
namísto na zdroj na serveru `example.com` odkazuje na adresu za znakem zavináče v celočíselném formátu (v zápisu IPv4 by se jednalo o adresu 10.0.0.1.). Tento matoucí zápis se tak velmi často využíval ve spamových kampaních.

4.1.3 Adresa

Adresa je část autority, která specifikuje, kde se daný server nachází. Samotná adresa může být podle specifikace určena IPv4 adresou v tečkovém formátu, IPv6 (nebo jakýkoliv budoucí formát) v hranatých závorkách anebo doménovým názvem.

Dále dle RFC dokumentu (BERNERS-LEE et al., 2005, s. 18) lze zadat adresu IPv4 pouze ve standardním formátu s tečkami oddělujícími oktety. Prohlížeče jsou

ovšem v tomto případě značně benevolentnější a přijmou adresu i v jiných formátech. Následující zápisy jsou ekvivalentní:

- `http://127.0.0.1/` je standardní zápis IPv4 adresy,
- `http://2130706433/` je zápis IPv4 adresy v desítkovém formátu jako jedno číslo,
- `http://0x7f.1/` je nestandardní zápis IPv4 s prvním oktetem definovaným pomocí hexadecimální hodnoty a ostatními oktety spojenými v jednu decimální hodnotu,
- `http://017700000001/` je adresou zapsanou v osmičkové soustavě, která je uvedena nulovým prefixem.

Obdobně benevolentní se ukazuje být chování prohlížečů i v případě nestandardních znaků v doménovém názvu. Podle dokumentu RFC 3986 je možné pro doménový název použít alfanumerické znaky, tečku a případně pomlčku. V praxi ovšem prohlížeč pracuje i se znaky mimo tuto množinu. Jedním z příkladů je použití Unicode znaku U+3002 "。" v doménovém názvu. Prohlížeč jej zpracuje jako tečku. Důvodem pro toto chování je, že pro čínské uživatele je snazší zadat tento znak než tečku. Adresy „`http://www.google.com`“ a „`http://www。google。com`“ jsou tak v moderních prohlížečích ekvivalentní (testováno na 64 bitových verzích Firefox 57, Chrome 62, Internet Explorer 11).

4.1.4 Port

Port je nepovinná část URL adresy, pomocí níž lze určit použití nestandartního portu pro danou službu. Prohlížeč běžně komunikuje pomocí TCP nebo UDP protokolu se serverem, takže je potřeba pomocí portu určit aplikaci, se kterou se bude komunikovat. U běžné komunikace se o určení portu stará samotný prohlížeč a není třeba jej speciálně instruovat (port 80 pro HTTP, 21 pro FTP atd.). Pokud ovšem dochází k používání nestandartního portu pro daný protokol, pak je potřeba jej explicitně uvést.

4.1.5 Cesta ke zdroji

Cesta ke zdroji je specifikace daného zdroje na konkrétním serveru. Odpovídá klasickému zápisu absolutní cesty k souboru, který je používaný v operačním systému Unix.

4.1.6 Dotazovací řetězec

Dotazovací řetězec je nepovinná část URL adresy, která umožňuje zasílat parametry na server. Parametry umožňují vyspecifikovat vlastnosti získávaného dokumentu. Tato část URL je uvozena znakem otazníku a jednotlivé parametry mají tvar `název=hodnota`. (BERNERS-LEE et al., 2005, s. 22-23)

To, jakým způsobem jsou jednotlivé parametry odděleny, není v dokumentaci specifikováno. Úmluva mezi vývojáři je, že se používá znak ampersand &. Více parametrů má pak tvar `název=hodnota&název1=hodnota1`.

4.1.7 Fragment

Fragment je další nepovinná část URL adresy, jejíž význam spočívá především v odkazování na části dokumentu přímo na straně klienta. Data nacházející se v části fragment nejsou vůbec odesílána na server, ale jsou zpracovávána zásadně prohlížečem. (BERNERS-LEE et al., 2005, s. 25)

V současnosti se ovšem tato část URL adresy používá i pro ukládání hodnot potřebných pro správnou funkcionalitu skriptu v rámci dokumentu. Hlavní důvod pro tuto variantu je především snížení režie spojené s odesíláním požadavků na server.

4.1.8 Řídicí znaky

Z výše uvedeného je zřejmé, že v URL se nachází několik řídicích znaků vymezující jednotlivé části. Tyto znaky by se neměly nacházet v jednotlivých složkách adresy. Jako rezervované jsou určeny tyto znaky „:“, „/“, „?“, „#“, „[“, „]“ a „@“. (BERNERS-LEE et al., 2005, s. 13)

Mimo řídicí znaky existuje ještě množina znaků, které jsou vyhrazeny pro případné budoucí použití. Jedná se o znaky „!“, „$“, „&“, „'“, „(“, „)“. „*“, „+“, „,“, „;“ a „=“. Znak ampersand našel užití v oddělování parametrů v dotazovacím řetězci.

Existují situace, v nichž se speciální znaky mohou vyskytnout i mimo předpokládané místo. V takovém případě je potřeba je zakódovat. V praxi se běžně užívá procentuální kódování, ve kterém je znak nahrazen procentem a svým ASCII číslem. Znak lomítka se stane `%2F`. Samozřejmostí je, že se musí v případě použití takového kódování zakódovat i samotný znak procenta. Procento se stane `%25`.

4.2 HTTP protokol

HTTP protokol je protokol situovaný na aplikační vrstvě ISO/OSI modelu. V obměnách se používá pro sdílení dokumentů již od roku 1990. Ve své původní verzi HTTP/0.9 umožňoval pouze jednoduchou výměnu dokumentů. Současná verze HTTP/1.1 v sobě obsahuje možnost definování režie přenosu, typu získaného zdroje a mnoho dalšího.

Jedná se o poměrně jednoduchý textový protokol, postavený na rodině TCP/IP. V nejjednodušší podobě je spojení mezi klientem a serverem iniciováno vytvořením TCP připojení k serveru na portu 80. Klient následně zašle URL požadovaného dokumentu. Po odeslání odpovědi server TCP připojení ruší.

Současná verze protokolu HTTP/1.1 řeší kromě zasílání dat i otázku režie. Například to, jak daný dokument v rámci prohlížeče otevřít či interpretovat. K definování režie se používají HTTP hlavičky, což jsou data ve tvaru

```
název: hodnota
```

zasílaná spolu s dokumentem. Příkladem často používaných hlaviček jsou hlavičky:

- `User-Agent` – informace o prohlížeči,
- `Host` – název serveru,
- `Accept` – podporovaný MIME typ; MIME je jednoduchý formát popisující typ souboru (PDF, formulář, …),
- `Referer` – původní stránka (v případě přesměrování), ze které vzešel požadavek na daný zdroj na serveru.

HTTP požadavek může vypadat:

```
GET /priklad.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

```
Host: www.example.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Server na takový požadavek odpovídá vlastní sérií HTTP hlaviček s informacemi o podporované verzi HTTP protokolu, číselném kódu popisujícím způsob proběhnutí požadavku a dalších náležitostech dokumentu. Příklad HTTP odpovědi:

```
HTTP/1.1 200 OK
Server: Apache/2.2.14 (Win32)
Content-Length: 88
Content-Type: text/html
Connection: close
```

Dle (FIELDING et al., 1999, s. 15) mohou být jednotlivé HTTP hlavičky ukončeny znaky CR (Carriage Return), LF (Line Feed) nebo jejich kombinací CRLF. To zvyšuje nároky na vývojáře, kteří nesmí zapomenout filtrovat u uživatelského vstupu všechny variace ukončení řádku. Nedostatečné filtrování znaku zalomení řádku může vést v případě zranitelné aplikace až k útokům proti zpracování HTTP protokolu.

Jedním z útoků proti zpracování HTTP je tzv. rozdělení HTTP odpovědi (HTTP response splitting). Jedná se o typ útoku, který nastává v případě, že aplikace vkládá vstup od uživatele bez ošetření přímo do některé HTTP hlavičky odpovědi. Touto zranitelností mohou trpět aplikace, které umožňují výběr mezi více jazykovými variantami. Pokud aplikace například umožňuje výběr mezi několika jazykovými variantami pomocí odkazů s URL parametrem `lang`

`http://priklad.com/intro.php?lang=`**`cz`**

a server tuto volbu ukládá do HTTP hlavičky v odpovědi

`Set-Cookie: lang=`**`cz`**`; Path=/ ,`

pak v případě neošetřeného vstupu může útočník vytvořit z jedné odpovědi několik za použití znaků CRLF pro nový řádek. Útočník upraví odkaz pro výběr jazykové verze na:

```
http://priklad.com/intro.php?lang=cz%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%0d%0aContent-
Type:%20text/html%0d%0aContent-
Length:%2013%0d%0a%0d%0avlastni_obsah
```

kde `%0d%0a` je znak pro CRLF zakódovaný pomocí procent. Tento požadavek vytvoří následující odezvu:

```
HTTP/1.1 200 OK
Set-Cookie: cz
Content-Length: 0
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 13

Vlastní obsah

Zbytek z první odpovědi…
```

Útočník tímto způsobem vytvoří dvě odpovědi. V druhé odpovědi může být nebezpečný skript nebo obecně cokoliv, co si útočník přeje, aby se zobrazilo oběti. Tato zranitelnost je velmi nebezpečná v okamžiku, kdy webová aplikace používá server pro cache (ukládání odpovědi serveru tak, aby bylo možné tuto odpověď opakovaně odesílat při stejném požadavku a snížila se tak zátěž). V takovém případě může útočník vyslat po prvním požadavku s výběrem jazykové verze okamžitě druhý požadavek na jiný zdroj na serveru, který chce tzv. „otrávit". Cache server poté spojí druhý požadavek s uměle vytvořenou druhou odpovědí. Tato situace vede k zobrazování dokumentu útočníka při požadavku na legitimní zdroj na severu (Stuttard a Pinto, 2011, s. 534-535).

4.2.1 Typy HTTP požadavků

Pro práci se zdroji na webovém serveru definuje protokol HTTP/1.1 celkem 8 následujících metod:

- GET slouží pro získání zdrojů ze serveru. Tento požadavek nemá vyvolat žádnou trvalou změnu stavu aplikace.
- POST slouží k zasílání informací serveru (především formulářových polí viz kapitola 4.3.1) pro zpracování s tím, že tento požadavek může způsobit změnu stavu aplikace. POST požadavek obvykle obsahuje také tělo s parametry.
- HEAD je spíše méně využívaná metoda podobná GET s tím rozdílem, že nedochází k získání celého zdroje ze serveru. Vrací se jen HTTP hlavičky. Obvykle tuto metodu používají vyhledávače pro ověření existence dokumentu.
- OPTIONS vrací seznam podporovaných metod na serveru.
- PUT slouží k nahrávání souborů na serveru, absolutní většina serverů jej ovšem z bezpečnostních důvodů nepodporuje.
- DELETE je opakem k metodě PUT, takže tato metoda slouží k mazání zdrojů ze serveru. Vzhledem k bezpečnostním důsledkům je skoro vždy tato metoda zakázaná.
- TRACE je metoda sloužící k testování provozu, jedná se o jistou obdobu příkazu PING. Tato metoda vrací informace o všech mezilehlých zařízeních mezi klientem a serverem a zároveň vrací zpět původní požadavek od klienta. Tato metoda může odhalit strukturu sítě, proto bývá skoro vždy zakázaná.
- CONNECT slouží především k vytvoření připojení skrz mezilehlé zařízení (které vystupuje jako proxy server). Metoda bývá obvykle zakázána, protože pokud by server umožňoval se připojit pomocí CONNECT, pak by útočník mohl vytvořit spojení do interní sítě přidružené k danému serveru. Nebo zneužít cizí server jako proxy a vydávat se tak za tuto entitu při dalším postupu.

4.2.2 Kódy odpovědi serveru

V rámci RFC 2616 existuje značné množství definovaných kódů, pomocí kterých může server informovat klienta o tom, jak proběhlo zpracování jeho požadavku. Hlavní kategorie těchto odpovědí jsou:

- 200-299 značí úspěšné dokončení požadavků. Mezi nejvíce používané z této kategorie se budou počítat kódy:
  - `200 OK` v případě správného zpracování požadavku,

    - `204 No Content` v případě úspěšného požadavku, u kterého není předpokládaná odpověď.
- 300-399 jsou používány pro statusy, které sice nejsou chybové, ale požadují speciální zpracování ze strany klienta, například
    - `301 Move Permanently` kód, který instruuje prohlížeč k navštívení nové URL definované v odpovědi v HTTP hlavičce `Location:` (jedná se o přesměrování).
- 400-499 jsou chybové stavy na straně klienta. Může se jednat o
    - `400 Bad Request` server není schopen daný požadavek zpracovat v tom formátu, v jakém je zaslán,
    - `401 Unauthorized` chybějící patřičné autentizační údaje.
- 500-599 jsou především chyby na straně serveru, například
    - `500 Internal Server Error` kód může být vyvolán špatnou konfigurací, programátorskou chybou a podobně.

Výše uvedené kódy jsou sice uvedeny v rámci RFC dokumentu, ovšem nebývají vždy obsluhovány správným způsobem. Webové aplikace leckdy vrací `200 OK` i v případě, že došlo k chybě atp.

4.2.3 Cookies

Cookies, aktuálně definované v dokumentu RFC 6265 (Barth et al., 2011), nejsou součástí definice protokolu HTTP/1.1. Jedná se ovšem o jeho velmi důležité rozšíření.

Server může prostřednictvím hlavičky `Set-Cookie: cookie=hodnota` instruovat klienta k zasílání páru `cookie=hodnota` ve všech svých následovných požadavcích prostřednictvím HTTP hlavičky `Cookie`. Server může tímto způsobem identifikovat jako související různé požadavky pocházející od jednoho klienta. (Barth et al., 2011, s. 3)

Cookies také představují nejrozšířenější způsob, pomocí něhož lze udržovat jedno sezení a požadavky od uživatele autentizované bez potřeby opakovaně zadávat přihlašovací údaje při každé žádosti o zdroj. Mimo cookies lze pro tento účel použít HTTP autentizaci, kontrolu pomocí IP adresy anebo také certifikáty

Cookies mohou mimo samotný název a hodnotu obsahovat dodatečné atributy. Ty se uvozují středníkem. Jedná se například o:

- `Expires` specifikující čas vypršení platnosti cookie,
- `Path` určující v rámci jaké cesty na serveru se bude cookie aplikovat,
- `Secure` atribut určující, zda může být cookie zaslána pouze zašifrovaným kanálem (HTTPS),
- `HttpOnly` atribut určující, zda může být cookie čitelná i pro skripty (jedná se o obranu proti krádeži prostřednictvím XSS útoku viz kapitola 6.1).

4.2.4 HTTPS

Ač HTTP protokol dokáže řešit režijní věci spojené s přenosem dokumentů a dat relativně vhodným způsobem, tak sám o sobě neklade příliš velký důraz na bezpečnost. Data zasílaná prostřednictvím tohoto protokolu jsou v čisté nezašifrované podobě a každý prvek mezi klientem a serverem si může přečíst obsah zpráv (pokud by k šifrování nedocházelo přímo v konkrétní aplikaci). Vzhledem k jasným bezpečnostním dopadům byla představena alternativa k HTTP; HTTPS. Pokud má přenos probíhat prostřednictvím HTTPS, pak je potřeba jej uvést ve schématu v URL.

Přenos přes HTTPS probíhá prostřednictvím SSL nebo TLS protokolů. Tyto dva protokoly v sobě obsahují popis mechanismu výměny klíčů mezi klientem a serverem. Po výměně klíčů dochází k vytvoření spojení a samotné výměně zpráv. HTTPS stojí dále na certifikátech ověřujících identitu daného serveru a asymetrické kryptografii. (Pedersen, 2005)

4.3 HTML kód

HTML představuje hlavní formát a jazyk pro tvorbu online dokumentů na internetu. Základní syntaxe tohoto jazyka je jednoduchá. V rámci HTML se tvoří hierarchická struktura značek, které mají parametry a které obsahují textové uzly ve svém těle.

Základní šablona webové stránky vytvořené pomocí nejnovější verze HTML5 vypadá následovně (A Basic HTML5 Template For Any Project, 2016):

```
<!doctype html>

<html lang="en">
<head>
  <meta charset="utf-8">

  <title>The HTML5 Herald</title>
  <meta name="description" content="The HTML5 Herald">
  <meta name="author" content="SitePoint">

  <link rel="stylesheet" href="css/styles.css?v=1.0">

  <!--[if lt IE 9]>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/html5shiv/3.7.3/html
5shiv.js"></script>
  <![endif]-->
</head>

<body>
  <script src="js/scripts.js"></script>
</body>
</html>
```

V předchozím příkladu HTML stránky lze vypozorovat, že existuje množina řídicích znaků, která značí začátek a konec značek, název a hodnotu parametru a jiné. Jedná se o znaky ostrých závorek, jednoduché a dvojité uvozovky a o značku ampersand. Tyto znaky by se neměly ve validním HTML dokumentu nacházet uvnitř hodnoty parametru, ostré závorky by se neměly nacházet v těle jiné značky a podobně.

Nedostatečné ošetření těchto znaků může vést ke zranitelnosti XSS, kdy je útočníkův kód zaslán serveru (může se jednat o URL parametr v rámci GET požadavku) a který je na serveru vložen bez ošetření do navrácené stránky. Tato zranitelnost může vést až ke kompletní kompromitaci prohlížeče oběti (viz kapitola 6.1).

Samotný HTML dokument může být zpracováván dvěma různými způsoby. Jedním z nich je XHTML mód, v němž program pro zpracování obsahu způsobí výjimku při chybě v syntaxi na stránce.

Oproti tomu HTML mód se snaží někdy až příliš „agresivně“ uhodnout, jaký byl záměr autora. Vyplní neuzavřené párové značky, názvy parametrů mohou být zapsány malými a velkými písmeny, hodnoty parametrů mohou být bez uvozovek atd.

V rámci HTML dokumentu se nachází i značky, které vyžadují přepnutí kontextu zpracování obsahu. Jedná se hlavně o značky `<script>` a `<style>`, které v rámci XHTML nemusí být správně zobrazeny. Od toho slouží konstrukce

```
<script>
<![CDATA[
var x = 0
…
]]>
</script>
```

která zabrání zobrazení samotného textu.

Problematika vkládání řídicích znaků do těla jednotlivých elementů v HTML dokumentu je řešena pomocí kódování entit. Toto kódování má jednoduchou syntaxi; daný kód je uvozen znakem ampersand, dále následuje textové označení a nakonec je zapsán středník. Levá ostrá závorka se tak stává `&lt;`, pravá ostrá závorka `&gt;`, samotný ampersand se zakóduje jako `&amp;` atd. Kódování funguje i v rámci parametrů a značek ve zdrojovém HTML dokumentu.

4.3.1 HTML elementy

V této kapitole jsou popsány některé HTML elementy, které jsou zásadní z hlediska tématu této práce.

Odkazy představují nejjednodušší způsob, jakým lze vytvořit referenci na jiné zdroje v těle dokumentu. Odkaz může obsahovat i netradiční schéma jako

`javascript:`, `mailto:` atp. Dále odkazy mohou obsahovat parametr `target`, který popisuje, v jakém kontextu se má nový dokument otevřít. Nový dokument se může otevřít na nové stránce nebo může nahradit aktuálně zobrazovaný dokument. Příklad klasického odkazu je

```
<a href="http://www.priklad.com/">Odkaz</a> .
```

Formuláře představují způsob, jakým lze zaslat informace na server (alternativa k parametrům zapsaným přímo v dotazovacím řetězci). Svým způsobem se jedná o obdobu odkazu s tím rozdílem, že data zaslaná na server se vytváří dynamicky s ohledem na to, jaké hodnoty v sobě formulář obsahuje. Příkladem formuláře může být:

```
<form method=GET action="/ stranka.cgi">
  Jméno: <input type=text name=jmeno>
  Příjmení: <input type=text name=prijmeni >
  <input type=submit value="klikni!">
</form>
```

Tento formulář obsahuje pole pro jméno a příjmení spolu s tlačítkem pro zpracování formuláře typu `submit`. Samotný formulář může odesílat akci typu GET (GET je i implicitní varianta v případě, že není vyplněna žádná metoda) specifikovanou parametrem `method`. V případě odeslání tohoto formuláře dojde ke zpracování všech hodnot formulářových polí a jejich přepsání pomocí procentového kódování. Dále nastane jejich sestavení a odeslání v části URL s parametry na relativní adresu popsanou v parametru `action`. Dynamicky vytvořený požadavek by mohl vypadat například následovně:

```
/stranka.cgi?given=Erwin&family=Schr%C3%B6dinger .
```

Metoda formuláře může být ovšem nastavena i na typ požadavku POST. Existuje několik formátů zaslání požadavků, ale nejčastější jsou následující dva:

- Klasický způsob, tedy data jsou zakódována stejně jako v případě GET požadavku, jen jsou zaslána v těle zasílané zprávy. Tento mód se definuje v HTTP hlavičce:

`Content-Type: application/x-www-form-urlencoded .`

- Druhý formát umožňuje upload souborů na server. Formulář musí obsahovat pole typu `file`:

```
<input type="file">
```

a musí být zapsán se správným atributem

`enctype="multipart/form-data" .`

Rámce jsou značkou, která umožňuje zobrazit jeden dokument v rámci druhého. Nejčastějším typem jsou značky IFRAME:

`<iframe src="http://www.priklad.com/"></iframe> .`

Rámec má svůj vlastní kontext, ve kterém pracuje JS. Dále platí, že dokumenty v různých rámcích se nesmí ovlivňovat, pokud nepochází se stejného zdroje (viz kapitola 5.1).

4.4 CSS

CSS slouží k formátování a určení vzhledu HTML dokumentů. V současnosti je nejnovější verze CSS3.

CSS lze aplikovat v HTML třemi různými způsoby. První možností je vytvořit v dokumentu blok `<style>`, druhý způsob je načíst externí soubor s pravidly formátování a třetí je použít vnořené formátování (využít atributu `style` pro HTML značky).

Pokud vzhled není aplikován použitím atributu `style` u konkrétní značky, pak je potřeba definovat pomocí selektoru, na které prvky se bude formátování vztahovat. V nejjednodušší podobě by takové formátování mohlo vypadat následovně:

```
img {
  border-size: 1px;
  border-style: solid;
}
```

Výše uvedený kód používá obecný selektor `img` pro vybrání všech obrázkových elementů na stránce a nastavuje jim nepřerušovanou hranici o tloušťce 1 pixel. Selektor se ovšem nemusí vztahovat jen na obecné elementy, ale může se vztahovat na libovolnou jejich podmnožinu.

CSS3 umožňuje také použít @ direktivy, které mohou sloužit mimo jiné i k načtení externího CSS stylu do dokumentu.

4.5 Skripty na straně klienta

Skripty na straně klienta umožňují manipulaci HTML dokumentu, zobrazování dialogových oken, otevírání nových dokumentů a obecně automatizaci chování v prohlížeči. Nejrozšířenějším jazykem na straně klienta je Javascript (JS).

JS je v základní podobě relativně jednoduchý jazyk, který je interpretován dynamicky (tedy až v okamžiku jeho čtení). Syntaxe vychází z programovacího jazyka C. Javascript disponuje automatickou správou paměti, která uskutečňuje uvolňování paměti po již nepoužívaných objektech. Dále se jedná o jazyk s dynamickým typováním. To znamená, že typ proměnné (číslo, řetězec) je určen až podle toho, jaká data jsou do proměnné uložena.

Každý HTML dokument zobrazený v prohlížeči má vlastní kontext, ve kterém JS pracuje. Tento kontext definuje omezení, které kód v dokumentu má. Za jinak nezměněných podmínek totiž platí, že skripty z různých dokumentů (například načtených do více rámců na jedné stránce) se nemohou navzájem ovlivňovat. Tento stav je bezpečnostním mechanismem, kterému se v angličtině říká sandboxing.

Zpracovávání JS kódu probíhá v prvním kroku tak, že se zkontroluje správnost syntaxe každého bloku (blok může být všechen kód v jedné značce `<script>`). Pokud je syntaxe správná, pak dojde k převodu do binární podoby. V opačném případě dojde k vyvolání syntaktické chyby, program pro čtení ukončí zpracovávání bloku a přeskočí na další. Následující kód má dva bloky a záměrně zavedenou syntaktickou chybu v prvním bloku:

```
<script>
  var my_variable1 = 1; // Po chybě tento řádek nemá význam
  var my_variable2 =
```

```
</script>

<script>
  2;
</script>
```

Prohlížeč v okamžiku nalezení chyby přestane zpracovávat první blok. Toto přerušení zpracování má za důsledek ignorování všech instrukcí v chybné části. První blok tak nemá žádné globální dopady na dokument. Druhý blok je ekvivalentní prázdné instrukci.

Zpracování funkcí je další krok po kontrole syntaxe. V tomto kroku dochází k registraci názvů funkcí, které prohlížeč nalezl v syntakticky správných blocích. Důsledkem tohoto způsobu zpracování je, že následující dva bloky kódu nejsou ekvivalentní. První

```
<script>
hello_world();

function hello_world() {
  alert('Test!');
}
</script>
```

proběhne v pořádku a po jeho spuštění vznikne v rámci stránky okno s pozdravem. Ve druhém případě

```
<script>
hello_world();
</script>

<script>
function hello_world() {
  alert('Test!');
}
```

```
</script>
```

dojde k chybě. Například pro Firefox 57 se bude jednat o chybu `ReferenceError: hello_world is not defined`. Tento skript nefunguje , protože jednotlivé bloky nejsou zpracovávány najednou, ale podle pořadí výskytu na stránce. Jinak řečeno dochází k volání funkce `hello_world()` před interpretací bloku s definicí.

Ke správnému spuštění kódu nedojde ovšem ani v tomto případě:

```
<script>
hello_world();

var hello_world = function() {
  alert('Test!');
}
</script>
```

Důvodem chyby je opětovné volání funkce před její definicí.

Po registraci funkcí proběhne v následném kroku interpretace instrukcí mimo funkce. K chybě může dojít v okamžiku tzv. výjimky (chyba při zpracování instrukce). V případě výjimky nedochází ke stornování doposud provedených změn v daném bloku. Pouze pro daný segment skončí zpracovávání instrukcí a prohlížeč přeskočí na další úsek kódu. Názorně lze tuto skutečnost demonstrovat pomocí skriptu:

```
<script>
function not_called() {
  return 42;
}
function hello_world() {
  alert("Test!");
  do_stuff(); // Výjimka, není definovaná funkce.
}
```

```
alert("Welcome to our demo application.");
hello_world(); // Výjimka

alert("Text."); // Nedojde ke spuštění kvůli výjimce.
</script>

<script>
  alert("Text"); // Bude spusteno
</script>
```

Toto chování, kdy vyvolání výjimky nezpůsobí stornování předešlých instrukcí v bloku, může vést až k nekonzistentnímu stavu v aplikaci.

Zpracovávání JS funkcí v rámci prohlížeče funguje na principu fronty (první dovnitř a první ven). Volané funkce jsou často zpracovávány synchronně, což znamená, že v případě výpočetně náročné operace dojde k zablokování ostatních akcí v prohlížeči. Prohlížeč tzv. „zamrzne“. S ohledem na tuto architekturu nenabízí prohlížeče funkci pro uspání zpracování kódu na určitý časový okamžik.

JS je jazyk interpretovaný až v okamžiku čtení zdrojového kódu. Tato skutečnost umožnuje dynamicky vytvářet a zpracovávat vlastní kód za pomocí funkce `eval()`.

4.5.1 Objekty prohlížeče

JS prostředí je postaveno okolo objektu reprezentujícího defaultní jmenný prostor pro všechny globální proměnné i funkce s názvem `window`. Prohlížeč kromě možnosti definovat vlastní funkce nabízí rozhraní, pomocí kterého lze s prohlížečem pracovat. Například funkce `alert()` je definována na hlavním jmenném prostoru. Může se ovšem jednat i o další používané funkce, jako jsou `blur()`, `focus()`, `setInterval()` atd.

V defaultním jmenném prostoru je definováno několik objektů, mezi nejzajímavější patří:

- `location` objekt, který nabízí funkce pro čtení URL dokumentu, v němž se skript nachází,

- `history` objekt, který slouží k pracování s historií prohlížeče, posouvání se v rámci historie a vkládání záznamů,
- `screen` objekt, který slouží pro definování rozměrů prohlížeče, okna atd.,
- `navigator` objekt, pomocí něhož lze získat informace o verzi prohlížeče,
- `document` objekt, pomocí něhož lze manipulovat s objekty v rámci dokumentu, jako jsou obrázky, cookies atd. Pomocí tohoto objektu lze získat přístup k Document Object Model.

Document Object Model (dále DOM) představuje stromovou reprezentaci hierarchie značek v rámci HTML dokumentů. Každá značka spolu se svými vlastnostmi a formátováním má svoji objektovou reprezentaci. Pro práci s tímto stromem nabízí JS několik metod. Jednou z nich je například `getElementById()` metoda volaná na objektu `dokument` pro vyhledávání elementů. DOM tedy zjednodušeně řečeno nabízí rozhraní pro práci s elementy v rámci HTML dokumentu.

5 Bezpečnostní principy prohlížečů

5.1 Same Origin Policy

Same-Origin policy (dále jen SOP) je poměrně starý koncept, který byl představen již roku 1995 firmou Netscape. Základní myšlenka tohoto konceptu je jednoduchá: pokud máme dva a více webových dokumentů v prohlížeči (například v několika rámcích `iframe`), pak tyto dokumenty spolu mohou komunikovat pouze v případě, že mají totožný protokol (schéma), doménový název a port. Tato kombinace definuje tzv. kontext stránky. Zabránění komunikace mezi dokumenty pocházejícími z jiných zdrojů slouží jako prevence krádeže dat či nebezpečné manipulace s DOM strukturou stránky z jiného zdroje. (Zalewski, 2012, s. 142)

Kombinace protokolu, doménového názvu a portu tvoří tzv. původ (anglicky origin). Podle tohoto principu se dva dokumenty odkazované pomocí URL adresy `http://www.example.com` a `https://www.example.com` liší v použitém protokolu a mají tedy jiný původ. Dále `http://www.example.com` a `http://example.com` se liší v doménovém názvu.

Nebýt tohoto principu, tak by útočník mohl kupříkladu velmi jednoduše zcizit data z e-mailového účtu hostovaného na jiném serveru. Stačilo by nalákat oběť na vlastní server a na stránce vytvořit rámec odkazující na poštovní schránku. Pokud by byla oběť přihlášena, útočník by si mohl přečíst obsah schránky pomocí svého JS kódu.

I přes pozitivní bezpečnostní dopady SOP může v mnoha situacích nastat, že spolu dva dokumenty z různých zdrojů potřebují komunikovat. Například komunikace mezi přihlašovací

`login.example.com`

a platební stránkou

`payments.example.com`

na straně klienta.

Dva způsoby, jak umožnit komunikaci dvou dokumentů z různých zdrojů, jsou použití vlastnosti `document.domain` v prohlížeči nebo použití `postMessage()` API (nabízené rozhraní). Každý z těchto přístupů má svá specifika.

5.1.1 document.domain

Použití `document.domain` přichází na řadu v okamžiku, kdy dva komunikující dokumenty sdílí koncovku ve svém doménovém názvu. Schéma a port obou dokumentů musí být totožné, jinak změna v této vlastnosti nebude mít požadovaný efekt.

Výše uvedený příklad problému stránek `payments.example.com` a `login.example.com`, které spolu chtějí komunikovat, lze vyřešit nastavením vlastnosti `document.domain` na stejnou hodnotu pro oba zdroje (stejný sufix). Obě stránky by provedly následující instrukci:

```
document.domain = "example.com" .
```

Poté by nic nebránilo jejich komunikaci a vzájemnému ovlivňování DOM.

Je nezbytně nutné, aby tuto instrukci provedly obě webové stránky. Pokud provede tento příkaz jen jedna stránka, pak není umožněno těmto dokumentům spolu komunikovat. Prohlížeč si interně eviduje, zda je vlastnost editovaná nebo původní.

Z bezpečnostního hlediska se tento způsob komunikace stává problematickým, pokud existují dokumenty i na jiných subdoménách domény `example.com`. Lze uvažovat stránku s adresou `blogs.example.com`, která bude mít značně menší nároky na zabezpečení oproti přihlašovací a platební stránce. Pokud se bude na této stránce vyskytovat zranitelnost umožňující útočníkovi spustit vlastní kód na straně klienta, pak mu nic nebrání změnit `document.domain` vlastnost na hodnotu `example.com` a bez omezení zpřístupnit a měnit třeba platební stránku v prohlížeči oběti. Jinak řečeno, změna vlastnosti `document.domain` redukuje bezpečnost celé aplikace na bezpečnost nejméně zabezpečené subdomény dané domény. (Zalewski, 2012, s. 143)

5.1.2 postMessage

HTML5 nabízí API pro umožnění komunikace mezi dokumenty s jiným původem. Především se jedná o funkci `postMessage()`, která řeší některé problémy spojené s užíváním `document.domain`. Ve zkratce tento mechanismus nabízí možnost zasílat libovolně dlouhou zprávu jinému dokumentu, který má k obsluze těchto

zpráv připravenou funkci. Výhodou tohoto přístupu je, že odesílatel může jednoznačně specifikovat příjemce a příjemce může přijímat zprávu jen od určitého odesílatele. V tom je zásadní rozdíl oproti přístupu s použitím `document.domain`, který umožnil i jiným dokumentům (z ostatních subdomén) zpřístupnit obsah, což vedlo k nepříjemným bezpečnostním důsledkům.

Praktické použití tohoto API může být následující. Stránka pro provedení plateb je identifikována adresou `https://payments.example.com`. Tato stránka potřebuje získat informace o uživateli. Aby toho docílila, tak načte do rámce dokument s adresou `https://login.example.com`. Tento rámec po načtení provede následující příkaz:
`parent.postMessage("user=bob", https://payments.example.com); .`
Prohlížeč tuto zprávu zašle původnímu dokumentu pouze v případě, že jeho původ je totožný s tím, který je definovaný v druhém parametru funkce. V druhém dokumentu, u příjemce, musí být definovaná obsluha. Ta může vypadat následovně:

```
addEventListener("message", user_info, false);
function user_info(msg) {
  if (msg.origin == "https://login.example.com") {
    // zpracovaní dat
  }
}
```

Výše uvedený přístup je považován za bezpečnější než použití vlastnosti `document.domain`. Tento mechanismus je ovšem také náchylný k nesprávnému použití. Následující kód je považován za nebezpečný:
`if (msg.origin.indexOf(".example.com") != -1 { … } ,`
neboť útočník může snadno vytvořit doménu `www.example.com.attacker.com` a tím obejít filtrování podle příjemce. Závažnost této chyby se značně odvíjí od toho, k jakým operacím dochází po přijetí zprávy.

5.1.3 SOP a XMLHttpRequest

`XMLHttpRequest` API umožňuje zasílat a zpracovávat HTTP požadavky a odpovědi v těle samotného dokumentu. Za nezměněných okolností lze požadavek odesílat jen

v případě, že požadavek směřuje na server, z něhož původně dokument pochází. Mezi doménou dokumentu a serveru musí být kompletní shoda a ani změna `document.domain` nemá na tento typ API žádný vliv. `XMLHttpRequest` má ovšem jednu velmi zásadní vlastnost z pohledu bezpečnosti. Zasílání požadavků tímto způsobem dává odesílateli možnost upravovat HTTP hlavičky. V případě běžných požadavků HTTP hlavičky nastavuje prohlížeč. (Zalewski, 2012, s. 146)

Přílišná kontrola, jaké HTTP hlavičky jsou odesílány prostřednictvím požadavku `XMLHttpRequest`, vede k závažným bezpečnostním důsledkům. Jedním z nich může být úprava HTTP hlavičky `Content-Length`. Správně naformátovaná zpráva umožní „propašovat" dodatečné požadavky v rámci jednoho HTTP sezení (viz kapitola 4.2).

Z toho důvodu se vývojáři webových prohlížečů rozhodli zakázat úpravu některých HTTP hlaviček a jejich tvorbu nechat kompletně na prohlížeči. Na stránkách vývojářů prohlížeče Firefox se nachází seznam hlaviček, které nelze programově upravovat. Jedná se o `Content-Length`, `Connection`, `Host`, `Origin` a mnoho dalších. (Forbidden header name, 2017)

Za jistých okolností mohou být požadavky zasílané prostřednictvím `XMLHttpRequest` směrované na jinou doménu, než je doména dokumentu. SOP situaci komunikace mezi různými doménami zabraňuje, a tak pro tyto případy bylo zavedeno rozšíření `Cross-Origin Resource Sharing` (CORS). To povoluje zasílat tento typ požadavků i na jiné domény. Aby prohlížeč zjistil, zda tento mezidoménový požadavek server povoluje, zasílá předběžný požadavek.

5.2 Bezpečnostní HTTP hlavičky

HTTP hlavičky představují jednoduchý mechanismus, pomocí něhož může server instruovat prohlížeč k použití dodatečných bezpečnostních opatření. HTTP hlaviček existuje nepřeberné množství, zde bude věnována pozornost hlavně hlavičkám důležitým z hlediska bezpečnosti. Pro výčet bezpečnostních hlaviček byla použita publikace (Alcorn, Frichot a ORRÙ, 2014, s. 13-14).

Content Security Policy (zkráceně CSP) je mechanismus navržený pro zabránění XSS útokům. CSP zavádí hlavičku `Content-Security-Policy` (případně `X-`

`Content-Security-Policy`). Tato hlavička specifikuje, odkud lze načíst skripty (z jakých domén), zda lze skript vkládat jako atribut HTML elementů, zda lze použít z hlediska bezpečnosti problematickou funkci `eval()` atd.

Jedná se o efektivní mechanismus používaný pro potlačení XSS zranitelností, neboť špatně ošetřený vstup na stránce nemusí vést při správné konfiguraci CSP ke spuštění útočníkova vloženého skriptu. Pokud se například zranitelnost nachází v atributu některého elementu, ale CSP zakazuje vkládání skriptu do atributů, pak nedochází k provedení škodlivého kódu.

Secure příznak vznikl jako reakce na skutečnost, že cookies jsou zasílány s požadavky prostřednictvím HTTP hlaviček nehledě na to, zda je provoz šifrovaný či nikoliv. Tato skutečnost má několik problematických důsledků týkajících se bezpečnosti, proto bylo zavedeno dodatečně značení cookies. Toto značení říká, že cookies mohou být zasílány pouze prostřednictvím zašifrovaného kanálu (HTTPS namísto HTTP).

Příznak HttpOnly je další označení cookies v rámci HTTP hlaviček. Toto označení říká, že daná cookie nemůže být zpřístupněna ze žádného skriptu, čímž se zabraňuje zcizení autentizačního tokenu pomocí XSS.

X-Content-Type-Options je hlavička, která vznikla jako reakce na skutečnost, že prohlížeče se snaží „uhodnout“ typ souboru vráceného ze serveru na základě jeho obsahu. Útočník může v takovém případě zneužít chování prohlížeče a například na server nahrát soubor, který je vydáván jako jpg obrázek, ale v prohlížeči bude interpretován jako skript (pokud stránka zakazuje nahrávání skriptů). Pro zabránění takové situace je použita direktiva `nosniff`, která zakazuje prohlížečům odhadovat typ souborů na základě obsahu a která nařizuje prohlížeči kontrolovat typ pomocí MIME.

Strict-Transport-Security hlavička instruuje prohlížeč k tomu, aby veškerá komunikace se serverem probíhala jen pomocí HTTPS

X-Frame-Options hlavička slouží k zabránění načítání stránky do rámců (`<iframe>`). Tato hlavička byla zavedena jako obrana proti metodě útoku s názvem clickjacking. V rámci tohoto útoku dochází k načtení třetí stránky do neviditelného rámce na určité stránce, kterou útočník vytvořil. Útočník se snaží nalákat oběť ke kliknutí na správné místo tak, aby proběhla akce na neviditelném dokumentu. Více o tomto útoku v kapitole 6.5. (Stuttard a Pinto, 2011, s. 511-512)

6 Zranitelnosti webových prohlížečů

V této kapitole jsou popsány zranitelnosti webových prohlížečů, které souvisí s problematikou rozšíření. V této kapitole jsou informace čerpány z publikace (Stuttard a Pinto, 2011, s. 431-515).

6.1 XSS

Cross Site Scripting (dále XSS) je velmi stará zranitelnost. Vyskytuje se v několika variantách, které se liší způsobem, jakým probíhá útok proti klientovi. Společným jmenovatelem všech těchto technik je, že dochází ke spouštění útočníkova kódu na straně klienta (v jeho prohlížeči). Pro tento útok je typické, že dochází ke zneužití důvěry prohlížeče, že daný kód pochází skutečně od webové aplikace.

6.1.1 Reflektovaný typ XSS

Reflektovaný typ XSS (anglicky reflected XSS) je velmi častá zranitelnost, která se často nachází v moderních webových aplikacích. Mnohdy se projevuje prostým zobrazováním uživatelského vstupu na výstupní stránce (část HTTP požadavku se stává součástí těla HTTP odpovědi) v neošetřeném formátu. Nejčastějším reprezentantem této zranitelnosti je výpis zprávy, která svůj text přebírá bez zpracování z URL. Například aplikace, která vypisuje chybové hlášení přímo z URL parametru `message`:

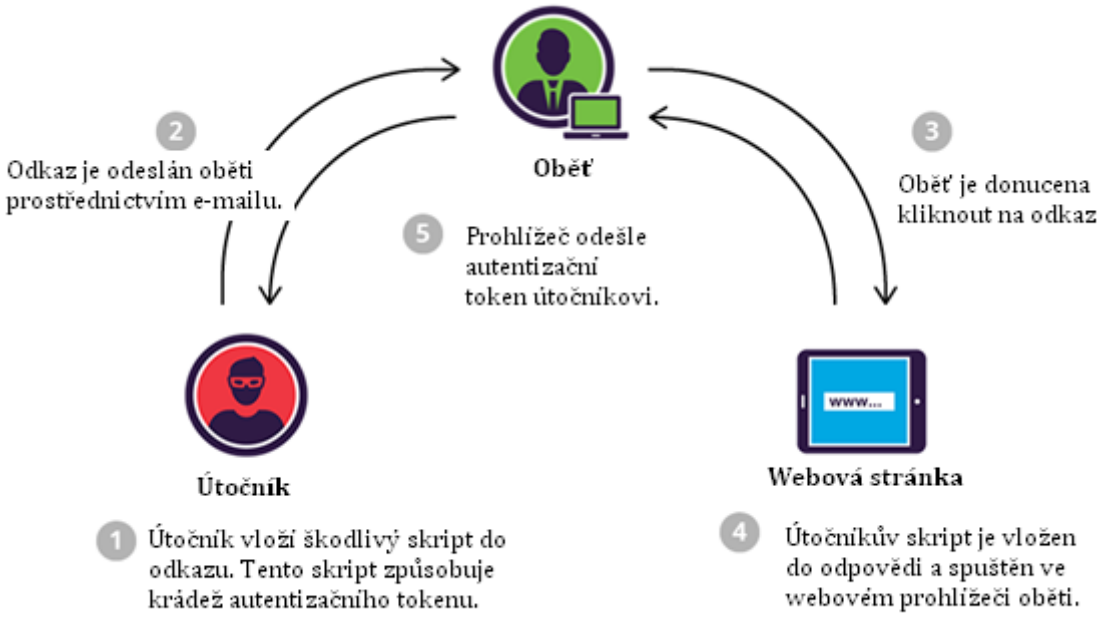
`http://mdsec.net/Error.ashx?message=Sorry%2c+an+error+occurred .`

Pokud útočník vloží do parametru vlastní kód, pak dojde po návštěvě URL adresy k jeho spuštění v prohlížeči klienta. Podvrhnutá URL adresa by vypadala:

`http://mdsec.net/Error.ashx?message=<script>alert(1)</script> .`

Uvedená adresa povede ke spuštění funkce `alert()` okna v rámci prohlížeče. Nápadnost této URL adresy se dá změnit použitím služeb pro zkracování adres.

Název reflektovaný typ XSS značí skutečnost, že je potřeba vytvořit takový požadavek na server, který obsahuje útočníkův kód, a aplikaci, která tento kód následně zobrazí (reflektuje) uživateli. Nejlépe je tento proces vidět na následujícím obrázku číslo 2 (demonstrace zcizení cookies za použití XSS).



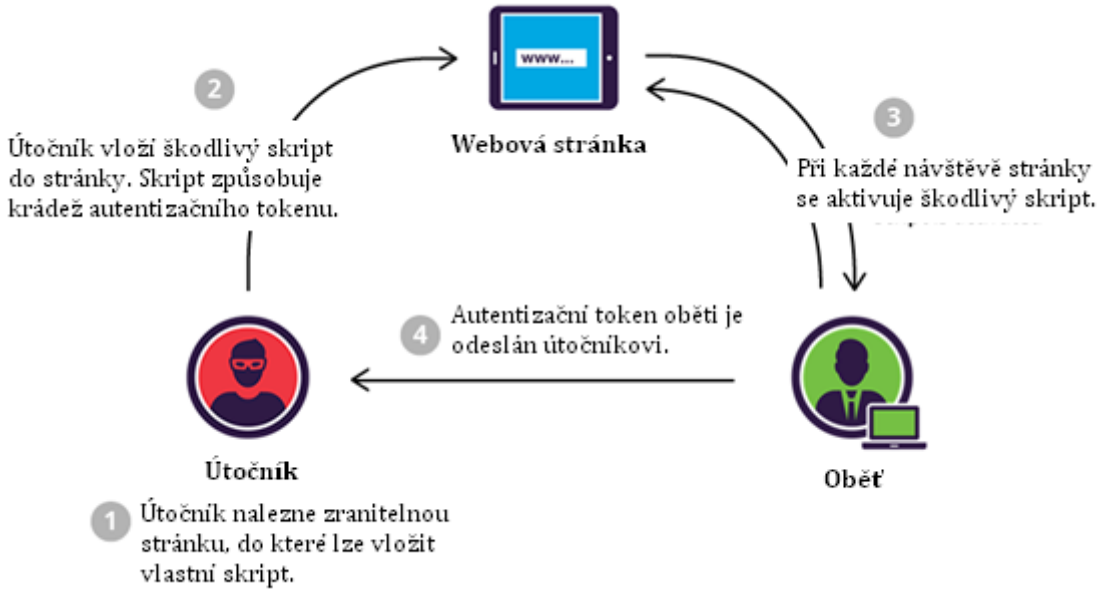
Obrázek 2 - Reflektovaný typ XSS, převzato z (REFLECTED CROSS SITE SCRIPTING (XSS) ATTACKS, 2018)

Tento útok obvykle obnáší jeden požadavek a jednu odpověď, proto se také někdy nazývá XSS prvního řádu.

6.1.2 Uložený typ XSS

Další kategorií je uložený XSS (anglicky stored XSS). Tento typ zranitelnosti obvykle vyvstane v okamžiku, kdy dochází k ukládání zaslaných dat od uživatele bez ošetření na straně serveru (obvykle v databázi) a tato data jsou následně opět zobrazena uživatelům. V okamžiku návštěvy stránky, která útočníkovi data zobrazí, dochází k útoku. Typická je tato zranitelnost pro stránky, v níž dochází k interakci mezi jednotlivými uživateli prostřednictvím společného média (například fórum).

K provedení uloženého typu XSS je potřeba minimálně dvou požadavků. Nejlépe je situace vidět na následujícím schématu (obrázek 3).



Obrázek 3 - Uložený typ XSS, převzato z (CROSS SITE SCRIPTING (XSS) ATTACKS, 2018)

V prvním kroku (požadavku) dochází k zaslání dat, která jsou následně uložena na serveru. V druhém kroku (požadavku) oběť zobrazí stránku s daty útočníka. Vzhledem k tomu, že jsou potřeba dva po sobě jdoucí požadavky, tak se tento typ XSS nazývá XSS druhého řádu

Z bezpečnostního hlediska je tento typ XSS závažnější než reflektovaný typ XSS. Útočníkovi odpadá složitý problém s přesvědčováním oběti k návštěvě vygenerovaného odkazu. Stačí návštěva stránky v libovolném okamžiku od spuštění útoku.

6.1.3 DOM XSS

Pro oba předchozí typy XSS platí, že musí útočníkův kód být zpracován webovou aplikací na straně serveru (ať už bezprostředně, nebo s odkladem). Webová aplikace zpracuje útočníkův kód a následně jej zašle uživateli zpět. Při provedení DOM XSS nedochází ke zpracování požadavku na straně serveru. Veškerý kód je proveden pouze na straně klienta. Jak název napovídá, tento útok souvisí s DOM strukturou dané stránky.

Častou obětí tohoto útoku se mohou stát aplikace, ve kterých dochází ke zpracování parametrů z fragmentů v URL na straně klienta a jejich zobrazování ve stránce (parametry za znakem # viz kapitola 4.1.7). Ukázka skriptu:

```
<script>
  var url = document.location;
  url = unescape(url);
  var message = url.substring(url.indexOf('message=')
                + 8, url.length);
  document.write(message);
</script>
```

Jelikož tento skript bez jakéhokoliv zpracování převezme hodnotu z parametru `message` v URL a zobrazí ji na stránce, tak útočník může vytvořit tento odkaz: `http://site.net/Error.asp?message=<script>alert('')</script>` . Ten povede při navštívení ke spuštění kódu na straně klienta.

Určitým způsobem je tento typ zranitelnosti více podobný reflektovanému typu XSS, protože je potřeba přesvědčit oběť k návštěvě útočníkem sestavené URL.

6.1.4 Obrana

Otázka XSS je spojena s problematikou přílišné důvěry ve vstup od uživatele a také značnou variabilitou míst, kam může být vstup vložen. Pro využití zranitelnosti stačí jen jedno neošetřené místo z desítek až stovek běžně se v daném dokumentu nacházejících.

Obrana proti reflektovanému a uloženému typu XSS je v mnoha ohledech podobná. Data od útočníka jsou vkládána do stránky buď do nebezpečných míst v dokumentu, nebo útočník může vkládat řídicí znaky do svého vstupu a měnit tak kontext. Kontextem je zde myšlen různý způsob zpracování dat v závislosti na jejich místě v dokumentu. Například data u elementu v kontextu atributu `value` jsou zpracována jinak než data v kontextu atributu `onclick`. Z takové úvahy plyne, že je potřeba identifikovat tzv. řídicí znaky (měnící způsob zpracování dat) a také místa vstupu. Prevenci výše zmíněných dvou typů XSS lze vymezit do třech bodů:

1. validace vstupu,
2. validace výstupu a
3. eliminace nebezpečných míst, kam lze vložit data.

Validace vstupu by měla obsahovat sledování toho, zda nejsou data příliš dlouhá, zda obsahují jen povolené znaky a zda odpovídají formátu (vložená e-mailová adresa je opravdu adresou a ne skriptem).

Validace výstupu značí především proces eliminace nebezpečných znaků, které mohou měnit kontext. Například změna kontextu při použití následující konstrukce je velmi jednoduchá:

```
<input type="text" value="vstup">
```

Pokud nedochází k dostatečnému ošetření dat, pak lze změnit způsob zpracování dat pomocí

```
"><script>alert();<script><" .
```

Útočník může vytvářet vlastní elementy pro skriptování nebo měnit strukturu stránky.

V rámci HTML existuje několik řídicích znaků, které umožňují měnit kontext. Jedná se o znaky, které lze snadno překódovat do bezpečné podoby pomocí HTML entit:

- " je html entita `&quot;`,
- ' je html entita `&apos;`,
- & je html entita `&`,
- < je html entita `<`,
- > je html entita `>`.

Výše uvedeným ošetřením nebezpečných znaků lze eliminovat XSS ve velké většině případů. Ovšem i když webová stránka není náchylná ke zranitelnosti XSS, tak může být zranitelná vůči jiným technikám (viz kapitola 6.2). Na bezpečnost je tak potřeba nahlížet systémově.

Některá místa by neměla zpracovávat uživatelský vstup v žádném případě. Jedná se o místa atributů pro obsluhu událostí nebo obsah `<script>` značek. I přes

existenci filtru existuje velká šance pro nalezení nebezpečné sekvence, která filtr obejde.

Speciální kategorii obrany je obrana proti DOM XSS. Z hlediska návrhu totiž není vhodné, aby JS kód zpracovával DOM a následně výsledek operace vkládal do stránky. Nedochází totiž k zasílání na server, jenž by data zpracoval, a tak zaregistroval případný útok. Mechanismus obrany proti DOM XSS je značně komplexní a je tak nad rámec této práce.

6.2 Podvržení požadavku

Podvržení požadavku (anglicky Request Forgery) je typ útoku, pomocí kterého lze unést prohlížeč a donutit jej k činnosti, která nemá původ u samotného uživatele. V principu jsou tyto útoky spojeny se situací, v níž útočník provádí činnost a je při tom autentizován jako některý z běžných uživatelů. Zvláštností tohoto útoku je, že pro podvrhnutí oběti identity není potřeba znát autentizační token (cookies ani přihlašovací heslo).

6.2.1 On-Site Request Forgery

On-Site Request Forgery (OSRF) je typ podvržení požadavku od uživatele, který probíhá v kontextu jedné webové aplikace. Svým způsobem se jedná o útok podobný uloženému typu XSS s tím rozdílem, že i když je stránka chráněna proti XSS, tak může stále být zranitelná vůči OSRF.

Příkladem může být následující situace. Lze uvažovat aplikaci fóra, na kterém uživatelé mohou zadávat různé typy příspěvků. HTTP požadavek pro vytvoření příspěvku vypadá následovně:

```
POST /submit.php
Host: wahh-app.com
Content-Length: 34

type=question&name=daf&message=foo
```

jehož výsledkem je příspěvek v následujícím formátu.

```
<tr>
<td><img src=”/images/question.gif”></td>
<td>daf</td>
<td>foo</td>
</tr>
```

Vkládání obsahu je dostatečně ošetřeno na znaky <,>,", ' atd., takže není aplikace náchylná k útokům typu XSS. Na první pohled to nemusí být zřejmé, ale útočník ovládá ještě jednu část stránky. Tou je obsah atributu `src` elementu `<image>`. Pokud útočník vyvolá požadavek podobný tomu předešlému, s tím rozdílem, že parametr `type` bude mít hodnotu:
`../admin/newUser.php?username=daf2&password=0wned&role=admin# ,`
tak při zobrazení stránky kýmkoliv dojde k zaslání HTTP požadavku na tuto stránku a z hlediska serveru se bude zdát, že původcem je oběť.

V okamžiku zobrazení stránky s podvrženým obrázkem administrátorem nastane vytvoření nového účtu s administrátorskými právy. Znak mřížky slouží pro eliminaci přípony .gif z požadavku (fragment v URL). K funkčnosti tohoto útoku je potřeba, aby vytváření uživatelů probíhalo použitím metody GET namísto metody POST.

6.2.2 Cross-Site Request Forgery

V rámci Cross-Site Request Forgery (CSRF) dochází k vytvoření stránky útočníkem, která po zobrazení v prohlížeči oběti způsobí HTTP požadavky na zranitelnou aplikaci. Oproti OSRF tak není útok limitován pouze na kontext jedné stránky

Ve své podstatě se u tohoto útoku jedná o požadavek, který pochází z jiné domény, než na kterou je směřován. SOP v tomto případě nedokáže tomuto útoku zabránit. Důvodem je fakt, že SOP implementace v prohlížečích nezabraňuje zasílání požadavku, ale až zpracování odpovědi.

Hlavní důvod existence CSRF útoku (resp. zranitelnosti) je, že webová aplikace považuje za dostačující pro autentizaci uživatele použít jen jeden autentizační token (obvykle cookies). Pokud dojde k první autentizaci pomocí jména a hesla, pak server zašle patřičné cookies prohlížeči. Ten je na oplátku zasílá

s každým následným požadavkem na server. Z předchozí věty plyne nutnost být přihlášeny pro to, aby byl útok úspěšný.

6.2.3 Obrana

Obranou proti CSRF je použití dodatečného tokenu. Tento token umožňuje sledovat, zda požadavek přišel od uživatele, který byl již přihlášen, nebo zda se snaží podobný požadavek někdo jiný vytvořit. Pokud přišel požadavek ze třetí strany, která není autentizovaná, pak dokument z této stránky nebude mít správný token. Absence tokenu způsobí zneplatnění požadavku.

Tokeny pro identifikaci relace podléhají stejným pravidlům jako ostatní autentizační tokeny; musí být nepředvídatelné, nesmí být náchylné k uhodnutí hrubou silou atd. Problém nastává v okamžiku, kdy je aplikace náchylná i k XSS. Pomocí útoku XSS lze provést krádež relačního tokenu, a tak kompletně ochranu proti CSRF útoku obejít.

6.3 Man-in-the-Middle útok

Man-in-the-Midle (volně přeloženo jako člověk uprostřed, dále jen MITM) je opět velmi starý druh útoku. Principem je zachytávání komunikace mezi dvěma účastníky třetí stranou. Pokud aplikace neprovádí kontrolu integrity zasílaných dat a také neprovádí šifrování, pak je možné data měnit a číst třetí stranou. (Desmedt, 2005)

Tento útok je pomocí rozšíření webového prohlížeče velice snadno proveditelný, neboť prohlížeč nabízí rozhraní pro zachytávání odesílaných a přijatých dat (viz kapitola 7.3.1). V okamžiku zachytávání provozu mohou rozšíření začít odebírat bezpečnostní HTTP hlavičky a usnadnit tak práci útočníkovi v dalších fázích útoku.

6.4 Drive-by stahování

Drive-by stahování (drive-by download) představuje dle (Lee et al., 2013) techniku používanou pro distribuci škodlivého softwaru prostřednictvím zranitelností nacházejících se v samotném prohlížeči. Distribuce škodlivého softwaru může probíhat za použití vlastní stránky nebo zneužitím stránky cizí.

Způsob distribuce malware za použití drive-by stahování stojí na exploitaci (využití zranitelností) v samotném prohlížeči, jeho pluginech nebo operačním systému. Exploitace následně vede ke spuštění vlastního kódu útočníka na počítači oběti.

Jelikož drive-by stahování v zásadě zneužívá zranitelnosti na straně klienta nejen v samotném prohlížeči, pak je útočná plocha pro tento typ útoku enormní. Klient obvykle využívá velkého množství pluginů (viz kapitola 9.1) jako třeba Adobe Acrobat, Adobe Flash Player, Apple QuickTime, Microsoft ActiveX atd. Množství možných zranitelností se tak značným způsobem rozšiřuje

Tento útok je obvykle sestaven z několika po sobě jdoucích kroků. V první řadě musí být nějakým způsobem škodlivý obsah nahrán na veřejně dostupné místo. Útočník k tomu může zvolit dvě cesty. Buď k tomu využije vlastní server a bude se snažit oběť nalákat k návštěvě, anebo útočník ovládne již existující server za použití některé z jeho slabin. Po infikování daného serveru je potřeba přilákat oběti, čehož je docíleno například masovým rozesíláním spamu nebo zpráv na sociálních sítích. (Lee et al., 2013, s. 51)

Dalším krokem bude stažení škodlivého obsahu do prohlížeče oběti. Útočník musí v tomto okamžiku vyhodnotit velké množství informací o použitých verzích pluginů, operačním systému atd., aby mohl použít správný typ útoku. Škodlivý obsah je tak dodán až v okamžiku, kdy je navštívená stránka zobrazena prohlížečem s alespoň jednou zranitelnou komponentou.

Samotné zneužití zranitelnosti probíhá spuštěním škodlivého obsahu v rámci webového prohlížeče. V tomto okamžiku už dochází ke zpracování kódu útočníka na straně klienta a hlavního cíle celého procesu bylo dosaženo. Útočník může mít zadáno v tomto kódu například stahování dalších programů do počítače. Další škodlivá aktivita je tak považována za poslední krok celého procesu drive-by stahování. (Lee et al., 2013, s. 51)

S problematikou tohoto útoku je spojeno i téma exploitovacích souprav (anglicky exploit kits). Jedná se o komplexní software, jehož cílem je identifikovat zranitelnosti na straně klienta. Ty jsou porovnány vůči vlastní databázi existujících zranitelností. Pokud je nalezena shoda, pak je provedena samotná exploitace.

Tato problematika je spojena do jisté míry s rozšířeními. Rozšíření v prohlížeči může ovlivňovat to, jaké stránky se uživateli budou zobrazovat. Pokud

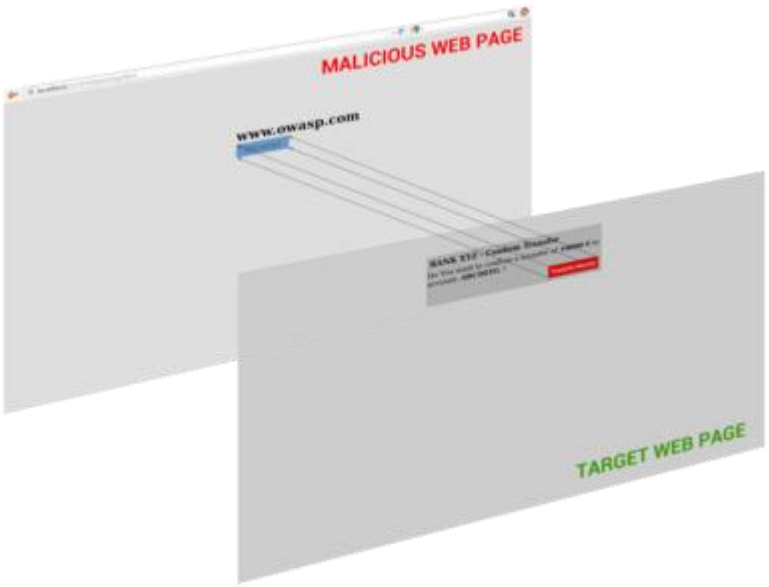
si útočník přeje, pak může oběť přesměrovat na stránky s exploitovací soupravou a provést útok.

6.5 UI Redress

UI redress je soubor technik, pomocí kterých útočník může podvrhnout požadavek oběti. Původcem požadavku je oběť, ta si ovšem není odeslání vůbec vědoma. Odeslání požadavku je totiž provedeno pomocí triku útočníka.

Nejjednodušší podoba tohoto útoku spočívá v načtení dokumentu, na kterém chce útočník provést činnost, do neviditelného rámce. Pokud se nastaví správně pozice rámce, pak lze zlákat oběť ke kliknutí na některý z elementů na stránce útočníka. Pokud dojde ke kliknutí, pak událost proběhne na neviditelném rámci, což povede k akci na webu třetí strany. Oběť tak ve své podstatě interaguje s jinou stránkou, než se na první pohled zdá.

Pro úplnost je na obrázku 4 vyobrazeno překrytí stránky zobrazené oběti stránkou, na níž má dojít k dané činnosti.



Obrázek 4 - Clickjacking a překrytí stránek, převzato z Testing for Clickjacking (OTG-CLIENT-009)

Tento útok je možný i v případě, že je implementována ochrana proti CSRF za použití dalších tokenů pro identifikaci sezení. Důvod pro fungování útoku i s použitím ochrany proti CSRF je ten, že veškerá komunikace probíhá legitimním způsobem.

Stránka je načtena do rámce v kontextu autentizovaného prohlížeče. Po kliknutí dochází k legitimnímu odeslání požadavku. Jediný rozdíl je, že uživatel si není neviditelného rámce vědom. V minulosti se jako ochrana proti tomuto útoku používala technika identifikace načtení do rámce (anglicky framebusting). Tyto postupy detekce byly mnohokrát obcházeny, a tak je v současnosti hlavním řešením tohoto problému použití hlavičiky `X-Frame-Options`, která načítání do rámce v prohlížeči zakazuje.

Tento útok je z hlediska rozšíření velmi snadno proveditelný. Rozšíření má přístup nejen k obsahu zobrazované stránky, ale také může odebírat bezpečnostní hlavičky.

7 Architektura systémů rozšíření

7.1 Chrome

Rozšíření si lze představit jako kompaktní aplikaci, jejímž účelem je nějakým způsobem rozšířit stávající funkcionalitu prohlížeče. Obvykle se jedná o balíček HTML, CSS a JS kódu spolu s dalšími webovými zdroji (například ikonkami). Balíček má formát s příponou „.crx“, ale jedná se ve své podstatě o zipový soubor. (CRX Package Format. Chrome developer, 2018)

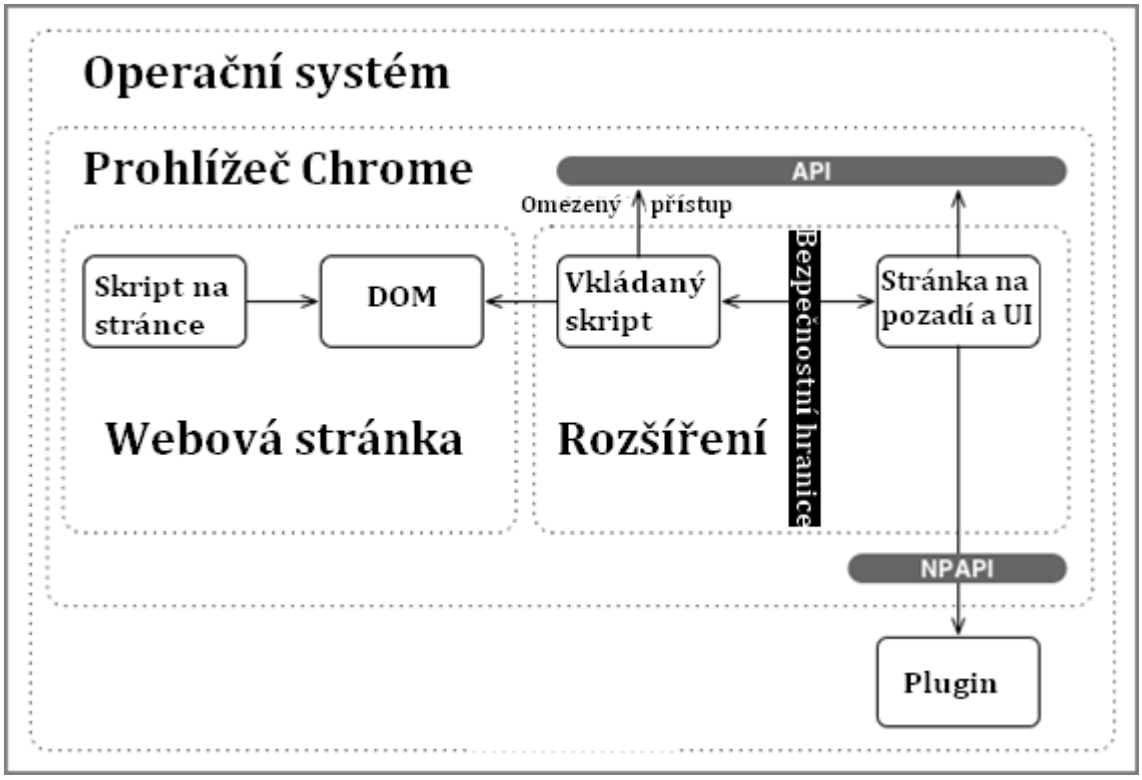
Hlavní kód rozšíření se obvykle nachází v tzv. background page (dále stránce na pozadí). Kód v této stránce má přístup k privilegovanému API prohlížeče, které umožňuje práci se záložkami, historií, cookies, odesílanými daty atd. Privilegované API se liší od obyčejného tím, že pro kód JS v obyčejném webovém dokumentu není přístupné.

Stránka na pozadí za jinak nezměněných podmínek pracuje po celou dobu, po kterou je prohlížeč spuštěn. Jelikož neustálý běh může být náročný na zdroje počítače, tak prohlížeč nabízí i alternativu. Tou je tzv. event page. Tato varianta stránky na pozadí má také přístup k privilegovanému API, ovšem je vyvolána jen v případě programově definované události. Pokud nebude nutné odlišení, pak se v práci dále používá spojení stránka na pozadí pro obě varianty.

Přístup k nabízeným funkcím prohlížeče je potřeba deklarovat v rozšíření prostřednictvím seznamu oprávnění. Seznam oprávnění je definován v tzv. manifestovém souboru (anglicky manifest file).

Další položkou v rozšíření mohou být tzv. content scripts (dále vkládané skripty), které se vkládají přímo do zobrazených webových stránek. Tam mohou upravovat a měnit DOM.

V neposlední řadě existují v rozšíření tzv. UI pages (grafické rozhraní). Cílem této komponenty je umožnit uživateli ovlivňovat chování rozšíření pomocí grafického rozhraní. Celkové schéma architektury prohlížeče s rozšířením je vyobrazeno na obrázku 5.



Obrázek 5 - Architektura rozšíření Chrome, převzato z (Alcorn, Frichot a ORRÙ, 2014, s. 321)

7.2 Komponenty

7.2.1 Stránka na pozadí

Stránka na pozadí má za úkol řídit a koordinovat jednotlivé složky rozšíření za použití JS kódu. Stránka může být aktivní trvale anebo může mít nastavenou direktivu `persistent` na hodnotu `false`. Tato direktiva určuje, že dojde k vyvolání stránky jen v případě některé JS události (tzv. event page).

Důvodem pro centralizaci do této komponenty je především přístup k privilegovanému API prohlížeče. Jedná se například o funkce pro práci s historií nebo zpracovávání HTTP hlaviček požadavků i odpovědí. Jiné typy komponent přístup k privilegovanému API mají omezený a JS kód webové stránky nemůže zpřístupnit toto API ani v omezeném rozsahu (alespoň v teorii, pokud rozšíření disponuje zranitelností, pak může v některých případech dojít ke zneužití a zpřístupnění nepřímo).

V souboru manifestu lze definovat libovolné množství stránek, které budou pracovat na pozadí prohlížeče. Každá stránka ovšem běží jen v jedné instanci

s výjimkou anonymního režimu, kdy za určitých okolností může dojít k rozdvojení. (Event Pages, 2018) (Background Pages, 2018)

7.2.2 Grafické rozhraní

UI stránky jsou grafickým rozhraním rozšíření. Jedná se o vyskakovací okna atd. Obvykle se prostřednictvím těchto stránek mění nastavení. Tyto stránky je potřeba definovat v rámci manifestového souboru.

Grafické rozhraní nemá přístup k DOM zobrazovaných webových stránek. Pokud by bylo potřeba nějakým způsobem strukturu DOM měnit, tak je nutné využít vkládaný skript (viz následující kapitola). S vkládaným skriptem je možné komunikovat prostřednictvím mechanismu zasílání zpráv. (Alcorn, Frichot a ORRÙ, 2014, s. 325)

7.2.3 Vkládaný skript

Vkládaný skript je komponenta rozšíření, která je načítána přímo do zobrazovaných webových stránek. Oproti UI stránkám nebo skriptu běžícímu na pozadí tak přímo pracuje s tím, co uživatel vidí. Jinak řečeno má přístup k DOM. To z tohoto skriptu tvoří lákavý cíl pro útočníky. (Alcorn, Frichot a ORRÙ, 2014, s. 324)

Další odlišnosti oproti jiným komponentám rozšíření je absence CSP (Content-Security-Policy). Praktickým dopadem této skutečnosti je, že nelze vypnout nebezpečnou funkci `eval()`, což dělá z této položky tu nejméně důvěryhodnou. Vkládaný skript má proto zredukovaný seznam možných funkcí, které může z celého API prohlížeče zpřístupnit.

Jedním z bezpečnostních principů architektury rozšíření je, že vkládaný skript sice může ovlivňovat vzhled stránky manipulací s DOM, ale nemůže volat JS funkce v dokumentu. Více vkládaných skriptů ani nemůže komunikovat za běžných okolností mezi sebou. Nelze si představit situaci, v níž by jeden vkládaný skript byl knihovnou pro ostatní vkládané skripty, protože ty nebudou moci nabízené funkce přímo zavolat. (Content Scripts, 2018)

Vkládaný skript si lze představit více jako součást stránky než jako součást rozšíření. Se zbytkem rozšíření, což je obvykle stránka na pozadí, může komunikovat jen pomocí k tomu určeného API pro zasílání zpráv.

Na druhou stranu může být z vkládaného skriptu vyvolán `XMLHttpRequest` požadavek na doménu stránky, v níž je skript vnořen. Režim, ve kterém může skript pracovat s DOM, ale nemůže zpřístupňovat funkce a proměnné mimo vlastní zdrojový kód, se nazývá izolovaný svět (isolated world). Komunikace v takovém režimu může se stránkou probíhat třeba za pomocí úpravy DOM, kdy jedna komponenta zapisuje a druhá čte. (Content Scripts, 2018)

7.2.4 Manifestový soubor

Manifestový soubor, označen manifest.json, je soubor ve formátu JSON, který se musí nacházet vždy v kořenovém adresáři daného rozšíření a který definuje přístupová práva, jednotlivé použité skripty, vzhled ikon a mnoho dalšího. Jedná se o první přečtený dokument při načítání rozšíření.

7.3 Bezpečnostní aspekty

7.3.1 Chrome API

Pro vývoj rozšíření nabízí prohlížeč Chrome velké množství funkcí v rámci svého API pro práci s historií, záložkami, stahovanými soubory atd. V současnosti se API dělí na několik skupin: stabilní, beta, vyvíjené a experimentální funkce. V této práci je dále uveden seznam funkcí zajímavých z hlediska praktické části:

- `chrome.bookmarks` slouží k přístupu k záložkám v rámci prohlížeče. V rámci tohoto API lze záložky zpřístupnit, tvořit i mazat. Z hlediska bezpečnosti je toto API zajímavé především pro možnost zcizit seznam důležitých stránek oběti. Oprávnění pro toto API je `bookmarks`.
- `chrome.cookies` je API pro práci s cookies v rámci prohlížeče. V manifestového souboru lze definovat, jakých stránek se toto oprávnění k přístupu ke cookies týká. Oprávnění je `cookies`.
- `chrome.desktopCapture` slouží k zachycení obrazovky uživatele. Oprávnění je `desktopCapture`.
- `chrome.downloads` umožňuje začít, sledovat a upravovat proces stahování souborů. I toto API má svoje vlastní oprávnění a tím je `downloads`.

- `chrome.extensions` umožňuje výměnu zpráv mezi vkládaným skriptem a stránkou na pozadí.
- `chrome.history` slouží, jak název napovídá, k práci s historií. Používá oprávnění `history`.
- `chrome.i18n` umožňuje použití více jazyků.
- `chrome.management` slouží k evidenci nainstalovaných rozšíření v prohlížeči. Používá oprávnění `management`.
- `chrome.proxy` umožňuje nastavit proxy server pro prohlížeč a daný protokol. Oprávnění používá `proxy`.
- `chrome.storage` slouží k práci s lokálním uložištěm v prohlížeči. Oprávnění je `storage`.
- `chrome.tabs` umožňuje práci s jednotlivými otevřenými stránkami. Pro většinu práce na otevřených dokumentech není potřeba deklarovat oprávnění, pro zbytek se používá `tabs`.
- `chrome.webRequest` se užívá pro práci se síťovým provozem. Lze jej použít pro modifikaci a čtení. Oprávnění je `webRequest`.
- `nativeMessaging` umožňuje rozšíření komunikovat s nativní aplikací v počítači uživatele. Oprávnění je v tomto případě stejnojmenné.

Každé z tohoto API je spojeno s určitým oprávněním v rámci manifestového souboru, které se musí explicitně uvést. Výše zmíněná oprávnění se zadávají do manifestového souboru pomocí pole s názvem `permissions`. Například použití API pro práci se záložkami by se v manifestovém souboru definovalo pomocí zápisu v následujícím formátu:

```
{
  ...
  "permissions" : [
    "bookmarks"
  ],
  ...
}
```

Výše uvedené API se týká především stránky na pozadí. Vkládané skripty mají přístup k většině funkcí zakázán i přes uvedení definice do manifestového souboru. Jedinou výjimku tvoří následující API:

- `chrome.extension`,
- `chrome.i18n`,
- `chrome.runtime`,
- `chrome.storage`.

Toto ovšem není pro škodlivé rozšíření velké omezení, neboť lze z vkládaného skriptu zpřístupnit privilegované API nepřímo za pomoci zasílání zpráv. Jedná se tak pouze o omezení pro neškodlivé rozšíření, které by mohlo obsahovat zranitelnost. (Declare Permissions, 2018)

7.3.2 Izolované světy

Pro každé rozšíření platí, že v rámci manifestového souboru je definováno, které soubory budou načítány jako stránky na pozadí, jaké jako vkládaných skript a jaké budou sloužit jako grafické rozhraní.

Tyto tři složky spolu mohou komunikovat a také zpřístupňovat části chrome.* API. Komunikace ovšem probíhá různými způsoby podle toho, jaké komponenty spolu komunikují. Kupříkladu stránka na pozadí a grafické rozhraní sdílí stejnou paměť a tak mohou spolu komunikovat přímo. Grafické rozhraní může provést následující kód:

```
var bg = chrome.extension.getBackgroundPage();
bg.console.log('Zaslano z popup!');
```

a tím vyvolá zápis v konzoli u stránky na pozadí.

Vkládaný skript má jiné vlastnosti. Především se stránkou na pozadí nesdílí paměť, takže tyto dvě komponenty spolu komunikují pouze pomocí speciálního rozhraní `chrome.extension.sendRequest`. Toto komunikační rozhraní ovšem nenabízí možnost zasílat reference na objekty. Jelikož nelze zasílat reference na

objekty, tak nelze z vkládaného skriptu volat funkce ze stránky na pozadí a obráceně.

Vkládaný skript je také oddělen od dokumentu, do něhož je vložen. Je tomu tak především z toho důvodu, že pokud by mohl skript na stránce sdílet prostředí se vkládaným skriptem, tak by mohl volat jeho funkce. To by mohlo například vést k volání `chrome.extension.sendRequest` a tím až k ovlivňování stránky na pozadí. (Alcorn, Frichot a ORRÙ, 2014, s. 327-328)

7.3.3 Injektáž skriptu do stránky

Vzhledem k výskytu vkládaného skriptu v tzv. izolovaném světě, je otázkou, zda lze některým způsobem zpřístupnit objekty stránky nepřímo.

Izolované světy existují především jako obrana proti přístupu z JS kódu v dokumentu stránky k privilegovanému API, ke kterému má přístup vkládaný skript a případně stránka na pozadí. Z druhé strany (ze strany vkládaného skriptu do dokumentu stránky) ovšem lze docílit získání přístupu pomocí injektáže. Vložit vlastní kód lze pomocí (Content Scripts, 2018):

- vložení souboru,
- vložení skriptové značky,
- vložení funkce,
- vložení pomocí atributu elementu (`onclick`,…).

7.3.4 CSP a rozšíření

Otázka, odkud mohou být zdroje načítány do stránky v rámci rozšíření (skripty, …), vedla k zavedení CSP (viz kapitola 5.2). Od manifestového souboru verze dvě je CSP zavedeno do všech rozšíření implicitně. Rozšíření si ovšem může explicitně tuto konfiguraci přenastavit.

CSP bylo v rámci rozšíření zavedeno také jako obrana proti XSS. XSS může totiž velmi snadno vzniknout i v rámci této komponenty prohlížeče. Vkládaný skript má přístup k DOM stránky a zároveň může komunikovat se stránkou na pozadí, která má přístup k vysoce privilegovanému API.

Zatímco pro dokumenty načtené v prohlížeči platí, že CSP nastavené je definováné buď pomocí HTTP hlaviček anebo pomocí meta elementu v hlavičce dokumentu, tak pro rozšíření se definuje CSP do manifestového souboru pomocí páru `"content_security_policy": "hodnota"`.

Od manifestového souboru verze 2 je implicitní nastavení CSP pro rozšíření: `script-src 'self'; object-src 'self'`.

Tato konfigurace má několik dopadů na fungování. V první řadě není možné spouštět kód z funkce `eval()` v rámci stránky na pozadí ani grafického rozhraní (vkládaný skript na druhou stranu není nastavením CSP ovlivněn vůbec). Toto nastavené je implicitní, protože tato funkce je velmi častým zdrojem XSS zranitelností. Obdobně nelze načítat skripty z externích zdrojů.

Dalším místem, kde nebude možné spouštět skript, jsou tagy `<script>` v dokumentu grafického rozhraní. Vkládání vstupu od uživatele do těchto značek se mnohokrát ukázalo jako nebezpečné i přes použití filtrování. (Alcorn, Frichot a ORRÙ, 2014, s. 330)

Načítání skriptů je možné pouze v případě, že se nachází na stejném zdroji jako dokument (tedy v rámci balíčku rozšíření). Externí skripty nabízejí útočníkovi příliš velké možnosti útoku. Také není možné žádným způsobem načíst externí objekty typu Flash, Java apod.

Zvolnění CSP lze dosáhnout explicitním uvedením hodnoty pro direktivu `content_security_policy`. Implicitní nastavení lze tedy přepsat explicitně za použití vlastní CSP direktivy v manifestovém souboru. Ač tento postup není doporučovaný, tak mnoho knihoven spoléhá na využití funkce `eval()`, čehož jsou si vývojáři vědomi. Ti mnohdy preferují novou funkcionalitu před bezpečností. (Alcorn, Frichot a ORRÙ, 2014, s. 331)

Speciální problematiku tvoří CSP v případě vkládaného skriptu. Jelikož vkládaný skript není HTML soubor, ale pouze skript, pak nelze v jeho případě definovat stejná pravidla pro načítání externích zdrojů. Vkládaný skript může obsahovat funkci `eval()`, která bude fungovat nezávisle na nastavení CSP pro dané rozšíření.

Složitá situace nastane v případě vložení elementů (konkrétně `<script>`) vkládaným skriptem do HTML dokumentu, který má vlastní striktní CSP politiku. Jelikož se CSP nevztahuje na vkládaný skript, pak některé situace vedou ke spuštění kódu nezávisle na CSP stránky a některé situace naopak nevedou k žádnému spuštění. Následující příklad

```
document.write("<script>alert(1);</script>");
```

povede ke spuštění nezávisle na politice stránky, do které je skript vkládán. Naopak, následující kód

```
document.write("<button onclick='alert(1);'>Ok</button>'");
```

v případě kliknutí na vložené tlačítko nepovede ke spuštění skriptu. Dochází totiž ke vložení elementu, který patří stránce a ta má zakázané vložené skriptování. (Content Security Policy (CSP), 2018)

7.3.5 Mezidoménové XMLHttpRequest požadavky

`XMLHttpRequest` požadavek je v případě webových stránek omezen SOP. Za jinak nezměněných podmínek nelze vytvořit požadavek pomocí `XMLHttpRequest` na server s jiným původem (schéma, adresa a port), než je zdrojový dokument. Toto lze změnit použitím těchto HTTP hlaviček na straně serveru:

- `Access-Control-Allow-Origin`,
- `Access-Control-Allow-Methods`,
- `Access-Control-Allow-Headers`.

V případě přijetí těchto hlaviček prohlížeč mezidoménové požadavky na server povolí.

Rozšíření podléhají jiným pravidlům než webové dokumenty. Samotný původ rozšíření je definován jen na dokumenty nacházející se v něm. Lze vytvořit například příslušný `XMLHttpRequest` na zdroj v kořenovém adresáři dokumentu s názvem `config.json`:

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = handleStateChange;
xhr.open("GET",chrome.extension.getURL('/config_resources/config
.json'), true);
xhr.send();
```

Požadavky na zdroje umístěné jinde budou automaticky ignorovány, pokud nejsou povoleny v `permissions` v manifestovém souboru. V rámci manifestového souboru může být definován přístup na servery Google použitím:

```
"permissions": [
  "http://www.google.com/",
  "https://www.google.com/"
]
```

Z hlediska bezpečnosti velmi specifické obecné definice domén typu
`"http://*/" ,`
které umožňují definovat zasílání požadavků na libovolnou adresu URL. (Cross-Origin XMLHttpRequest, 2018)

7.4 Firefox

Architektura rozšíření prohlížeče Firefox se v současnosti značně přibližuje té, která je použitá v prohlížeči Chrome. Vývojáři Firefoxu přijali za úkol vytvořit takový systém, ve kterém lze nainstalovat rozšíření použitelná i v prohlížeči Chrome. V listopadu roku 2017 Firefox přejal architekturu WebExtensions a ukončil tak podporu architektury XUL, která se v minulosti potýkala s bezpečnostními problémy. Firefox v rámci svého interního prostředí definoval objekt `chrome`. (What are extensions?, 2018)

7.4.1 Architektura

Firefox od konce roku 2017 přejal architekturu WebExtensions používanou v prohlížeči Chrome. Rozšíření v prohlížeči Firefox musí obsahovat soubor `manifest.json`, který určuje, jaká stránka bude pracovat na pozadí, která bude na zobrazovaných webových stránkách a také co bude tvořit grafické rozhraní.

Spolu s obdobnou logikou skriptu na pozadí (přístup k privilegovanému API prohlížeče) má i tento skript podobná omezení. Jedním z nich je implicitně nastavená politika CSP, která zabraňuje fungování funkce `eval()`.

Stránky tvořící uživatelské rozhraní představují způsob, jakým uživatel může nastavovat a měnit atributy rozšíření. V rámci rozšíření lze definovat vyskakovací okno, stránku s nastavením, boční panel atd.

V rámci stránek s uživatelským rozhraním je možné volat stejné privilegované API jako u stránek na pozadí. Je také možné v rámci UI stránek získat přístup ke stránce na pozadí pomocí `runtime.getBackgroundPage()`.

V rámci rozšíření je možné vytvářet vlastní stránky s přístupem k privilegovanému API prostřednictvím `windows.create() a tabs.create()`. Tyto stránky budou mít přístup ke stejnému API jako stránky na pozadí.

Pro vkládaný skript platí stejná pravidla v prohlížečích Firefox i Chrome. V rámci tohoto skriptu lze vytvářet `XMLHttpRequest` požadavky, využívat některá privilegovaná API a také komunikovat se stránkou na pozadí. Se skripty na stránce lze komunikovat pomocí mechanismu zasílání zpráv. (Anatomy of an extension, 2018)

8 Útoky proti rozšířením

Rozšíření přináší další kód do prohlížeče a zvětšují tak možný prostor pro nalezení zranitelnosti. Prostřednictvím útoku na rozšíření tak může nastat například XSS zranitelnost, a to i v případě, že samotná webová aplikace takovou zranitelnost nemá. Důsledky takové situace závisí na tom, čeho chce útočník dosáhnout a jak je zranitelnost kritická.

Pro úspěšný útok proti doplňkům je potřeba dvou kroků. V prvním kroku by mělo dojít k tzv. značkování (fingerprinting) rozšíření. Útočník by měl identifikovat, jaká rozšíření se na straně klienta používají. Problém pro útočníka je, že prohlížeč nenabízí API v rámci HTML dokumentu pro identifikaci a je potřeba identifikovat rozšíření pomocí různých triků.

V druhém kroku útočník využije znalosti o nainstalovaných rozšířeních, a pokud existuje zranitelnost v některém z nich, pak ji využije.

8.1 Identifikace rozšíření

8.1.1 XHOUND

Problematice detekování nainstalovaných rozšíření bylo v minulosti věnováno několik prací. Jednou z prací je (Starov a Nikiforakis, 2017). V ní autoři demonstrují, že lze prostřednictvím detekce nainstalovaných rozšíření jednoznačně identifikovat uživatele, a to i přes to, že neexistuje jednoznačné API, které by umožňovalo získat seznam již nainstalovaných.

Detekce probíhá prostřednictvím frameworku XHOUND, který sleduje chování nainstalovaných rozšíření. V rámci detekce se sledují úpravy na DOM struktuře autory vytvořených stránek. Ukázalo se, že identifikace prostřednictvím rozšíření představuje relativně přesný nástroj pro deanonymizaci uživatelů.

Pro identifikaci rozšíření ovšem nestačí pouze sledovat úpravy na DOM struktuře. Rozšíření může například být aktivní pouze v případě návštěvy některé z hojně navštěvovaných domén (Facebook, Yahoo, Youtube). Lze tak vytvořit dvě kategorie; rozšíření závislá na URL a rozšíření nezávislá (toto odlišení funkcionality může být definováno nejen v souboru `manifest.json`, ale také v samotném zdrojovém kódu). (Starov a Nikiforakis, 2017, s. 3)

Identifikace některých rozšíření probíhá nezávisle na URL. Může se jednat o správce hesel na stránkách. Jedním reprezentantem této kategorie je LastPass. Toto rozšíření vytváří u HTML elementů pro hesla tlačítko umožňující uložení pro budoucí použití a to nezávisle na stránce, kterou si uživatel zobrazuje. Obdobně i blokování reklam (AdBlock) funguje nezávisle na URL.

Na druhou stranu nejvíce populárních rozšíření obvykle cílí na některé z populárních stránek (ať už se jedná o Facebook, Twitter, Youtube atd.). Jedná se o rozšíření závislá na URL. Například rozšíření s názvem Twitter zobrazuje tlačítko na stejnojmenné stránce. Tato skutečnost ovšem umožňuje vývojářům této stránky detekovat existenci tohoto rozšíření právě testováním existence tlačítka. V minulosti se ukázalo, že také například stránky Skype.com, Whitehouse.gov atd. své uživatele aktivně sledují a identifikují. Sledování uživatelů je prováděno právě i prostřednictvím identifikace nainstalovaných rozšíření.

Pokud na straně útočníka existuje seznam DOM úprav, které jsou unikátní pro dané rozšíření, pak je možné sestavit takovou stránku, která se bude tyto akce snažit vyvolávat. Na základě změn bude možné provést identifikaci.

XHOUND framework implementuje dvě možné cesty identifikace změn DOM. První z nich jsou falešné stránky. Jedná se o dynamicky generované stránky, které tvoří prvky na základě jejich vyhledávání v rámci JS. Pokud dojde kupříkladu k volání

```
document.getElementById("id") ,
```

pak dojde k vytvoření tohoto prvku na stránce. Ač není tento přístup kompletně bez chyby, tak tímto způsobem je možné vysledovat velké množství úprav DOM.

Druhou možností jsou tzv. statické stránky. Tyto stránky mají svůj obsah pevně daný. Stránka může obsahovat odkazy na sledovací skripty třetích stran, reklamy atd. Tyto prvky mohou aktivovat rozšíření, které by jinak nebylo možné detekovat. Činností takových rozšíření může být třeba blokování reklam.

V práci (Starov a Nikiforakis, 2017) bylo ukázáno, že k relativně úspěšné identifikaci je postačující mít přehled o unikátních úpravách, které dané rozšíření vykonává. Praktický výzkum v této oblasti ukázal, že velké množství rozšíření provádí unikátní akce na DOM.

8.1.2 Alternativní způsoby identifikace

Kromě výše zmíněného způsobu identifikace nainstalovaných rozšíření pomocí úprav na DOM lze použít i alternativní způsoby. V roce 2017 byla vydána odborná práce (Sanchez-Rola a Santos, 2017), která se zabývala identifikací nainstalovaných rozšíření prostřednictvím testování toho, jaké soubory jsou v rozšíření veřejně dostupné.

Chrome i Firefox nabízejí v současnosti možnost pro dané rozšíření specifikovat, které soubory budou veřejně dostupné z webového dokumentu. Veřejně dostupné soubory konkrétního rozšíření lze zobrazit pomocí odkazu `chrome-extension://[extID]/[path]` .
Pokud není žádný ze zdrojů jako veřejně přístupný nastaven, pak se automaticky považují všechny soubory za privátní. Tomuto mechanismu v prohlížečích se říká ACS systém (Access Control Settings). (Sanchez-Rola a Santos, 2017, s. 679-680)

ACS je definován v souboru `manifest.json`. Příkladem nastavení veřejně dostupných zdrojů je:

```
"web_accessible_resources": [
      "images/*.png",
      "style/double-rainbow.css",
      "script/double-rainbow.js",
      "script/main.js",
      "templates/*"
]
```

Ukázalo se, že tento obranný mechanismus je náchylný pro útok postranním kanálem . V tomto případě se jednalo o časový útok.

Časový útok v tomto případě souvisí s mechanismem ověřování existence souboru. Pokud soubor existuje, ale není veřejný, pak testování trvá déle, než když soubor v daném rozšíření neexistuje. Tento časový rozdíl je způsoben tím, že při požadavku na zdroj je potřeba na straně prohlížeče nejprve zjistit, zda dané rozšíření a soubor existuje a následně pak to, zda je daný soubor definován jako externě dostupný v jeho manifestovén `manifest.json`. Pokud soubor neexistuje,

pak je proveden pouze test existence souboru. (Sanchez-Rola a Santos, 2017, s. 680-683)

8.2 Zneužití zranitelnosti

Po úspěšné identifikaci rozšíření může dojít k samotnému útoku. Celkem existuje několik variant zneužití zranitelností.

8.2.1 Cross Context Scripting

Cross Context Scripting (skriptování napříč kontexty, dále CCS) je útočná technika, která způsobuje přenesení instrukcí útočníka z méně privilegovaného kontextu do více privilegovaného. Konkrétně v případě rozšíření jde o proces, v němž se určité webové stránce povede vložit svůj kód do privilegované zóny (např. skript na pozadí). Pokud rozšíření disponuje touto zranitelností, pak může útočník získat přístup k privilegovanému API na straně klienta.

Ač prioritou útoku bude především vkládaný skript, neboť ten nemá žádné omezení definované CSP, tak i stránka na pozadí může představovat zajímavý cíl. Stránka na pozadí může mít například špatně nastavenou politiku CSP. Zranitelnost může vzniknout i ze způsobu, jakým jsou data přenášena od vkládaného skriptu směrem ke skriptu na pozadí. Pokud jsou neošetřená data dále zpracovávána a samotný vkládaný skript disponuje zranitelností, pak může útočník nepřímo získat přístup k privilegovanému API. (Alcorn, Frichot a ORRÙ, 2014, s. 339)

8.2.2 MITM

Rozšíření ke své činnosti potřebují v určitých situacích načítat obsah z některého místa na internetu. Může se jednat o aktualizaci seznamu URL, ze kterých jsou načítány škodlivé skripty atp. Problém může nastat v případě, že stahování ze serveru probíhá jen prostřednictvím HTTP protokolu (tedy nešifrovaným způsobem). Pokud jsou takto přenášená data zasílána do funkce `eval()`, pak lze provést CCS útok prostřednictvím MiTM. (Alcorn, Frichot a ORRÙ, 2014, s. 340-341)

8.2.3 Obejití CSP

Vkládaný skript představuje pro útočníka obvykle velmi lákavý cíl. Je tomu tak především kvůli absenci CSP a také přímému přístupu k DOM zobrazované webové stránky. Pokud se vyskytne zranitelnost v rámci vkládaného skriptu, může dojít k provedení útoku podobnému klasickému útoku XSS.

Mimo defaultní vlastnost vkládaných skriptů lze využít i konkrétních chyb vývojářů. Pokud se vývojáři rozhodnou stanovit CSP pro rozšíření tak, aby skript na pozadí umožňoval pracovat s funkcí `eval()`, pak při neošetřeném vstupu od uživatele do této funkce může vzniknout zranitelnost. (Alcorn, Frichot a ORRÙ, 2014, s. 344)

Největším problém s oběma výše uvedenými způsoby ignorování CSP je ten, že tímto způsobem dochází k obcházení politiky stanovené webovou aplikací. I přes to, že samotná aplikace nemusí mít XSS zranitelnost, pak při použití rozšíření může na klientovi tato zranitelnost vzniknout. (Alcorn, Frichot a ORRÙ, 2014, s. 346)

8.2.4 Obejití SOP

SOP (viz kapitola 5.1) představuje základní princip webové bezpečnosti. Stojí na relativně jednoduchém principu, který říká, že dokumenty mající jiný původ by se neměly být schopny navzájem ovlivňovat. Ukazuje se ovšem, že v některých případech mohou rozšíření způsobit narušení tohoto bezpečnostního mechanismu.

Vkládaný skript umožňuje zasílat `XMLHttpRequest` požadavky na domény bez ohledu na původ dokumentu (pokud je tak umožněno v rámci manifestového souboru). Tyto požadavky mohou obsahovat cookies, takže může docházet tímto způsobem ke krádeži autentizačních tokenů. (Alcorn, Frichot a ORRÙ, 2014, s. 347)

Obsahový skript pracuje s DOM stránky. Pokud dochází k nesprávné práci s touto strukturou v kontextu rozšíření, pak je možné provést útok podobný DOM XSS. V rámci DOM XSS dochází ke změně struktury stránky takovým způsobem, aby došlo k úpravě chování skriptu. Pokud takové rozšíření implementuje funkci `eval()` pro práci s daty pocházejícími z DOM, tak může dojít až k CCS. Zranitelné rozšíření pak v takovém případě může na pokyn útočníka uskutečnit autentizovaný požadavek na určitou stránku, stáhnout důvěrná data a následně je odeslat na útočníkovu doménu. (Alcorn, Frichot a ORRÙ, 2014, s. 348-350)

9 Útoky proti pluginům

9.1 Pluginy

Funkcí pluginů v rámci prohlížeče je prostřednictvím svého rozhraní zpřístupnit kód třetí aplikace, aby jej bylo možné využít k provádění činností, které samotný prohlížeč nenabízí. Jedná se tak ve své podstatě o spojovací prvek mezi prohlížečem a jinou aplikací. Samotný proces instalace pluginu způsobuje přidání kódu do prohlížeče, jenž toto spojení umožní.

Strukturně lze rozložit plugin na dva funkční podsystémy. Jedná se o prohlížečové API a skriptovací API. API pro prohlížeč umožňuje komunikaci mezi prohlížečem a externí aplikací. Druhý podsystém, skriptovací API, nabízí rozhraní pro manipulaci s konkrétním objektem (například pdf soubor). Toto rozhraní je zpřístupněno obvykle pomocí JS.

V minulosti byl seznam používaných rozšíření napříč spektrem prohlížečů obsáhlý. Jmenovitě docházelo k častému užívání pluginů jako Flash, Acrobat, Javu, QuickTime, Silverlight, RealPlayer a VLC plugin. (Alcorn, Frichot a ORRÙ, 2014, s. 371)

V tuto chvíli je vhodné rozlišit pluginy a rozšíření, neboť tyto komponenty fungují odlišným způsobem. Zatímco rozšíření implementuje svou funkcionalitou pomocí JS a API samotného prohlížeče, pak plugin využívá konkrétní externí aplikace (směrem k celému prohlížeči). Rozšíření je navíc obvykle zpřístupněno na různých stránkách a případně na pozadí, zatímco plugin je obvykle spojen s určitým typem souboru, který server zasílá klientovi.

Jinak řečeno plugin je vyvolán až v okamžiku, kdy prohlížeč přijme určitý typ souboru. Soubor může být ve stránce zpřístupněn použitím objektu `<object>`, `<embed>` nebo zaslán s definovaným MIME typem (např. `aplication/pdf`). (Alcorn, Frichot a ORRÙ, 2014, s. 372-373)

Na druhou stranu plugin volá funkcionalitu třetí aplikace, ale nejedná se ovšem o aplikaci samotnou. Funkcionalita může být zmenšena oproti původní aplikaci a může dojít k potřebě otevřít dokument externě mimo prohlížeč. Platí

ovšem, že velmi často je nalezená zranitelnost společná jak pro aplikaci, tak pro plugin. (Alcorn, Frichot a ORRÙ, 2014, s. 374)

Ač v jádru jsou pluginy efektivní způsob rozšíření stávající funkcionality, v minulosti byly častým cílem útoků. Po opakovaných bezpečnostních incidentech implementovali vývojáři funkcionalitu click to play (neboli kliknout ke spuštění). Tato funkcionalita má zamezit automatickému spouštění pluginů bez vědomí uživatele a ztížit tak jejich exploitaci. Pro spuštění pluginu je třeba dát explicitní souhlas uživatele.

Oproti rozšířením je detekování nainstalovaných pluginů snazší. Sám prohlížeč nabízí API pro identifikaci nainstalovaných verzí.

9.2 Exploitace

Exploitaci pluginů, pro jejich snížený význam v současnosti, bude věnován jen krátký přehled. V prvním kroku obvykle dochází k analýze použitých pluginů na straně klienta. Identifikace se prování pomocí stránky útočníka, kterou oběť navštíví. Na této stránce se může nacházet exploitovací souprava (viz kapitola 6.4). Pokud se povede identifikovat zranitelný plugin (resp. jeho zranitelná verze), pak přichází na řadu exploitace.

9.3 Současný stav

Tato kapitola je zde pro úplnost, protože tématika bezpečnosti pluginů je s bezpečností prohlížečů nerozlučně spojena. V současnosti existuje ovšem trend minimalizovat vliv pluginů na prohlížeče (především ze strany prohlížečů Chrome a Firefox).

Prohlížeč Chrome (verze 63) nepodporuje značné množství pluginů a to právě především kvůli bezpečnosti. Stejně tak i prohlížeč Firefox vliv pluginů značně omezil. Odstoupení od této technologie se projevilo i na končící podpoře Javy v jednotlivých prohlížečích.

Z celkového hlediska je užívání pluginů na ústupu. Flash přestane být podporován samotnou firmou Adobe do roku 2020. Pro oba prohlížeče (Chrome i Firefox) je již nyní definován vlastní interní zobrazovač PDF souborů. (Java plug-in does not work in Firefox after installing Java, 2018)

10 Útoky za použití rozšíření

Tak, jako je možné napadnout zranitelné rozšíření a tím kompromitovat klienta, je možné využít možností nabízených výskytem privilegovaného API a vytvořit vlastní nebezpečné rozšíření. Škodlivá rozšíření představují velmi složitou a aktuální problematiku, protože s tímto fenoménem je spojeno značné množství bezpečnostních incidentů. Proto se vývojáři prohlížečů Firefox a Chrome uchýlili k omezení možných způsobů instalací rozšíření. Pro Chrome platí, že je před instalací potřeba nahrát zdrojové kódy na veřejné místo Chrome Web Store, kde je kód automaticky validován a zjišťována jeho škodlivost. Obdobně musí projít kontrolou i rozšíření v rámci prohlížeče Firefox.

10.1 Botnet

V této kapitole bylo čerpáno z publikace (Perrotta a Feng, 2017). Rozšíření se ukazuje být velmi zajímavým cílem pro útočníky. Na druhou stranu tato komponenta prohlížeče nabízí i vhodný způsob distribuce škodlivého obsahu. To je důsledek dvou aspektů spojených s doplňky. Prvním z nich je ohromný přesah. Rozšíření může mít až miliony uživatelů, kteří si jej můžou stáhnout a aktivně používat. Druhým aspektem je, že doplňky disponují mnoha funkcionalitami, které mohou být snadno zneužitelné. Jedná se především o přístup k privilegovanému API (historii, cookies atd.).

Rozšíření lze zneužít kupříkladu k tvorbě botnetu. Jedná se ve své podstatě o síť počítačů, které dostávají příkazy od určité autority (v tomto případě útočníka). Příkazy mohou obsahovat instrukce k provádění DOS útoků, infekci dalších počítačů atd.

Ač oproti minulosti jsou možnosti instalace škodlivého rozšíření omezené (nyní musí být rozšíření předem přidáno do Chrome Web Store, je potřeba vstupní poplatek atd.), tak stále existují možnosti distribuce škodlivého obsahu. Distribuce může proběhnout spolu s instalací trojského koně, kterého oběť stáhla. Také může dojít k donucení oběti k instalaci prostřednictvím technik sociálního inženýrství a v neposlední řadě útočníci kupují již existující populární rozšíření a přidávají k nim svůj škodlivý kód.

Problematika identifikace škodlivých rozšíření je odvozená od použitého prohlížeče. Například ve Firefoxu dochází k ručnímu procházení nových doplňků, zatímco v rámci Web Store je proces testování automatizovaný. Ač oba tyto způsoby mají své prokazatelné výsledky, tak přesto nejsou kompletně spolehlivé. Autoři tvrdí, že úspěšnost odhalení škodlivého obsahu ve Web Store je přinejlepším cca 96.5%. V práci (Perrotta a Feng, 2017, s. 4) se dokonce autorům povedlo svůj vlastní škodlivý doplněk na tržiště dostat.

Škodlivé chování se projevuje mnoha způsoby. Může docházet ke změně DOM stránky. To je způsobeno tím, že vkládaný skript má podstatě neomezený přístup k DOM. Lze snadno způsobit překrytí stránky jinou stránkou (viz kapitola 6.5). Stejně tak může docházet k přesměrování přenosu ze zašifrovaného kanálu na nezašifrovaný. Mimo tyto aspekty lze prostřednictvím rozšíření provést detekci zařízení, krást údaje o poloze, zcizit cookies, měnit HTTP hlavičky (kapitola 5.2) atd.

11 Praktická část

11.1 Architektura

V této části je popsána architektura detekčního systému Tali. Jedná se o systém, neboť v sobě zakomponována několik samostatných aplikací, které spolu navzájem komunikují. Vzhledem k největšímu zastoupení Google Chrome na trhu se systém Tali specializuje právě na tento prohlížeč.

Systém detekce je postaven na třech komponentách, které spolu navzájem komunikují. První komponentou je webové rozšíření v samotném prohlížeči. Tato komponenta má za úkol sledovat ostatní nainstalovaná rozšíření a informovat uživatele o tom, zda je některé z nich podezřelé . Rozšíření v rámci systému Tali je napsáno primárně pro prohlížeč Google Chrome. Do budoucna je možné tento prvek systému přepsat i pro ostatní prohlížeče. Například prohlížeč Firefox přijal na konci architekturu Web Extensions přítomnou v Google Chrome (viz kapitola 7.4), I přes tuto skutečnost v současnosti stále neexistuje úplná vzájemná kompatibilita.

Druhou komponentou je jednoduchý program napsaný v programovacím jazyce Java, který má za úkol zprostředkovat komunikaci mezi rozšířením v prohlížeči a jádrem systému. Výměna informací v této komponentě probíhá prostřednictvím JSON zpráv. Mimo zprostředkování komunikace nemá tento prvek žádnou jinou zodpovědnost.

Třetím a nejdůležitějším prvkem je samotné jádro aplikace. Jedná se o program napsaný opět v jazyce Java, který má několik částí a plní několik úkolů. Kromě samotné komunikace s rozšířením v prohlížeči funguje aplikace jako proxy server, který sleduje odchozí a příchozí provoz. Dále tato aplikace implementuje prvky statické i dynamické analýzy.

Programovací jazyk Java byl zvolen pro snadnou přenositelnost systému mezi operačním systémem Windows A Linux.

11.2 Získání trénovacích dat

Za účelem získání atributů podezřelých rozšíření bylo potřeba stáhnout ukázky jejich zdrojových kódů.

Správně fungující rozšíření lze snadno získat přímo v repositáři Chrome Web Store. Oproti tomu příklady škodlivých rozšíření nelze získat takto přímočaře, neboť Google ve svém Web Store promazává veškerý škodlivý obsah bez jakéhokoliv zálohování. Na internetu též nebyl nalezen žádný veřejný repositář s ukázkami škodlivého kódu. Z těchto příčin vznikla potřeba vytvořit vlastní repositář

Za účelem tvorby vlastního repositáře byla v prvním kroku nalezena stránka, která pravidelně archivuje veškerá rozšíření, jenž se v minulosti na Web Store nacházela. Tato stránka nabízí možnost si stáhnout mimo existující i již neexistující verze rozšíření odebraných z Web Store. Z této stránky bylo staženo celkem 400 náhodně vybraných položek. Ty položky, které se nenacházely na Web Store, byly podrobeny detailní analýze. Analýza chování proběhla ovšem i u rozšíření, která se v oficiálním repositáři nacházela. Důvodem pro toto rozhodnutí byla potřeba jistoty, že se škodlivé rozšíření nedostane do nesprávné kategorie.

11.2.1 Chrome Web Store

Chrome Web Store slouží jako repositář pro všechna rozšíření, která lze oficiálně nainstalovat do prohlížečů. Vzhledem k dlouhé historii zranitelností, ale i výskytu škodlivých rozšíření, je možné nainstalovat položku pouze v případě, že se nachází na Web Store. V rámci usnadnění práce vývojářům prohlížeč Chrome nabízí také možnost zapnout speciální režim, který načítání neoficiálních rozšíření umožňuje. Tento režim znesnadňuje možnost vmanipulovat oběť k načtení škodlivého software třetí strany.

Limitace instalace pouze na rozšíření, která jsou na Web Store, je zavedena především protože se Google snaží pomocí různých prostředků identifikovat, zda nedochází ke škodlivé činnosti. (AGGARWAL et al., 2017, s. 2)

11.3 Ruční klasifikace

U jednotlivých rozšíření, která nebyla nalezena v rámci Web Store, bylo potřeba zjistit, zda mají podezřelé atributy. Prvním indikátorem se ukázal být samotný název a následně také popisek v rámci manifestového souboru. Pokud rozšíření obsahovalo názvy populárních her a slovních spojení typu „Free games“ atp., pak se

jednalo ve staženém vzorku vždy o škodlivou kategorii. Ukázalo se též, že velmi průkazný je výskyt spojení „Candy Crush“. Pokud neslo rozšíření toto označení, pak vykazovalo vždy stejné rysy a jednalo se o druh adware (kód, jehož cílem je libovolným způsobem zobrazit reklamu či stránku autora rozšíření).

Autoři adware rozšíření leckdy sahali až ke „spamovému popisování“. Příkladem takového popisu je (rozšíření názvem Sniper Team HD):

"full_description": "When the going gets tough, there is only one team to join: the Sniper Team.\nHow to Play :\nSoldier, your base needs defending! Select your sniper and your weapon and get ready to defend your turf at all costs in this shooting game. Clear out invaders to pass a level, but remember to keep an eye on your defenses: they can only hold out for so long... Mouse = Aim & Fire Space = Zoom In/Out 1-4 = Switch Sniper Q = Switch Weapon R = Reloadcandy crush saga- clash of clans- temple run 2 games- subway surfers games- minecraft- gta v games- gta vice city- angry birds games- jetpack joyride games- fifa 2015 games- candy crush games- asphalt 8 games- minecraft games-the sims 3 games- temple run 2 games-fifa 16-clash royale\ntalking tom games- gta san andreas games- angry birds rio- Plants vs- Zombies games-angry birds star wars 2- gta games - angry birds rio games- angry birds epic games - gta 5 games - fifa 2014 games - pou games- minion rush games-monopoly games- flappy bird games-modern combat games-agar-io-minecraft pocket edition-minecraft pc\n\ncontinue to provide you with the most beautiful games in google chrome. Target shooting games sniper team is now one of the most beautiful you. After setting up your own team giriyosunuz a fierce struggle with the enemy. The enemy could be anywhere and do just what you need to trust your teammates. Sniper team game is a multiplayer game. Players will work with your team to kill their enemy soldiers. You are trying to kill you, they kill them. 2013 which is selected from the most beautiful war games sniper team oyunur be played online. If you want to play the game, you can add our application online sniper team.\n\nSome of the other games!:\n\nAccurate boy games, American soldier games, candy crush saga games, car racing games, earn to die 2013 games, fast and furious games, Formula Racer 2013 games, gta v games, gta vice city games, handless millionaire games, la rex games, renegade racing games, spider man 3 games, subway surfers games, tractor mania games, Disaster Will Strike 2\n\nFor more you can follow our profile."

Rozšíření sice nese název Sniper Team HD, ale autor neváhal do popisku přidat značné množství dalších populárních klíčových slov. Tato slova ovšem vůbec nesouvisí se samotným názvem. Jedná se o názvy populárních her jako GTA, Minecraft atd. a přítomnost takového množství názvu působí do jisté míry jako spam. Po prohlédnutí zdrojového kódu se potvrdilo, že rozšíření vykazuje rysy adware.

Dalším zajímavým indikátorem jsou výskyty přesmyček názvů. Pravděpodobně za účelem vyhýbání se detekci zvolili autoři jednoduchou taktiku použít přesmyčku nebo grafickou podobnost k jinak známému slovu. Velmi oblíbené slovo Minecraft bývá přepsáno na M|necraft, Alternativou bylo použití Cendiy Crush Saga namísto správného tvaru Candy Crush Saga. Pokud rozšíření bylo pojmenováno takovým způsobem, pak se vždy jednalo o škodlivý software.

Kromě názvu a popisku nabízí manifestový soubor další indikátory podezřelé činnosti. Z kapitoly 7.3.4 je zřejmé, že rozšíření samo o sobě disponuje relativně striktní politikou CSP omezující spouštění skriptů z internetu, použití funkce `eval()` atd. Jakékoliv explicitní přepsání CSP je potřeba podrobit analýze, zda je z hlediska funkcionality daného rozšíření potřeba. V některých případech je zvolnění CSP potřeba pro zavedení Google Analytics (legitimní způsob sledování způsobu využívání daného rozšíření uživatelem). V jiných případech se jedná o jasné bezpečnostní riziko. Velmi zajímavým příkladem podezřele nastavené CSP je rozšíření s neutrálním názvem Flowers (ID agicicjohejcncbjmgbhjggoelgjclad). CSP je v tomto případě definováno následovně:

```
"content_security_policy": "script-src 'self' http://03tracks.tk
https://counter.yadro.ru 'unsafe-eval'; object-src 'self';"
```

Toto nastavení CSP umožňuje nejen načítat skripty z externích zdrojů (z toho jedna doména nese velmi podezřelý název `tracks`), ale také umožňuje zavolat funkci `eval()`, viz část '`unsafe-eval`'. Výskyt povolení `eval()` v nastavení CSP značí jednoznačně velmi podezřelé chování. S přihlédnutím k názvu Flowers by se tak široce definované CSP nemělo v rozšíření vyskytovat.

Mimo samotné nastavení CSP je velmi důležité analyzovat i oprávnění, která dané rozšíření vyžaduje. Vzhledem k přístupu k velmi privilegovanému API se jedná o důležitý indikátor podezřelé činnosti. Ukázka z minulého odstavce s názvem Flowers je ideální demonstrací příliš velkého množství přístupu k funkcionalitě prohlížeče. Rozšíření s tímto názvem má totiž povolení k mnoha úkonům, které značně přesahují předpokládanou funkcionalitu (zobrazování obrázků květin). Definice oprávnění v manifestovém souboru je následující:

```
"permissions": [
    "tabs",
    "storage",
    "cookies",
    "*://*/*",
    "management",
```

```
    "clipboardRead",
    "clipboardWrite",
    "https://*/*",
    "http://*/*",
    "<all_urls>",
    "webRequest",
    "webRequestBlocking",
    "webNavigation"
]
```

Takto velmi široce definované nastavení umožňuje rozšíření obcházet hned několik bezpečnostních principů najednou. Kombinace `http://*/*`, `https://*/*`, `<all_urls>`, `*://*/*` umožňuje zasílat `XMLHttpRequest` požadavky na libovolnou doménu a komunikovat s libovolným zařízením na internetu. Toto oprávnění ve spojení s `unsafe-eval` nabízí útočníkovi možnost stahovat a spouštět nebezpečné skripty na straně klienta prakticky bez omezení. Jelikož tímto způsobem lze provést spouštění skriptů z jiných domén na navštívených webových stránkách, tak dochází k narušení SOP (kapitola 5.1). Vzhledem k přítomnosti oprávnění cookies je i možné, že se rozšíření pomocí `XMLHttpRequest` pokusí zcizit autentizační token.

Oprávnění `webRequest` a `WebRequestBlocking` umožnují analyzovat a měnit odchozí a příchozí provoz prohlížeče. Velmi snadno lze takovým způsobem odebírat bezpečnostní hlavičky (viz kapitola 5.2). Odebráním `Strict-Transport-Security` hlavičky lze dosáhnout přesměrováváním veškerého provozu ze zašifrovaného kanálu na nezašifrovaný (čímž lze pak snáze odposlouchávat komunikaci oběti). Dalším útokem může být odebírání `X-Frame-Options`, která zabraňuje útoku clickjacking (kapitola 6.5). V neposlední řadě může docházet k odebírání `Content-Security-Policy`, čímž se zvyšuje šance na úspěšný XSS útok (kapitola 6.1) v případě existující zranitelnosti na navštívené webové stránce.

Po prozkoumání oprávnění lze v manifestového souboru analyzovat vkládané skripty. Rozšíření z minulého odstavce disponuje následujícím nastavením:

```
"content_scripts": [
    {
        "js": [
            "jquery-1.11.3.min.js",
            "js.js"
        ],
        "matches": [
            "*://*/*"
        ]
    }
]
```

Je zřejmé, že rozšíření umožňuje vkládat skript `js.js` do všech navštívených stránek. Tento indikátor je ovšem sám o sobě bez ostatních ukazatelů velmi nejednoznačný, protože mnoho důvěryhodných rozšíření pracuje na všech navštívených stránkách. Jedná se tak pouze o doplňující informaci. Též název vkládaného skriptu sám o sobě nemívá velkou výpovědní hodnotu, neboť útočník může snadno pojmenovat soubor neutrálním způsobem.

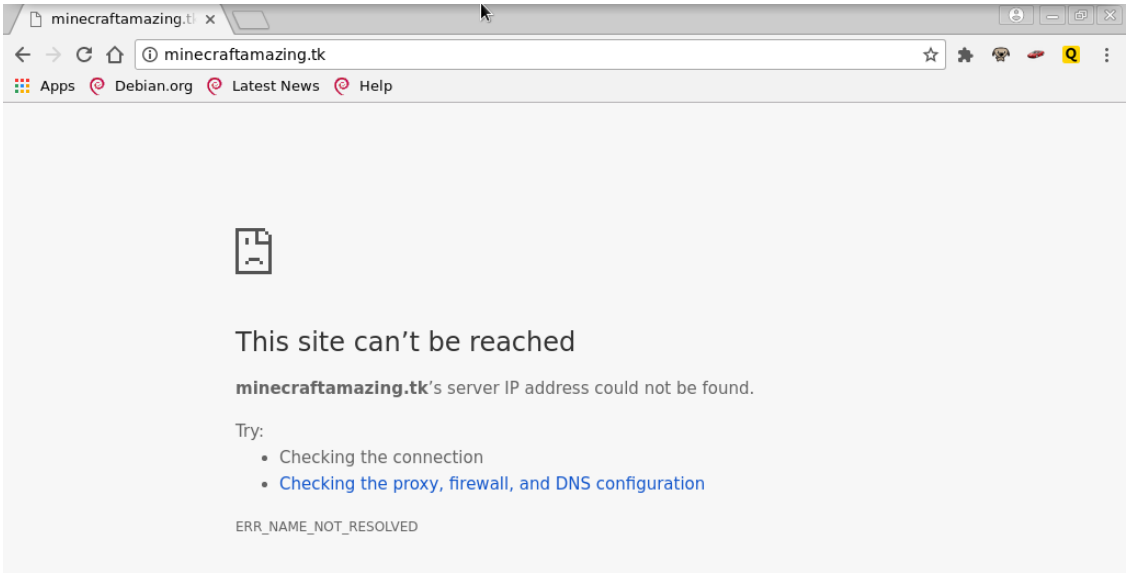
Po analýze manifestového souboru lze přejít k analýze zdrojových kódů. Pro účely této práce lze vycházet z (Jagpal et al., 2015, s. 586). Hned na první pohled při analýze libovolného zdrojového kódu je patrné, zda se jej autor pokusil učinit záměrně nečitelným tak, aby nebyla zřejmá na první pohled jeho funkcionalita (tzv. obfuskovat). Automatická detekce obfuskace je téma nad rámec této práce a analýze takového kódu nebude věnována přílišná pozornost. Pro úplnost lze ovšem uvést, že záměrně nečitelný kód lze podrobit dynamické analýze, ve které se sleduje samotné chování daného skriptu po spuštění. Od této chvíle bude v této práci uvažován jen kód, který je čitelný a lze v něm jednotlivá volání funkcí vypozorovat.

Hned první významným aspektem každého zdrojového kódu jsou v něm použité URL adresy. Bylo vypozorováno, že některá škodlivá rozšíření z kategorie adware měla velmi často opakující se vzorce v URL adresách. Jedním z typických znaků byl výskyt řetězce Minecraft. Vlastním zjištěním byly nalezeny tyto reference:

```
http://www.minecraftnewserie.ml
http://www.minecraftrealgold.gq
http://minecraftamazing.tk
http://www.popularhdgames.tk
http://softwarearea.net
http://www.minecraftblockgta.cf
http://speed-dial.ru/
http://www.minecraftallpc.ml
http://zungl.my1.ru
http://www.minecraftpcmode.cf
```

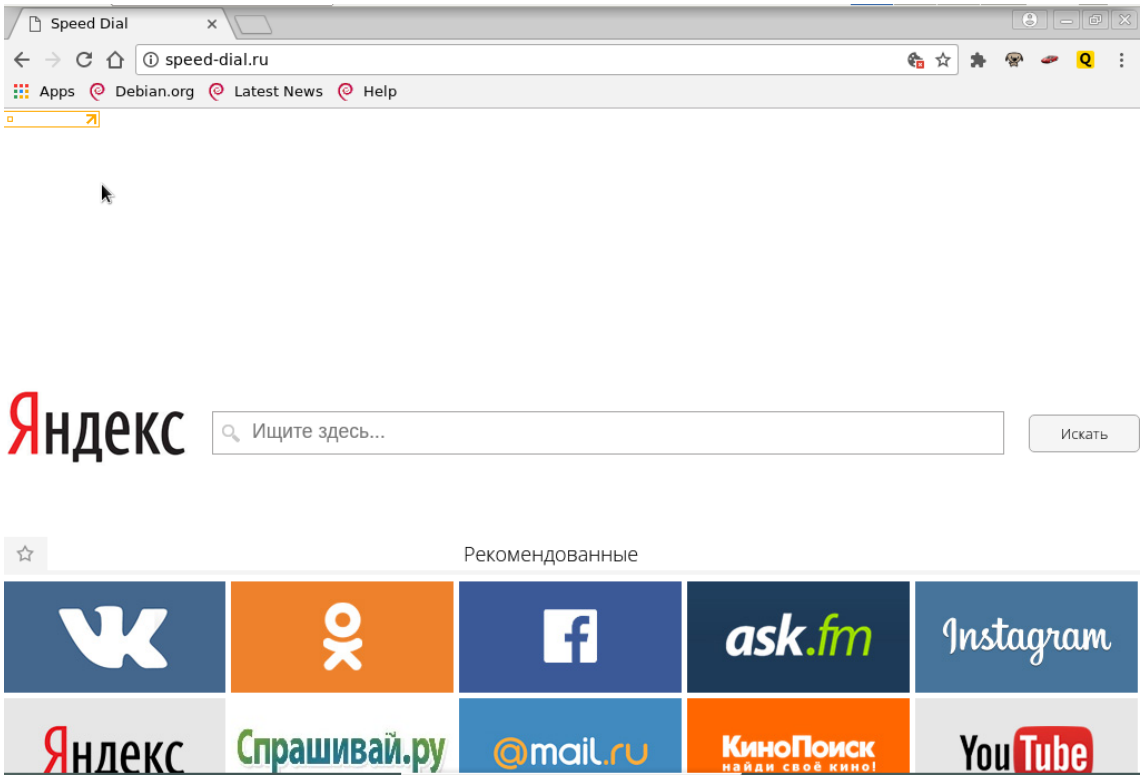
Vzhledem k malému vzorku zkoumaných rozšíření oproti všem nacházejícím se ve webovém repositáři Web Store nelze říci, jak velký podíl na škodlivých URL adresách mají URL adresy s řetězcem Minecraft. V rámci vybraného vzorku se jednalo ovšem o jednoznačný indikátor škodlivosti. Škodlivostí se v tomto případě myslí především vytváření vyskakovacích oken s adresou stránek obsahujících reklamy nebo závadný obsah. Lze si tak také všimnout, že tyto URL adresy mají spíše méně časté domény prvního řádu.

Při zobrazení některých z těchto stránek již daný server v okamžiku psaní práce nefungoval. Adresa `http://minecraftamazing.tk` odkazuje na neexistující server (viz obrázek 6).



Obrázek 6 - Nefunkční URL adresa

Na druhou stranu některé URL adresy odkazují na funkční server, ale s nejasným obsahem. Například `http://speed-dial.ru` odkazuje na stránku vyobrazenou na obrázku číslo 7.



Obrázek 7 - Podezřelá URL adresa

Otázkou zůstává, zda toto byl původní obsah. Je také možné, že se v tuto chvíli daná stránka pouze snaží působit důvěryhodně.

Ukázalo se také, že výskyt výše uvedených domén je často spojen s výskytem této množiny API:

- `chrome.runtime.onInstalled,`
- `chrome.runtime.onlaunched,`
- `chrome.runtime.onInstalled,`
- `chrome.app.runtime.onLaunched,`
- `chrome.app.runtime.onRestarted,`
- `chrome.runtime.setUninstallURL,…`

Adware toto API využívá k registraci funkcí pro tvorbu vyskakovacích oken se stránkami odkazujícími na uvedené škodlivé domény.

Některé z těchto stránek mohou být u antivirů zaregistrované jako škodlivé. Škodlivé URL adresy z rozšíření ovšem velmi často nejsou antiviry zaregistrovány vůbec.

Dalším reprezentantem kategorie adware je rozšíření s ID *ablnlophalgghmgodebpaeocjpelnlhp* a názvem Internet Download Manager. Toto rozšíření v sobě obsahuje tuto stránku na pozadí:

```
chrome.runtime.onInstalled.addListener(function(details) {
    if (details.reason == 'install') {
        chrome.browser.openTab({
            url: url.replace('{term}', 'first-launch')
        });
    }
});
chrome.app.runtime.onLaunched.addListener(function() {
    chrome.browser.openTab({
        url: url.replace('{ter  m}', 'app-launcher')
    });
});
chrome.app.runtime.onRestarted.addListener(function() {
    chrome.browser.openTab({
        url: url.replace('{term}', 'app-restarted')
    });
});

chrome.alarms.onAlarm.addListener(function(alarm) {
    chrome.browser.openTab({
        url: ('http://www.proappsnet.com')
    });
    chrome.alarms.clear("check-ext-exists");
});
if (chrome.runtime.setUninstallURL) {
    chrome.runtime.setUninstallURL('http://www.proappsnet.com');
```

```
} else {}
```

Tento zdrojový kód demonstruje úmysl vytvářet vyskakovací okno při největším možném množství akci spojených s instalací, spuštěním a odinstalací rozšíření.

Kroky popsané ve výše uvedených odstavcích mohou i přes občasnou nejednoznačnost relativně dobře poukázat na škodlivost. V případě samotných zdrojových kódů už neexistují tak jednoznačná vodítka (průkazná jako podezřele definované nastavení CSP v rámci manifestového souboru). Například výskyt obfuskace sice může značit snahu skrýt podezřelé chování, tato úprava zdrojového kódu ovšem má i legitimní použití. U každého z jednotlivých sledovaných atributů lze tak pouze určit pravděpodobnost, že se bude vyskytovat u škodlivého rozšíření. Nejinak tomu je i v případě relativně často zneužívaných funkcí jako `eval()`, `XMLHttpRequest`, `runtime.onInstalled` a mnoho dalších. V práci (Jagpal et al., 2015, s. 586) je například uvedeno, že pouze modifikace CSP hlavičky, snaha o zabránění odinstalace rozšíření a odinstalace ostatních rozšíření nainstalovaných v prohlížeči jsou velmi spolehlivým indikátorem škodlivosti. U ostatních atributů ovšem tato spolehlivost neplatí.

Zvláštní případ tvoří skripty, u kterých je zřejmá snaha skrývat podezřelá volání funkcí ve svém zdrojovém kódu pomocí primitivní obfuskace. Příkladem takového postupu je rozšíření s ID *aepiigcppinhepcligpmdafgddiaenln* a s názvem 7 Deadliest Weapons in History. Ve zdrojovém kódu se nachází funkce s názvy

- `magicImportanceIfCertainlyFat()`,
- `enterModelDegreeDugIncome()`,
- `ratherSunDevelopBurnRising()`,

které nekorespondují s žádnou činností. Ovšem na řádku 194 se nachází kód, který poukazuje na snahu skrýt funkcionalitu před automatickými detektory.

```
var aa = "XMLH",
    ab = "ttpRe",
    ac = "quest",
    ad = aa + ab + ac;
var xhr = new window[ad]();
```

```
xhr.open(type, url, true);
xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
        if (xhr.status === 200 || xhr.status === 204) {
            resolve(xhr.responseText);
        } else {
            reject(Error(xhr.statusText));
        }
    }
};
```

Výše uvedený kód zneužívá dynamické povahy JS (kapitola 4.5), tedy že k interpretaci dochází až v okamžiku zpracovávání kódu. Tato vlastnost umožňuje definovat `XMLHttpRequest` objekt až v okamžiku zpracovávání na základě odkázání na atribut globálního objektu `window`.

```
ad = aa + ab + ac;
var xhr = new window[ad](); // v ad  je řetězec „XMLHttpRequest“
```

Tato výše uvedená jednoznačně prokazatelná snaha ukrýt výskyt definice tohoto objektu je jasnou známkou podezřelého chování.

Výše uvedený přehled slouží pouze jako jednoduchá demonstrace postupu identifikace škodlivých rozšíření pomocí metod statické analýzy. Otázka identifikace vlastností škodlivých zdrojových kódů je značně komplexní a v mnoha ohledech vyžaduje i pokročilou analýzy chování, což je téma svou komplexností přesahující rozsah této práce.

11.4 Automatická klasifikace

Z předchozí kapitoly je zřejmé, že neexistuje jednoznačný identifikátor, pomocí kterého by bylo možné rozhodnout o škodlivosti. O té rozhoduje až kombinace více atributů. Z tohoto důvodu byl pro účely klasifikace využit tzv. klasifikátor. Klasifikátor představuje určitý algoritmus, který pro daný vstup dokáže rozhodnout, do jaké kategorie patří. V rámci této práce byl použit naivní bayesovský klasifikátor.

Naivní bayesovský klasifikátor byl pro účely této práce vybrán hned z několika důvodů. Těmito důvody jsou snadná implementace, mnohokrát prokázaná schopnost klasifikovat textové dokumenty, častá aplikace pro účely tzv. spamového filtrování (což představuje jistou obdobu problému klasifikace zdrojového kódu) a také to, že pro natrénování tohoto klasifikátoru není potřeba příliš velkého vzorku.

Pro získání trénovacích dat bylo náhodně staženo 400 rozšíření. Tyto vzorky byly podrobeny ruční analýze (jednalo se o 500 souborů zdrojových kódů). Pro usnadnění procesu trénování klasifikátoru se za každé sledované rozšíření vybral právě jeden soubor, který obsahoval buď škodlivou složku, nebo naopak skript, který nejlépe demonstroval bezpečné chování. Tato ruční filtrace proběhla především kvůli přítomnosti knihoven typu JQuery, které by mohly značným způsobem vychýlit četnosti sledovaných atributů (knihovny v sobě obvykle obsahují značné množství funkcí, ale knihovna není sama o sobě škodlivá). Následně také došlo k eliminaci rozšíření, která měla pouze vyskakovací HTML okna a žádný skript. HTML stránky byly odebrány, protože u nich se sledují jiné atributy než u JS souborů a bylo by potřeba použít klasifikátoru, který by sledoval jiné znaky.

Po eliminaci nevhodných skriptů a také rozšíření, která by nebyla klasifikátorem analyzovatelná, vznikl vzorek o velikost 130 zdrojových kódů. U těchto zdrojových kódů byla následně stanovena četnost sledovaných atributů a tyto četnosti byly použity pro trénování.

11.5 Dynamická analýza

Aplikace pro doplnění statického pohledu na detekci zahrnuje i analýzu odchozího provozu (především navštívené URL). Tato funkcionalita je pouze demonstrativní, jedná se o jisté doplnění komplexního systému detekce škodlivosti. Tato komponenta byla přidána do aplikace především jako odpověď na otázku, jak detekovat škodlivou činnost u silně nečitelného zdrojového kódu, který je typický při obfuskaci.

Škodlivé domény byly získány ručním procházením škodlivých rozšíření. Vzhledem k relativně malému vzorku analyzovaných rozšíření jich není příliš mnoho. Pro plnohodnotnou funkcionalitu v této oblasti by bylo vhodné zakomponovat obsáhlý seznam škodlivých domén. Vzhledem k absenci veřejného

repositáře škodlivých rozšíření a také toho, že nebylo možné použít veřejné seznamy škodlivých domén (domény z rozšíření se v nich nenacházely), bylo vhodné vytvořit si vlastní černou listinu. Doménové názvy nebyly nalezeny na stránkách MalwareDomainList.com a ani na stránce VirusTotal, což jen potvrzuje potřebu vytvořit si vlastní seznam.

11.6 Linuxové prostředí

Vzhledem ke složité instalaci systému Tali bylo rozhodnuto o demonstraci této aplikace v rámci operačního systému Linux, který je distribuován prostřednictvím virtuálního počítače.

Důvodů pro tuto formu distribuce bylo hned několik. Především odpadají problémy spojené s ruční instalací systému. Pro operační systém Windows je potřeba, aby proběhla konfigurace registrů. V registrech se zapisuje adresa souboru se specifikací aplikace, se kterou bude prohlížeč komunikovat. Jedná se o relativně složitý proces. V rámci systému Linux stačí pouze vložit registrační soubor do složky prohlížeče. Dalším problémem by mohlo být, že pro potřeby testování klasifikace se pracuje s příklady škodlivých nebo alespoň podezřelých rozšíření. Za účelem snížení rizika pro uživatele se přistoupilo k použití virtualizace.

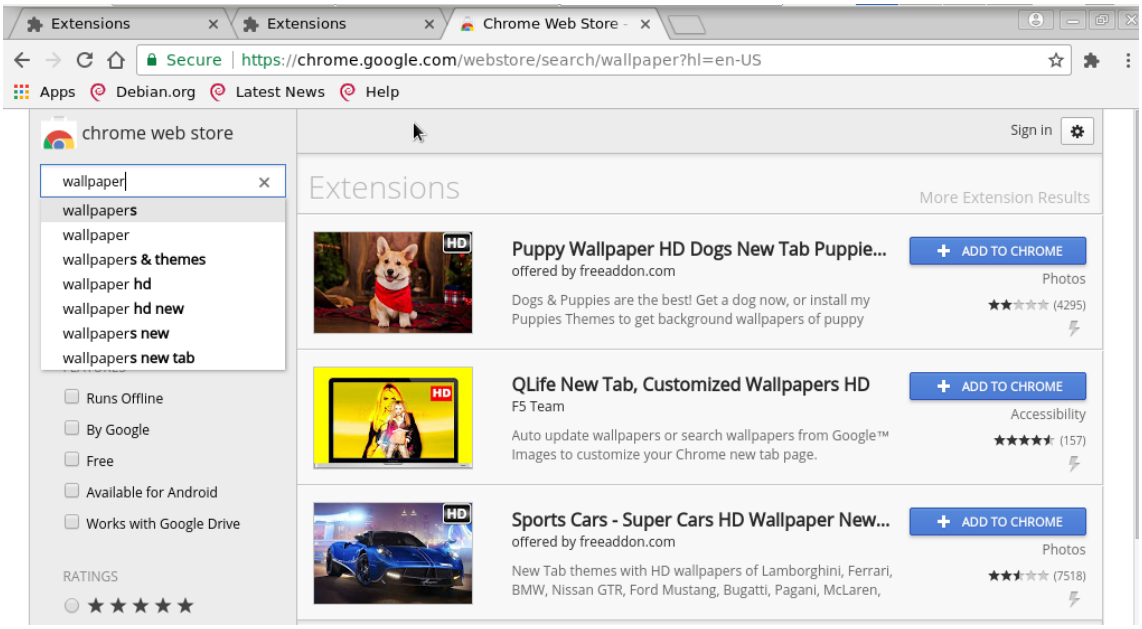
Pro načítání systému Tali je také potřeba připravit samotný prohlížeč pro práci v tzv. vývojářském módu. Tento mód zajišťuje, že lze načíst rozšíření, která se nenachází na oficiálním Web Store.

11.7 Systém Tali

Po ruční klasifikaci, trénování a instalaci do operačního systému Linux lze otestovat samotný systém pro detekci. Tento systém byl navržen tak, aby bylo možné testovat nainstalovaná rozšíření za běhu prohlížeče bez potřeby konfigurace a opětovného spouštění. Po spuštění stačí pouze najít rozšíření v rámci Web Store, nainstalovat jej a následně kliknout na tlačítko v prohlížeči pro zobrazení výsledků analýzy.

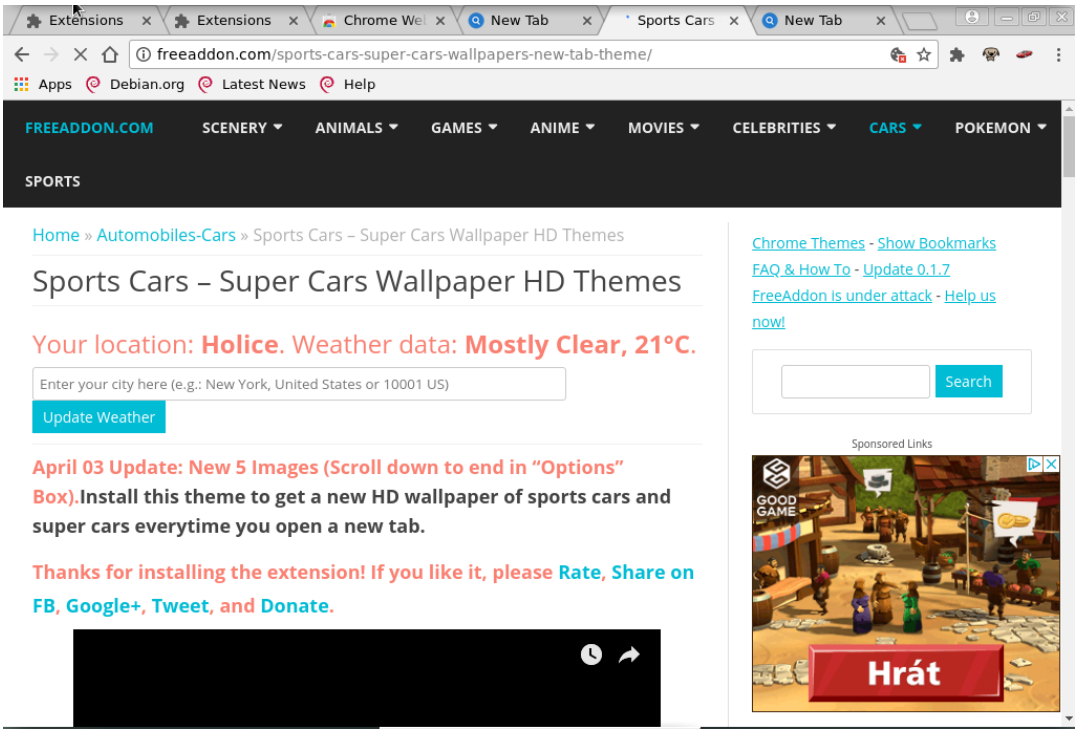
Pro demonstraci funkcionality byla vybrána tři rozšíření z Web Store, která v sobě nesou název Wallpaper. Během ruční klasifikace bylo zjištěno, že rozšíření s tímto názvem často měla příliš široce definovaná oprávnění a docházelo v nich ke zcizování autentizačních tokenů.

Na následujícím obrázku číslo 8 lze vidět konkrétní exempláře, které budou nainstalované.



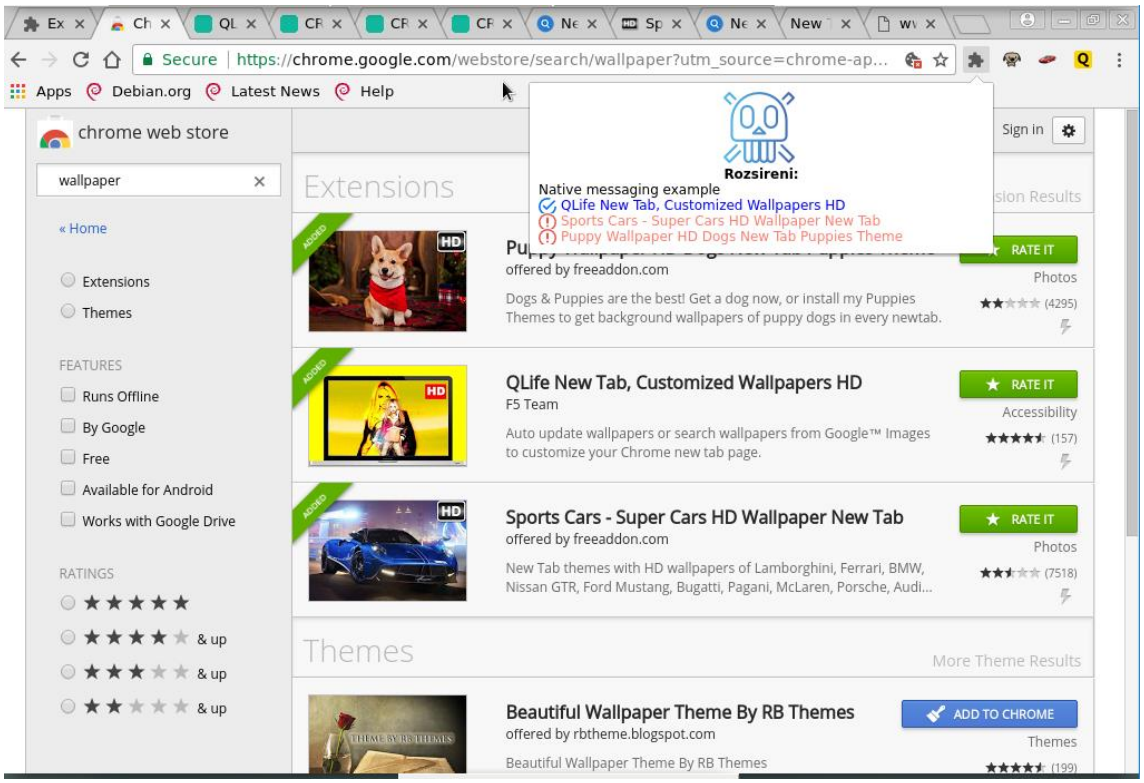
Obrázek 8 - Instalovaná rozšíření

Na první pohled je zřejmé, že první a třetí rozšíření má relativně velké množství špatných hodnocení. Po instalaci těchto dvou nepříliš pozitivně hodnocených rozšíření se zobrazí vyskakovací okno viditelné na obrázku číslo 9.



Obrázek 9 - Vyskakovací okno po instalaci rozšíření

Tato stránka nepůsobí příliš seriózním dojmem. To samo o sobě ovšem ke klasifikaci nestačí. Při analýze vypíše Tali upozornění viditelné na obrázku 10.



Obrázek 10 - Analýza rozšíření

Po instalaci těchto tří rozšíření identifikoval systém ta hůře hodnocená jako podezřelá a lépe hodnocené rozšíření jako v pořádku. Pro úplnost budou podrobeny analýze manifestové soubory.

Manifestové soubory negativně hodnocených rozšíření vypadají velice podobně. To lze vysvětlit i tím, že oba doplňky pochází od stejného vývojáře freeaddon. V obou manifestových souborech je definována velice široká množina oprávnění (viz obrázek 11).

```
"permissions": [
    "<all_urls>",
    "tabs",
    "storage",
    "unlimitedStorage",
    "cookies",
    "contextMenus",
    "notifications",
    "webRequest",
    "webRequestBlocking",
    "webNavigation",
    "management",
    "topSites"
],
```

Obrázek 11 - Manifestový soubor podezřelých rozšíření

Lze si všimnout, že obě rozšíření vyžadují přístup nejen ke cookies, ale také mohou analyzovat příchozí a odchozí provoz a sledovat ostatní nainstalovaná rozšíření v prohlížeči (oprávnění `management`). Tyto skutečnosti jsou velice podezřelé u doplňků s deklarovanou funkcionalitou měnit vzhled prohlížeče.

Naopak pozitivně hodnocené rozšíření QLife má ve svém manifestovém souboru bezpečnější sérii oprávnění (obrázek 12).

```
"permissions": [
    "<all_urls>",
    "tabs",
    "activeTab",
    "geolocation",
    "unlimitedStorage",
    "alarms",
    "topSites",
    "chrome://favicon/"
],
```

Obrázek 12 - Oprávnění pozitivně hodnoceného rozšíření

Tento soubor oprávnění lze uvažovat jako relativně bezpečný. Otázkou zůstává, zda by mělo mít rozšíření přístup k fyzické lokaci uživatele. Na tuto otázku nelze ovšem jednoznačně odpovědět.

12 Shrnutí výsledků

Práce podrobuje analýze problematiku škodlivých rozšíření. I přes skutečnost, že Web Store nenabízí možnost stáhnout rozšíření klasifikována jako škodlivá, byl vytvořen vlastní seznam zástupců této kategorie. Aplikace byly získány z veřejně dostupného archívu, který zálohuje rozšíření z Web Store. Při ruční analýze zdrojových kódů bylo zjištěno několik skutečností.

Škodlivá rozšíření lze rozdělit do několika různých kategorií. Nejčastější variantou byl tzv. adware. Velmi často se jednalo o jednoduchý kód, který po instalaci způsoboval vytváření vyskakovacích oken. Dalším reprezentantem byla rozšíření, která prokazatelně zcizovala autentizační tokeny a velmi pravděpodobně i historii. U těchto složitějších rozšíření docházelo i k odebírání bezpečnostních hlaviček. Většinou se tato rozšíření snažila nalákat uživatele ke stažení nabízením některých funkcionalit (například nové obrázky do prohlížeče). Tato kategorie by se dala definovat jako trojský kůň. Třetí kategorií byla obecně podezřelá rozšíření. Pro zástupce z této řady platilo, že ač nebyla na první pohled zřejmá pravá funkcionalita, tak ve zdrojovém kódu docházelo ke skrývání `XMLHttpRequest` atp. Jednalo se o podezřelé atributy bez možnosti ověření pravé funkcionality. K testování těchto rozšíření by bylo potřeba použít metody dynamické analýzy.

V této práci byl také naznačen jednoduchý postup, pomocí něhož lze obecně otestovat škodlivost rozšíření. Tento postup, nastíněný v kapitole 11.3, vychází z několika poznatků. Prvním z nich bylo, že rozšíření z kategorie adware v sobě velmi často obsahovala v libovolné formě řetězec s názvem populární hry. Nejčastějším řetězcem bylo slovo Minecraft. Každé rozšíření, které v sobě toto slovo neslo, bylo adware. Otevřenou otázkou zůstává, zda za těmito škodlivými rozšířeními nestojí jeden autor či skupina.

Na druhou stranu reprezentantem kategorie trojského koně byla nejčastěji rozšíření nesoucí název Wallpaper. Ta leckdy mimo deklarovanou funkcionalitu na pozadí přistupovala k citlivým údajům a odesílala je na domény třetích stran.

V průběhu analýzy bylo též zjištěno, že pro odhadnutí škodlivosti rozšíření je potřeba jak statického, tak i dynamického úhlu pohledu. Mnohdy nebyla činnost

zřejmá pouze ze zdrojového kódu, ale bylo potřeba dané rozšíření spustit v prohlížeči.

Velmi zásadním zjištěním bylo, že při zavedení nové úrovně obrany dochází vždy ke zvětšení útočné plochy. Při implementaci programu v programovacím jazyce Java bylo použito pro sledování použitých URL adres ve zdrojovém kódu regulárních řetězců. Jak se ovšem později ukázalo, pokud byl soubor se zdrojovým kódem relativně velký (například 1MB), pak způsob zpracování regulárních řetězců v programovacím jazyce Java neumožnil tento soubor přečíst. Došlo též k vyvolání výjimky, která způsobila pád celého programu. Ač se nejednalo o zásadní počin, ukázalo se být vhodné přemýšlet i o útocích přímo proti aplikaci samotné.

Výsledkem této práce je aplikace, která umožňuje automatizovat analýzu zdrojového kódu použitého v daném rozšíření a jeho následnou klasifikaci. Tato aplikace je do jisté míry schopna škodlivou činnost identifikovat a uživatele na ni upozornit.

13 Závěry a doporučení

Systém detekce se ukázal být do jisté míry funkční pro reprezentanty z kategorie adware a také kategorie trojského koně. V obou případech se ovšem jednalo o příklady neobfuskovaných zdrojových kódů. Tato skutečnost možnost reálného nasazení zmenšuje. Vytvoření efektivního programu schopného reálného nasazení by vyžadovalo vytvořit systém se složitostí na úrovni některých antivirových zařízení. Je tomu tak především proto, že existuje nespočet způsobů, jakými lze škodlivou činnost skrývat, jak komunikovat s doménami atp.

Jednou z vlastností navrženého systému je, že je třeba jej opakovaně přeučovat. Triky vývojářů škodlivých rozšíření se totiž neustále mění a přizpůsobují se novým podmínkám. Z tohoto důvodu je potřeba lidského experta, který bude rozšíření analyzovat a na základě pozorování stanovovat pravděpodobnosti výskytu sledovaných atributů ve škodlivém rozšíření. Proto systém Tali načítá pravděpodobnosti z externího souboru, aby jej bylo možné opakovaně aktualizovat.

Největší výzvou v této práci se ukázala být nutnost vytvořit vlastní repositář škodlivých rozšíření. Jen pro získání trénovacího vzorku o 130 položkách bylo potřeba ručně projít celkem přes 500 zdrojových kódů. Ruční procházení takového množství kódů klade velké nároky na experta, který má rozšíření klasifikovat. Tato skutečnost poukazuje na to, že problematika detekce škodlivých rozšíření je v současnosti stále velkou výzvou.

Jedním z rozšíření práce by mohlo být zkoumání vlivu jiných klasifikátorů na chování aplikace. Pro malé trénovací vzorky je naivní bayesovský klasifikátor vhodný, ale práci je možné do budoucna rozšířit o porovnání více odlišných způsobů klasifikace při použití většího množství trénovacích dat. Větší množství dat ovšem přináší potřebu analyzovat značné množství zdrojových kódů.

Problémem se ukázala být dynamická analýza. Teoreticky by bylo možné rozšířit práci o detekci obfuskace anebo o zaznamenávání volání funkcí. Pak by bylo možné hledat podezřelé sekvence volání, ovšem jedná se o téma, které je značně komplexní a samo o sobě by vyžadovalo další výzkum v této oblasti. Lze ovšem říci, že i přes omezenou uplatnitelnost při detekci v reálném prostředí je tento systém za současného stavu schopen detekovat jistou škodlivou činnost.

14 Seznam použité literatury

A Basic HTML5 Template For Any Project. LAZARIS, Louis. Sitepoint [online]. [cit. 2017-11-19]. Dostupné z: https://www.sitepoint.com/a-basic-html5-template/

AGGARWAL, Anupama, Bimal VISWANATH, Liang ZHANG, Saravana KUMAR, Ayush SHAH a Ponnurangam KUMARAGURU. I Spy with My Little Eye: Analysis and Detection of Spying Browser Extensions [online]. [cit. 2017-12-12]. Dostupné z: https://arxiv.org/pdf/1612.00766.pdf

ALCORN, Wade, Christian FRICHOT a Michele ORRÙ. The browser hacker's handbook. Indianapolis, IN: Wiley, 2014. ISBN 978-1118662090.

Anatomy of an extension. MDN Web Docs [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Anatomy_of_a_WebExtension

Background Pages. Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/extensions/background_pages

BARTH, A., et al. HTTP State Management Mechanism [online]. 2011 [cit. 2017-11-19]. Dostupné z: https://tools.ietf.org/html/rfc6265

BERNERS-LEE, Timothy, et al. Uniform Resource Identifier (URI): Generic Syntax [online]. 2005 [cit. 2017-11-19]. Dostupné z: https://www.ietf.org/rfc/rfc3986.txt

Client–Server Architecture. FURHT, Borko, ed. Encyclopedia of Multimedia [online]. Boston, MA: Springer US, 2008, s. 61-61 [cit. 2018-01-27]. DOI: 10.1007/978-0-387-78414-4_187. ISBN 978-0-387-74724-8. Dostupné z: http://www.springerlink.com/index/10.1007/978-0-387-78414-4_187

Content Scripts. Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/extensions/content_scripts

Content Security Policy (CSP). Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/extensions/contentSecurityPolicy

CROSS SITE SCRIPTING (XSS) ATTACKS. Web Application Security Knowledge Center | Incapsula [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://www.incapsula.com/web-application-security/cross-site-scripting-xss-attacks.html

Cross-Origin XMLHttpRequest. Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/apps/xhr

CRX Package Format. Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/extensions/crx

Declare Permissions. Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/extensions/declare_permissions

DESMEDT, Yvo. Man-in-the-Middle Attack. TILBORG, Henk C. A., ed. Encyclopedia of Cryptography and Security [online]. Springer US, 2005, s. 368-368 [cit. 2018-01-27]. DOI: 10.1007/0-387-23483-7_241. ISBN 978-0-387-23473-1. Dostupné z: http://www.springerlink.com/index/10.1007/0-387-23483-7_241

DOSTÁLEK, Libor a Alena KABELOVÁ. Velký průvodce protokoly TCP/IP a systémem DNS. 2. aktualiz. vyd. Praha: Computer Press, 2000. Komunikace & sítě. ISBN 80-7226-323-4.

Event Pages. Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/extensions/event_pages

FIELDING, Roy, et al. Hypertext Transfer Protocol -- HTTP/1.1[online]. 1999 [cit. 2017-11-19]. Dostupné z: https://tools.ietf.org/html/rfc2616

Forbidden header name. MDN Web Docs[online]. 2017 [cit. 2017-11-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Forbidden_header_name

HE, Jingrui a Yada ZHU. Social Engineering/Phishing. ALHAJJ, Reda a Jon ROKNE, ed. Encyclopedia of Social Network Analysis and Mining[online]. New York, NY: Springer New York, 2014, 2014-10-5, s. 1777-1783 [cit. 2018-01-27]. DOI: 10.1007/978-1-4614-6170-8_290. ISBN 978-1-4614-6169-2. Dostupné z: http://link.springer.com/10.1007/978-1-4614-6170-8_290

Chrome Extensions Archive [online]. 2018 [cit. 2018-04-01]. Dostupné z: https://crx.dam.io/

JAGPAL, Nav, Eric DINGLE, Jean-Philippe GRAVEL, Panayiotis MAVROMMATIS, Niels PROVOS, Moheeb ABU RAJAB a Kurt THOMAS. Trends and Lessons from Three Years Fighting Malicious Extensions. In: 24th USENIX Security Symposium. 2015, s. 579-593. ISBN 978 -1- 931971-232.

Java plug-in does not work in Firefox after installing Java. Java[online]. [cit. 2018-01-24]. Dostupné z: https://java.com/en/download/help/firefox_java.xml

LEE, Vanlam, Ian WELCH, Xiaoying GAO a Peter KOMISARCZUK. Anatomy of drive-by download attack. In: AISC '13 Proceedings of the Eleventh Australasian Information Security Conference [online]. 2013, 2013, s. 49-58 [cit. 2018-01-24]. ISBN 978-1-921770-23-4. Dostupné z: https://dl.acm.org/citation.cfm?id=2525489

Message Passing. Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/extensions/messaging

Native Messaging. Chrome developer [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.chrome.com/extensions/nativeMessaging

PEDERSEN, Torben. HTTPS, Secure HTTPS. TILBORG, Henk C. A., ed. Encyclopedia of Cryptography and Security [online]. Springer US, 2005, s. 268-269 [cit. 2018-01-27]. DOI: 10.1007/0-387-23483-7_189. ISBN 978-0-387-23473-1. Dostupné z: http://www.springerlink.com/index/10.1007/0-387-23483-7_189

PERROTTA, Raffaello a Feng HAO. Botnet in the Browser: Understanding Threats Caused by Malicious Browser Extensions [online]. 2017 [cit. 2018-01-24]. Dostupné z: https://arxiv.org/abs/1709.09577

REFLECTED CROSS SITE SCRIPTING (XSS) ATTACKS. Web Application Security Knowledge Center | Incapsula [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://www.incapsula.com/web-application-security/reflected-xss-attacks.html

SANCHEZ-ROLA, Iskander a Igor SANTOS. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In: Proceedings of the 26th USENIX Security Symposium [online]. 2017, s. 679-694 [cit. 2018-01-27]. ISBN 978-1-931971-40-9.

STAROV, Oleksii a Nick NIKIFORAKIS. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In: 2017 IEEE Symposium on Security and Privacy (SP) [online]. IEEE, 2017, 2017, s. 941-956 [cit. 2018-01-23]. DOI: 10.1109/SP.2017.18. ISBN 978-1-5090-5533-3. Dostupné z: http://ieeexplore.ieee.org/document/7958618/

STUTTARD, Dafydd a Marcus PINTO. The web application hacker's handbook: finding and exploiting security flaws. 2nd ed. Chichester: John Wiley [distributor], c2011. ISBN 978-1118026472.

TCP/IP. The Network Encyclopedia [online]. 2013 [cit. 2018-01-24]. Dostupné z: http://www.thenetworkencyclopedia.com/entry/tcp-ip

Testing for Clickjacking (OTG-CLIENT-009). OWASP [online]. [cit. 2018-01-27]. Dostupné z: https://www.owasp.org/index.php/Testing_for_Clickjacking_(OTG-CLIENT-009)

Uniform Resource Identifier (URI) Schemes. IANA[online]. 2018 [cit. 2018-01-24]. Dostupné z: https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml

VU, Quang Hieu, Mihai LUPU a Beng Chin OOI. Architecture of Peer-to-Peer Systems. VU, Quang Hieu, Mihai LUPU a Beng Chin OOI. Peer-to-Peer Computing [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 2010-10-20, s. 11-37 [cit. 2018-01-27]. DOI: 10.1007/978-3-642-03514-2_2. ISBN 978-3-642-03513-5. Dostupné z: http://link.springer.com/10.1007/978-3-642-03514-2_2

What are extensions? MDN Web Docs [online]. 2018 [cit. 2018-01-24]. Dostupné z: https://developer.mozilla.org/en-US/Add-ons/WebExtensions/What_are_WebExtensions

ZALEWSKI, Michal. The tangled Web: a guide to securing modern Web applications. San Francisco: No Starch Press, c2012. ISBN 978-1-59327-388-0.

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Systémové inženýrství a informatika
Forma: Kombinovaná
Obor/komb.: Informační management (im2-k)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Maisner Patrik	Zeyerova 715/6, Hradec Králové - Pražské Předměstí	I1600242

TÉMA ČESKY:

Bezpečnost webových prohlížečů

TÉMA ANGLICKY:

WEB BROWSER SECURITY

VEDOUCÍ PRÁCE:

Ing. Zuzana Němcová, Ph.D. - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Teoretická část
Základy webových technologií (HTTP,...)
Architektura moderních webových prohlížečů
Zranitelnosti webových prohlížečů (XSS, CSRF,...)
Architektura systémů rozšíření pro jednotlivé prohlížeče
Útoky proti doplňkům
Útoky za použití doplňků
Doplňky a malware

Praktická část
Vývoj pluginu pro detekci škodlivých rozšíření

Cíle
Cílem práce je analyzovat zranitelnosti webových prohlížečů a navrhnout plugin pro automatickou identifikaci škodlivých doplňků.

SEZNAM DOPORUČENÉ LITERATURY:

ALCORN, Wade, Christian FRICHOT a Michele ORRU?. The browser hacker's handbook. ISBN 978-1-118-66209-0.
ZALEWSKI, Michal. The tangled Web: a guide to securing modern Web applications. San Francisco: No Starch Press, c2012. ISBN 1-59327-388-6.
KENNEDY, David. Metasploit: the penetration tester's guide. San Francisco: No Starch Press, c2011. ISBN 1-59327-288-x.
STUTTARD, Dafydd a Marcus PINTO. The web application hacker's handbook: finding and exploiting security flaws. 2nd ed. Chichester: John Wiley [distributor], c2011. ISBN 978-1-118-02647-2.

Podpis studenta: Maisner ..... **Datum:** 13.10.2017

Podpis vedoucího práce: ..... **Datum:** 16.10.2017