VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

UZAMYKÁNÍ FORMULÁŘŮ V ROZŠÍŘENÍ JAVASCRIPT RESTRICTOR

FORMLOCK FOR JAVASCRIPT RESTRICTOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MATYÁŠ SZABÓ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LIBOR POLČÁK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Szabó Matyáš**
Program: Informační technologie
Název: **Uzamykání formulářů v rozšíření JavaScript Restrictor**
FormLock for JavaScript Restrictor
Kategorie: Web

Zadání:

1. Seznamte se s rozšířením JavaScript Restrictor.
2. Otestujte rozšíření FormLock a jeho podporu pro aktuální prohlížeč Firefox a prohlížeče založené na projektu Chromium.
3. Navrhněte integraci schopností rozšíření FormLock do projektu JavaScript Restrictor.
4. Integrujte FormLock do projektu JavaScript Restrictor.
5. Otestujte funkcionalitu na různých stránkách, českých i zahraničních. Minimálně na 20 stránkách proveďte důkladné ruční otestování.
6. Zhodnoťte výsledky projektu a navrhněte možná pokračování.

Literatura:

- Oleksii Starov, Phillipa Gill a Nick Nikiforakis. Are You Sure You Want to Contact Us? Quantifying the Leakage of PII via Website Contact Forms. Proceedings on PETS 2016(1): str. 20-33.
- Günes Acar, Steven Englehardt a Arvind Narayanan. No boundaries: data exfiltration by third parties embedded on web pages. Proceedings on PETS 2020(4): str. 220-238.
- Martin Bednář. *Automatické testování projektu JavaScript Restrictor*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 26. října 2020

Abstrakt

Tato práce se zaměřuje na problematiku ochrany osobních údajů při vyplňování formulářů na internetu. V této práci je analyzováno řešení ve formě webového rozšíření Formlock, které upozorňuje uživatele na potenciálně nebezpečné formuláře a snaží se jej chránit před únikem těchto dat třetím stranám. Rozšíření Formlock je analyzováno a na základě získaných poznatků je navržena jak integrace do rozšíření Javascript Restrictor, tak vylepšení současných opatření. Nově implementovaná opatření jsou otestována a jsou vyhodnoceny možnosti k jejich budoucímu rozšíření.

Abstract

This thesis focuses on protecting personally identifiable information (PII) during filling and submitting of web forms. Formlock, a prototype trying to resolve the PII leakage caused by forms by warning the user about the potentially malicious web forms and giving them an option to try and prevent the leak, is tested and then it's defensive capabilities are improved and integrated into Javascript Restrictor. Lastly, the new and integrated measures are tested and their possible future improvements are evaluated.

Klíčová slova

webové formuláře, JavaScript Restrictor, JavaScript, uzamykání formulářů, bezpečnost webových prohlížečů, WebRequest API

Keywords

web forms, JavaScript Restrictor, JavaScript, locking web forms, web browser security, WebRequest API

Citace

SZABÓ, Matyáš. *Uzamykání formulářů v rozšíření JavaScript Restrictor*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Uzamykání formulářů v rozšíření JavaScript Restrictor

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Libora Polčáka, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Matyáš Szabó
8. května 2021

Poděkování

Chtěl bych poděkovat vedoucímu mé práce Ing. Liborovi Polčákovi Ph.D. za čas, který věnoval konzultacím a cenným radám, které vedly k dokončení této práce.

Obsah

Kapitola 1

Úvod

V současné době naprostá většina stránek, které každodenně navštěvujeme, používá JavaScript [23]. Použitím tohoto jazyku se ulehčuje zátěž na cílové servery, jelikož mnoho výpočtů je možné provést na straně klienta. Návštěvník poté vidí dynamickou webovou prezentaci. Avšak prvky této prezentace mohou obsahovat různé programy, které budou sbírat všemožné informace.

Využití těchto sledovacích prvků se bohužel stává častější každým dnem. Sledovací prvky se mohou na stránku umístit z mnoha důvodů, ať už to je cílený záměr vlastníka, který chce vést sběr dat nebo poskytovat někomu údaje k stejnému cíli, nebo pouhé použití kódu třetích stran obsahujícího různé sledovací prvky.

Třetí strany mohou často toto sledování realizovat poskytováním obsahu a reklam netušícím klientům.

Sledovací prvky bývají umístěny všude včetně kontaktních stránek, kde má uživatel možnost zadat do formuláře svoje osobní údaje jako emailová adresa, jméno a příjmení, což z nich dělá ideální cíle pro získání identifikujících informací.

Ochrana proti tomuto typu útoků spadá hlavně na stranu uživatele, jelikož se nemůžeme spoléhat na to, že neopatrný programátor svoje stránky opraví a nebo že vlastník přestane se svými taktikami na požádání.

Cílem této práce bylo nastudovat problematiku úniku osobních dat třetím stranám a jak implementovat obranná opatření do již existujícího rozšíření prohlížeče JavaScript Restrictor, které obsahuje nemálo mechanik zabraňujících útočníkům provádění škodlivého kódu u uživatele. Pro integraci bylo vybráno rozšíření Formlock [25], které již obsahovalo částečné řešení tohoto problému.

Práce je rozdělena následovně. Kapitola 2 vysvětluje pojem rozhraní prohlížeče, jejich strukturu a popisuje rozhraní, která byla důležitá v rámci této práce a shrnuje nové změny relevantní pro vyvíjení rozšíření na prohlížeči Google Chrome. Kapitola 3 popisuje motivaci k vytvoření Formlocku, jeho návrh a jeho funkčnost při osobním testování. Kapitola 4 rozebírá obranné aspekty rozšíření JavaScript Restrictor, oblasti kde se jeho funkcionalita překrývá s Formlockem a interní fungování úrovní obrany. Kapitola 5 obsahuje krátký popis, jak by se dalo Formlock integrovat do JavaScript Restrictoru a jakým způsobem by se dala současná implementace vylepšit a rozšířit. Kapitola 6 obsahuje implementační detaily včetně rozdílů na různých prohlížečích. Kapitola 7 popisuje postupy použité při manuálním testování včetně jednoduché stránky na testování manipulace s lokálními daty. V kapitole 8 jsou na závěr rozebrány výsledky této práce a jsou navrženy vylepšení pro případ pokračování.

Kapitola 2

Rozšíření prohlížeče

Pro pochopení fungování rozšíření rozebíraných v této práci je nutné projít si fundamentální základy rozšíření prohlížečů. Tato kapitola se zabývá vysvětlením základních pojmů souvisejících s fungováním rozšíření na různých prohlížečích.

Sekce 2.1 vysvětluje pojem rozšíření a rozebírá základní principy, kterými se všechna rozšíření řídí. Sekce 2.2 popisuje nejdůležitější komponenty struktury rozšíření a jejich funkcionalitu. Sekce 2.3 vysvětluje principy fungování rozhraní, která byla důležitá pro implementaci této práce. Sekce 2.4 probírá rozdíly mezi jednotlivými implementacemi rozhraní pro rozšíření. Sekce 2.5 vysvětluje nejdůležitější změny v nové verzi platformy pro rozšíření na prohlížeči Google Chrome. Sekce 2.2, 2.3 a 2.4 jsou převážně převzaté z dokumentace Mozilla MDN Web Docs [10].

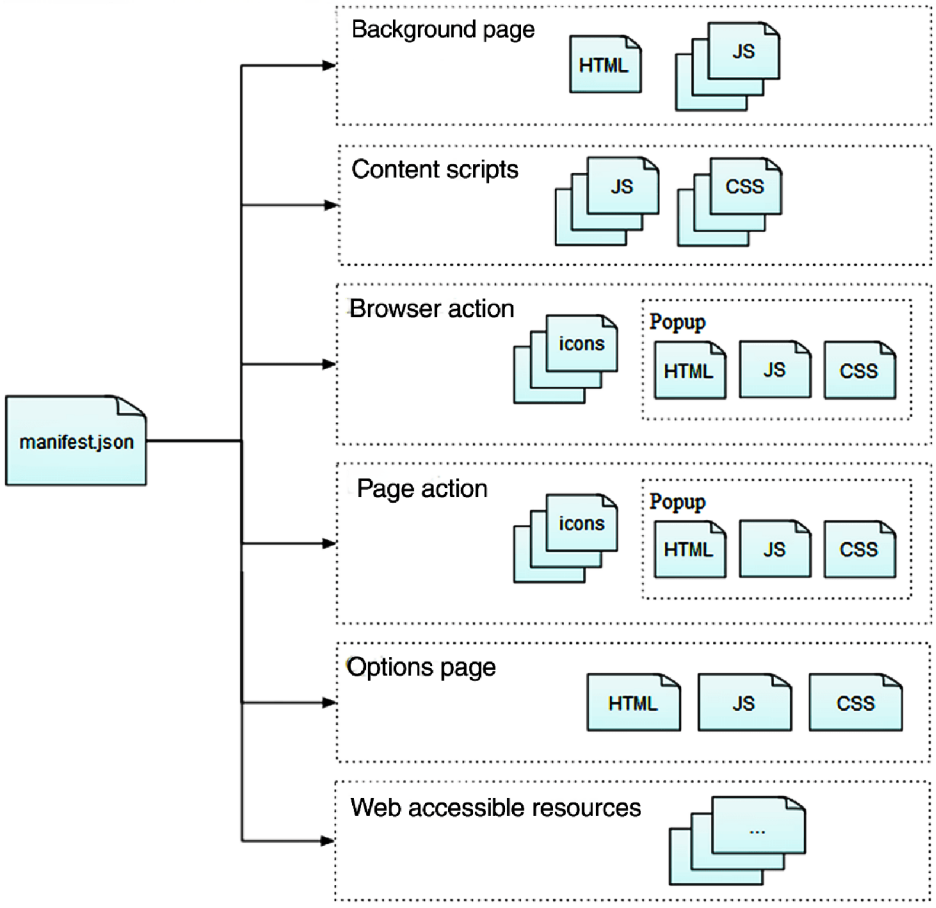
2.1 Co je rozšíření?

Rozšíření prohlížeče jsou programy, které běží v kontextu internetového prohlížeče a umožňují vývojářům rozšířit a modifikovat jeho chování [20]. Možnosti těchto rozšíření zahrnují vše od jednoduchých úprav jako například zvýraznění textu na stránce nebo ukázání vyskakovacího okna až po složitější případy použití jako blokování požadavků dle zadaných parametrů nebo manipulaci s lokálním úložištěm či databází.

Byť měl původně každý prohlížeč svoje rozhraní pro rozšíření (například XUL a XPCOM u Firefoxu), tak se v posledních letech začalo převážně využívat rozhraní WebExtensions, které je modelováno podle rozhraní, které používá Google Chrome a další prohlížeče založené na jádru Chromium. WebExtensions rozhraní však není naprosto identické mezi prohlížeči a má místy rozdíly, které bývají vyznačené přímo v dokumentaci [9].

Všechna rozšíření prohlížeče jsou umístěna v tzv. sandboxech (izolované prostředí uvnitř kterého nemůže aplikace ovlivňovat další programy nebo jejich data). To v praxi znamená, že v daném prohlížeči by mohl být nainstalován libovolný počet rozšíření, která by si nebyla automaticky vědoma přítomnosti ostatních. Rozšíření tedy nemohou přistupovat ke kódu a k paměti ostatních, nedochází ke konfliktu jmen (takže dvě rozšíření mohou mít například stejný soubor se jménem *background.js*) a ke komunikaci mezi rozšířeními dochází pomocí mechaniky zpráv, které bývá poskytována rozhraním pro zprávy.

2.2 Struktura rozšíření



Obrázek 2.1: Struktura rozšíření prohlížeče. Převzato z [10]

Manifest.json je jediný soubor, který musí nutně být v každém rozšíření. Je to textový soubor ve formátu JSON (JavaScript Object Notation), který obsahuje informace jako je jméno, autoři, verze, poskytovaná práva a podobně. Zároveň obsahuje odkazy na ostatní soubory v rámci rozšíření.

První jsou tzv. *background scripts* (skripty, které běží na pozadí). Tyto slouží k udržení dlouhodobého stavu rozšíření nebo provádění operací, které jsou nezávislé na životnosti jednotlivých webových stránek a nebo oknech prohlížeče. Tyto skripty jsou spuštěny hned po načtení rozšíření samotného a jsou přítomny, dokud uživatel dané rozšíření neodstraní nebo nevypne. Těmto skriptům jsou dostupná rozhraní z WebExtensions, což však vyžaduje deklarovaní práv v manifestu (například *cookies* pro využití rozhraní pro práci s cookies). Dále jsou dostupná i DOM (Document Object Model) rozhraní, jelikož tyto skripty běží na stránkách v pozadí. Pokud stránka na pozadí nebyla vytvořena explicitně autorem, tak je pro daný skript vygenerována automaticky. Skripty na pozadí mohou také odesílat *XHR* (XMLHttpRequest) požadavky na jakékoliv cíle deklarované v manifestu, ale nemohou přímo manipulovat s obsahem stránek. K tomu se používají tzv. *content scripts*, se kterými lze následně komunikovat pomocí zpráv.

Součástí rozšíření také mohou být rozšiřující stránky, které nesouvisí přímo s nějakým předefinovaným komponentem uživatelského rozhraní [13]. Tyto stránky se neuvádí v manifestu, stačí, aby byly umístěny v lokaci daného rozšíření a následně byly otevřeny pomocí *windows.create()* nebo *tabs.create()*. Tyto stránky mají také přístup k rozhraním určeným pro skripty na pozadí. Speciálním případem těchto stránek jsou boční panely, vyskakovací okna a stránky s nastavením rozšíření, které se deklarují v manifestu.

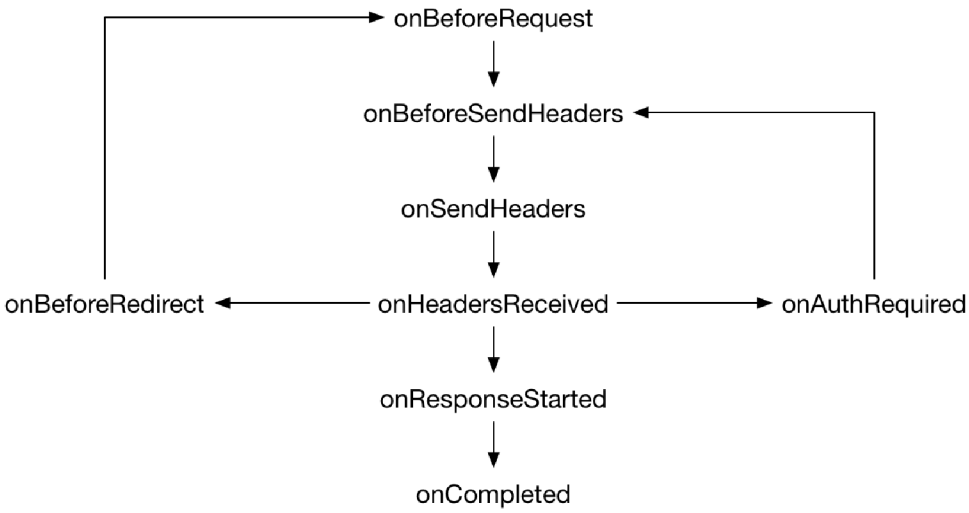
Pro práci s obsahem stránek jsou určeny tzv. *content scripts*, které mají přístup k DOM daných stránek a běží v jejich kontextu. Liší se však od skriptů, které používá stránka samotná tím, že mohou posílat *XHR* požadavky na domény jiného původu, mají dostupnou malou podmnožinu rozhraní WebExtensions a mohou komunikovat se svými skripty na pozadí pomocí zpráv. Ke skriptům stránky sice nemohou přímo přistupovat, ale mohou využít dostupného rozhraní pomocí *window.postMessage()* pro zaslání zprávy.

Na konec můžeme také do manifestu uvést zdroje, které se stanou dostupné na webu. Mezi ně patří například obrázky, HTML, CSS a podobně, které chceme zpřístupnit skriptům na pozadí, vloženým skriptům a nebo skriptům stránek.

2.3 Důležitá rozhraní

WebRequest

Rozhraní WebRequest definuje sadu událostí ze životního cyklu HTTP požadavků a umožňuje na ně navázat funkce, pomocí kterých může vývojář nejen sledovat průběh těchto požadavků, ale v určitých částech i požadavky modifikovat. Tento životní cyklus lze vidět na obrázku 2.2, který zobrazuje všechny dílčí fáze. Je nutné pouze podotknout, že v každé části průběhu požadavku může nastat fáze *onErrorOccurred* indikující chybu.



Obrázek 2.2: Životní cyklus požadavku. Převzato z [19]

U všech fází (až na *onErrorOccurred*) lze při přípravě odposlechu udat tři parametry: callback (funkce, kterou vyvolá daná událost), objekt, jehož atributy slouží k filtrování požadavků (url, druh zdroje, identifikátor okna apod.), a další objekt, který slouží k dodání dalších informací specifických pro danou etapu požadavku. Callback při vyvolání dostane

objekt *details*, který obsahuje informace o dané etapě požadavku včetně unikátního identifikátoru, podle kterého prohlížeč rozlišuje jednotlivé požadavky [19].

Fáze *onBeforeRequest* nastává před posláním požadavku, což znamená ještě před tím, než je navázáno TCP spojení. *onBeforeSendHeaders* je vyvolána těsně před posláním požadavku, kdy jsou už připraveny počáteční hlavičky. Tato fáze existuje, aby vývojář mohl upravit hlavičky ještě před jejich odesláním. *onSendHeaders* je hlavně informativní fáze, která nastane po tom, co všechna rozšíření měla možnost upravit hlavičky a jsou tedy ve svém finálním stavu před posláním do sítě. *onHeadersReceived* je vyvolána vždy, když dorazí hlavička HTTP odpovědi, což vzhledem k různým přesměrováním a autentizaci může nastat několikrát za jeden požadavek. Tato fáze také umožňuje upravovat příchozí hlavičky. *onAuthRequired* nastane když server požaduje od uživatele údaje k autentizaci. Při této fázi může vývojář požadavek zablokovat, poslat údaje asynchronně a nebo synchronně. *onBeforeRedirect*, *onResponseStarted* a *onCompleted* jsou pouze informativní fáze, které nastanou před přesměrováním, při přijetí prvního bytu a při dokončení celého požadavku v tomto pořadí.

Web Storage

Rozhraní Web Storage je mechanismus sloužící k jednoduchému ukládání dat prohlížeči v párech klíčů a jejich hodnot v intuitivnější formě než cookies [18]. Web Storage poskytuje dva sklady pro práci, *sessionStorage* a *localStorage*.

sessionStorage slouží k ukládání dat, jejichž životnost bude pouze v rámci relace (dokud je prohlížeč otevřený, včetně obnovení stránky apod.). Poskytuje separované místo pro data z každého zdroje. Zároveň je poskytované místo větší než pro cookies (5MB). *localStorage* má stejnou funkci, ale jeho data nejsou omezena pouze na současnou relaci a přetrvají i její ukončení. Uložená data neobsahují žádný čas vypršení a dají se odstranit pouze za použití JavaScriptu a nebo promazáním pomocí vývojářských nástrojů v prohlížeči.

Při práci s těmito sklady je taky nutné mít na paměti, že v rámci původu se bere v potaz i použitý protokol, takže například *http://stranka.cz* a *https://stranka.cz* by měly oddělené sklady.

IndexedDB

IndexedDB je nízkoúrovňové rozhraní, které slouží k uložení velkého množství dat na klientské straně, což je jednen z aspektů, kterými se liší od *Web Storage* [14]. Toto rozhraní poskytuje transakční databázový systém, který je podobný systémům řízení báze dat relačních databází založených na jazyku SQL (Structured Query Language). Na rozdíl od těchto systémů však využívá tabulek s fixními sloupci a celá databáze je založená na JavaScriptu a objektově orientovaná.

Při práci s *IndexedDB* je nutné mít na paměti její základní koncepty.

Data v databázích toho typu jsou uložena v páru klíč a hodnota. Hodnoty mohou být komplexní strukturované objekty a klíče mohou být atributy těchto objektů. Stejně jako u ostatních databází jsou i zde dostupné indexy, které můžeme použít, abychom rychle hledali podle atributů těchto objektů.

IndexedDB je založena na transakčním databázovém modelu, takže všechna práce probíhá v rámci transakcí. V rámci tohoto rozhraní máme mnoho objektů, které reprezentují indexy, tabulky, kurzory apod., ale všechny jsou svázány s určitou transakcí. Nelze tedy provádět příkazy nebo otevírat kurzory mimo transakce. Zároveň jsou transakce prováděny plně automaticky a nelze je provést manuálně [14].

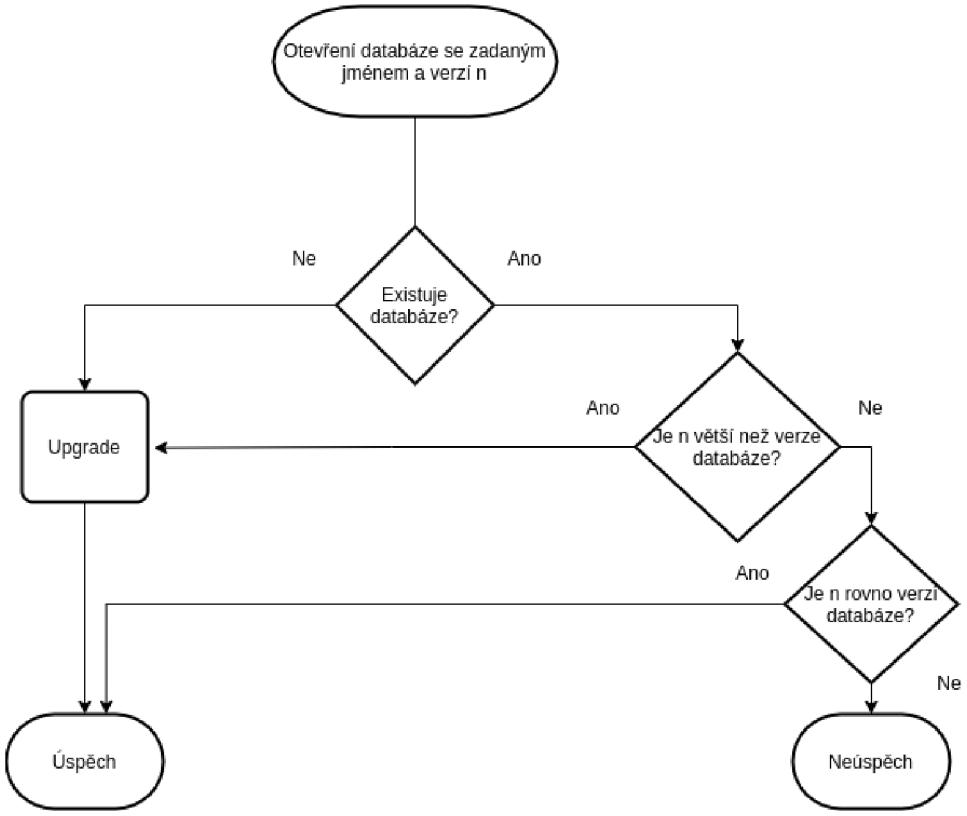
Toto rozhraní je převážně asynchronní, takže je nutné předávat callback funkce. Všechny požadavky používají události DOM k upozornění vývojáře na výsledek dané operace. Tyto události DOM mají atribut *type*, který většinou indikuje buď úspěch nebo selhání události. Úspěšné události se nepropagují výš a nelze je zrušit, zatímco neúspěšné události se propagují a lze je zrušit. Toto je důležité mít na paměti, jelikož neošetřené neúspěchy přeruší transakci, ve které se nachází. Požadavky mají také atributy *readyState*, *result* a *errorCode*, které popisují stav požadavku. Atribut *result* může být různých typů na základě toho, jak bych vygenerován [14].

IndexedDB databáze jsou objektově orientované, takže se v databázích nachází tzv. *object stores* (mechanismus, který ukládá záznamy v databázi), které jsou určitého typu.

IndexedDB nepoužívá jazyk SQL. Používá dotazy na index, který vyprodukuje kurzor, se kterým se iteruje přes sadu výsledků.

Je dodržována politika stejného původu, přičemž jako původ se zde bere kombinace domény, protokotolu aplikační vrstvy a port, na kterém je skript prováděn. Každý původ má svoji sadu databází, které jsou s ním asociovány.

Dalším důležitým aspektem *IndexedDB* je vlastní systém verzí schémat vizualizovaný na obrázku 2.3. Jelikož jsou tyto databáze na straně klienta a ne serveru, tak k nim vývojáři stránky nemají neustálý přístup a není tedy možné všechny ve stejný čas aktualizovat na nová schémata, která by obsahovala například nové *object stores*. K tomuto slouží celočíselný údaj verze, který je zadáván vždy při otevření databáze [14].



Obrázek 2.3: Otevření databáze

Vždy při otevírání databáze se zkontroluje, jestli databáze se zadaným jménem vůbec existuje. Pokud ne, tak se zavolá metoda pro aktualizaci databáze, kde bývá definováno

vytvoření *object stores* pro danou verzi. Pokud databáze existuje, tak je zkontrolováno, jestli je zadaná verze větší než současná verze dané databáze v prohlížeči, což případně vede opět na aktualizaci. Otevření databáze s menší verzí, než momentálně existuje v prohlížeči, není možné a vede k chybě. Vývojářům stránky tedy stačí při úpravách schématu databáze pouze rozšířit funkci sloužící k aktualizaci o nové změny, které následně provede klientský prohlížeč, až si stáhne novou verzi skriptu.

2.4 Rozdíly mezi implementacemi WebExtensions

V rámci této práce je nutné brát v potaz, že na standardizaci rozhraní WebExtensions se stále ještě pracuje, takže i když jej podporuje většina prohlížečů včetně Firefoxu, Chrome, Edge a Opery, tak se pořád nacházejí rozdíly v jednotlivých implementacích. Zde budou popsány hlavně rozdíly mezi implementacemi pro prohlížeče založené na jádře Chromium a prohlížečem Firefox, který pohání jádro Gecko [12].

Prvním rozdílem je jmenný prostor reprezentovaný proměnnou, přes kterou se přistupuje k rozhraním určeným pro rozšíření. Skupina prohlížečů založených na Chromiu používá jmenný prostor *chrome*, zatímco Firefox využívá *browser*. Firefox se však na pomoc vývojářům, kteří potřebují přenést svoje rozšíření, rozhodl podporovat i použití proměnné *chrome* a callback funkcí, ale to ještě neznamená, že budou naprosto všechna rozšíření fungovat stejně [11].

Druhým rozdílem je zpracování asynchronních událostí. Standardem společného rozhraní by mělo být využití tzv. *promises* (objekt reprezentující eventuální úspěšné nebo neúspěšné zakončení asynchronní operace [17]). Avšak zatím všechny prohlížeče kromě Firefoxu používají callback funkce.

Třetím rozdílem jsou poskytovaná rozhraní na jednotlivých prohlížečích. Některé funkce mohou poskytovat podobnou, ale ne totožnou funkcionalitu a mohou požadovat jiné argumenty. Některá rozhraní dokonce přímo chybí jako například rozhraní *declarativeContent*, které je dostupné na Chromium prohlížečích a DNS rozhraní, které naopak je dostupné na Firefoxu, ale ne na Chromium prohlížečích [11].

Posledním rozdílem je seznam podporovaných klíčů uvnitř manifestu daného rozšíření.

2.5 Chrome Manifest V3

Vývojáři prohlížeče Google Chrome vydali ve verzi 88 tzv. Manifest V3 a rozšíření na něm postavená jsou přijímána na Chrome Web Store od počátku tohoto roku. Manifest V3 je platforma, na základě které jsou rozšíření vyvíjena. Nová verze přináší jak fundamentální změny, tak změny poskytovaných rozhraní.

Mezi hlavní fundamentální změny patří dostupnost uživatelských dat pro daná rozšíření. Autoři Manifestu V3 chtějí, aby rozšíření prohlížeče neměla neustálý přístup k uživatelským datům a aby si místo toho si o různá práva žádala například při běhu. Záměr této změny je, aby uživatel měl větší přehled na tím, co nainstalovaná rozšíření dělají. Za účelem větší bezpečnosti bude také prohlížeč přísněji kontrolovat požadavky rozšíření na zdroje, které jsou mimo kontext daného rozšíření [4].

Service workers místo stránek na pozadí

Mezi první z hlavních funkcionálních změn patří nahrazení stránek na pozadí pomocí tzv. *Service workers* (skript, který běží v pozadí, separátně od webové stránky nebo uživatel-

ských interakcí [16]). Hlavním rozdílem mezi stránkou na pozadí a *Service workerem* je ten, že *Service worker* je ukončen, když není používán a pak restartován, když je zapotřebí. Reakce na zprávy a události se tedy nemohou spoléhat na globální stav, takže musí využít jiných prostředků jako například rozhraní *IndexedDB*, které jim je dostupné.

Změna úprav požadavků

V Manifestu V3 je zavedeno nové rozhraní pro modifikaci požadavků *declarativeNetRequest*. Smyslem tohoto rozhraní je předat kontrolu nad úpravami požadavků prohlížeči. Rozšíření tedy nově nebude zachytávat požadavky a měnit je, ale místo toho požádá prohlížeč o zhodnocení daného požadavku a jeho modifikaci na základě deklarovaných pravidel. Autor rozšíření tedy bude moci implementovat většinu běžných případů použití bez toho, aby potřeboval oprávnění od uživatele [5].

Vzdáleně hostovaný kód

Rozšíření nebudou moci načítat kód, který se nachází mimo kontext daného rozšíření. Cílem této změny je možnost lepšího zhodnocení bezpečnosti rozšíření, která jsou posílána na Chrome Web Store. Autoři nového Manifestu doporučují buď použití vzdálených konfiguračních souborů, které by si rozšíření mohlo stáhnout za běhu a rozhodnout se jaké vlastnosti využít, a nebo přesunout logiku daného kódu na samostatnou webovou službu, se kterou by dané rozšíření mohlo komunikovat.

Promises

S novou verzí Manifestu také přijde větší podpora pro *promises*. Většina populárních rozhraní nyní podporuje *promises* a autoři mají jako cíl tuto podporu rozšířit na všechny související metody. Stále je však možné využít mechaniky *callback* funkcí. Při použití rozhraní s *callback* funkcí se zablokuje vrácení *promise* [6].

Kapitola 3

Únik informací z formulářů

Tato kapitola se zaměřuje na rozšíření Formlock[1], které pro prohlížeč Google Chrome implementuje ochranu uživatele při vyplňování webových formulářů. Bylo představeno v práci *Are You Sure You Want to Contact Us? Quantifying the Leakage of PII via Website Contact Forms* [25] jako prototyp řešení úniku informací z vyplňovaných formulářů.

Sekce 3.1 se zaměřuje na motivaci autorů k provedení průzkumu [25], který zjistil, kolika z nejpopulárnějších 100 000 stránek unikají identifikující data a jak se to děje. V sekci 3.2 je shrnuta funkcionalita rozšíření, zatímco sekce 3.3 obsahuje přehled autorovu realizaci nejdůležitějších ochranných prvků. Sekce 3.4 popisuje dojmy z osobního testování a popisuje několik chyb v současné verzi a sekce 3.5 slouží jako krátké shrnutí celého rozšíření.

Použití Formlocku a editace zdrojových souborů byla schválena autorem Oleksii Starovem za podmínky, že v původních souborech bude v hlavičce uvedeno jeho jméno a odkaz na původní repositář.

3.1 Studie ze Stony Brook University

Motivací pro vytvoření Formlocku byla studie z roku 2016, která se zaměřovala na sledování uživatelů na kontaktních stránkách (stránky, kde uživatelé mohou zadat informace pro kontakt). Součástí cíle této práce bylo pochopit sledování na kontaktních stránkách třetími stranami ve větším měřítku, konkrétně 100 000 nejvíce navštěvovaných stránek na základě seznamu získaného od zdroje Alexa. Hlavními otázkami tohoto hledání byly „Jaký je potenciál pro únik dat?“, „Jaké třetí strany získávají identifikující informace z formulářů omylem?“ a „Jaké třetí strany získávají identifikující informace z formulářů úmyslně?“

Pro automatizaci prohledávání stránek byl využit web crawler (program, který systematicky prochází webové stránky za účelem hledání informací), jehož cílem bylo nalézt kontaktní stránku, identifikovat všechna očekávaná pole se vstupem, naplnit je odpovídajícími údaji a poté formulář odeslat. Zároveň bylo nutné sledovat HTTP provoz kvůli únikům identifikujících dat a přítomnosti obsahu třetích stran, který by se mohl pokoušet o získání daných dat přímo a nebo i s pokusem se dané akce zamaskovat.

Správné nalezení a rozlišení požadovaných prvků však nebylo jednoduché, jelikož uživateli stačí pro navigování stránky vizuální indicie a nemusí řešit například zanoření různých elementů.

Pro nalezení stránky obsahující kontaktní formulář se autoři rozhodli, že mezi stránkami, na které odkazuje hlavní stránky, budou hledat takovou s klíčovým slovem *contact*. Následně

[1] `https://github.com/ostarov/Formlock`

se crawler snažil identifikovat formulář, který byl doopravdy určený ke kontaktování a ne k jiným účelům jako přihlášení nebo vyhledávání. Tohoto docílili určením určitých kritérií jako vyloučení formulářů, které byly neviditelné nebo jejich vykreslovací rozměry byly pod určitou úroveň nebo vyloučení těch, které měly méně než dvě vstupní pole. Z těchto důvodů se tedy nepokoušeli identifikovat formuláře, které byly implementované pomocí jiných technologií jako např. Flash nebo Java.

Pro vyplnění nalezených formulářů se crawler pokoušel o vyplnění jednotlivých polí na základě jejich určení. Zadávání náhodných řetězců a čísel by mohlo vést k zamítnutí odesílaných dat, jelikož stránky většinou využívají JavaScriptové funkce na kontrolu validity zadaných dat. Při neúspěšném odhadnutí účelu daného pole zadal crawler emailovou adresu, která byla vytvořena čistě pro tento experiment. Všechna vyplňovaná data byla identická, aby se je dalo jednodušeji hledat v sesbíraném provozu.

Jako úspěšné vyplnění považovali autoři situaci, kdy byli přesměrování na jinou stránku, která obsahovala fráze indikující správné zadání formuláře.

Za únik identifikujících informací bylo považováno, když byla vyplněná data poslána na doménu třetí strany. Dále také kvantifikovali potenciální úniky sledováním dynamicky načítaných knihoven, které by měly možnost přistoupit k údajům ve formuláři téměř kdykoliv [25].

Při identifikaci úniků dat v HTTP provozu bylo nutné brát v potaz to, že hledání údajů v samotné komunikaci by nebylo efektivní, jelikož by byly vynechány případy, kdy by pro skrytí těchto údajů byla využita různá kódování nebo jiné mechaniky pro jejich zamaskování. Pro detekci různých pokusů o maskování a případný únik dat se autoři rozhodli vždy poslat 3 formuláře, přičemž do prvních dvou zadali stejnou emailovou adresu a do třetí jinou. Následně byly vybrány všechny parametry z jednotlivých požadavků včetně řetězce požadavku, cookies, hlaviček a dotazů POST. Pokud některý z parametrů byl stejný při prvních dvou požadavcích, ale jiný u třetího, tak byl daný formulář označen za potenciálně nebezpečný a byl určen k manuálnímu přezkoumání.

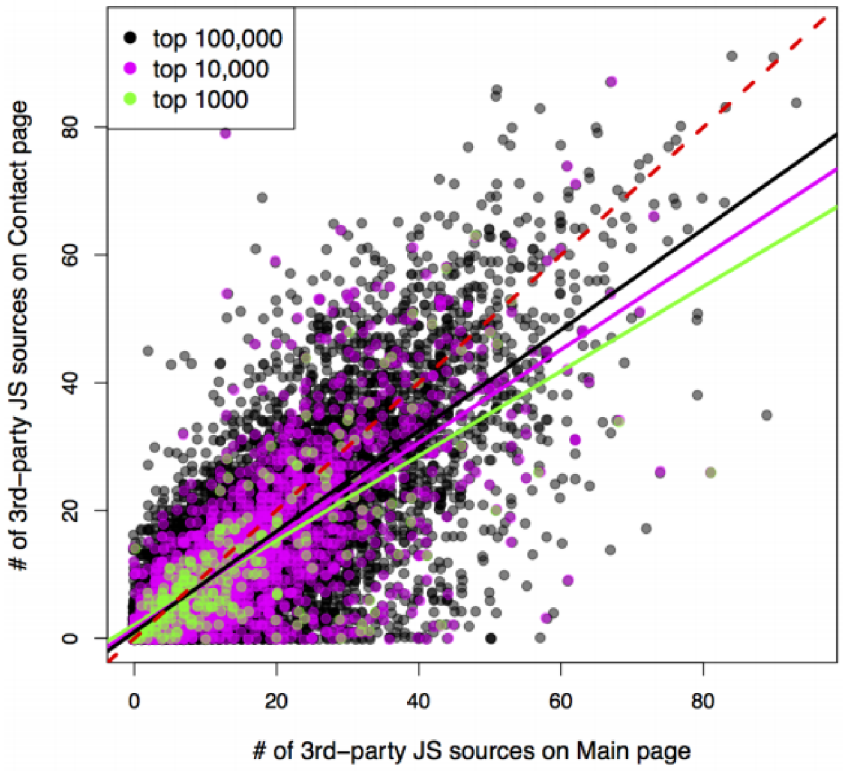
Jako jeden z prvních aspektů byl zkoumán počet načtených knihoven třetích stran, které sice nemusí nutně sbírat uživatelská data, ale jsou toho schopny [21]. Následně také zkoumali poměr zahrnutých knihoven na hlavních a kontaktních stránkách viz obrázek 3.1. Při analýze získaných dat došli k tomu, že populární webové stránky používají kód z desítek cizích unikátních zdrojů, přičemž jejich počet se zvyšuje s klesající popularitou stránek. Zároveň však bylo zjištěno, že většina webových administrátorů bere zřetel na bezpečnost a používají méně cizích knihoven na svých kontaktních stránkách. Avšak 29% dělá naprostý opak využívají 20krát více knihoven na kontaktních stránkách.

Při zkoumání přímých úniků dat pomocí kontrolní proxy bylo zjištěno, že 2.5% (423) ze všech formulářů používalo metodu GET, která není bezpečná pro přímý přenos, jelikož uchovává odeslaná data v URL, ke které může útočník lehce přistoupit pomocí jednoduchého skriptu. To znamená, že při odeslání požadavku na jakoukoliv službu třetí strany jsou údaje z formuláře dostupné bez jakékoliv námahy. Z těchto 2.5% byla identifikující data odeslána až na 3573 unikátních domén třetích stran, přičemž průměrný počet domén třetích stran, které získaly tyto informace byl 8. I takto malé procento z celkového počtu je alarmující, jelikož každá z předem zmíněných 3573 domén může tyto údaje využít k identifikaci a sledování uživatele na jiných doménách.

Úniky dat však nebyly způsobeny pouze nezodpovědností. Většinu úmyslných úniků dat, které tvořily až 6.1% (1035) z celkového počtu, měly na starosti tzv. Form buildery (aplikace poskytující již připravené formuláře) a různé marketingové machinace.

V prvním případě jsou data poslána Form builderu, který je následně přepošle buď emailem nebo jiným způsobem odpovědným osobám stránky, na které se daný formulář nacházel. Tento případ byl sice považován za úmyslný únik, avšak v praxi se tento postup používá i u legitimních případů, kdy vlastník stránky nemá prostředky k zpracování získávaných dat, takže je předá třetí straně, která se na zpracování těchto dat zaměřuje jako například zpracovatelé osobních údajů.[2]

V druhém případě jdou data marketingovým službám, které jsou náchylnější ke sdílení než předem zmíněné Form builders, jelikož se tyto osobně identifikující informace (informace, které mohou být použity k rozlišení nebo sledování identity uživatele s nebo bez pomoci jiných veřejně dostupných údajů, které mohou být spojeny s určitým jedincem [8]) dají jednoduše využít k cílenému marketingu ať už ve formě reklam na stránkách nebo emailových nabídek. Tam to však nekončí. Jsou společnosti, které si získané identifikační údaje uloží a využijí je k nalezení dalších informací o daném uživateli z veřejně dostupných zdrojů, pomocí kterých vytvoří o uživateli profil. Ten je následně uložen do databáze, která je přístupná všem platícím zákazníkům. Může se tedy stát, že uživatel kontaktuje jednu stránku a na další je již identifikován, aniž by jí předal přímo nějaké informace. Marketingové týmy se následně mohou jednoduše zaměřit na potenciální zákazníky.



Obrázek 3.1: Počet JavaScriptových souborů třetích stran na hlavní stránce (dole) a na kontaktní (nalevo). Převzato z [25]

Pro běžné uživatele to tedy znamená, že po vyplnění nebezpečného formuláře se informace, podle kterých by mohli být identifikováni, mohou dostat do nějaké on-line databáze,

[2] `https://www.gdpr.cz/gdpr/heslo/zpracovatel-osobnich-udaju/`

díky které mohou být rozpoznáni při pouhé návštěvě jiných webových stránek, které si za přístup do této databáze platí.

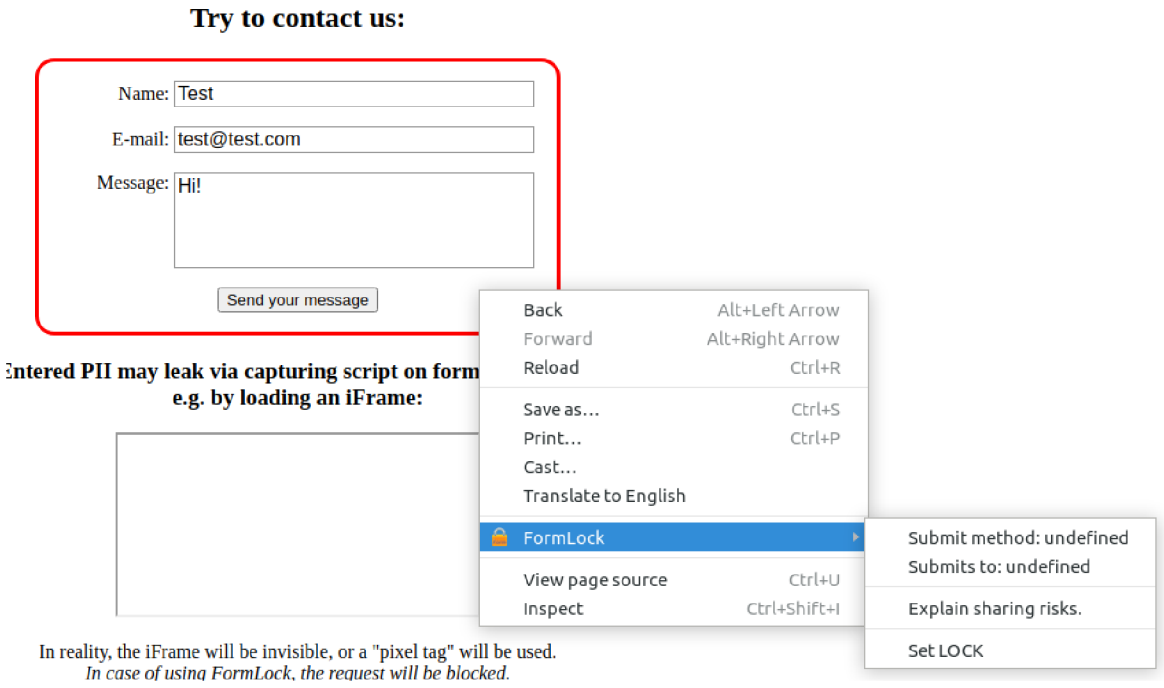
Součástí výsledků byl i empirický pohled na tento fenomén a tak byly zkoumány emaily na testovacích adresách. Bylo zjištěno, že po celém experimentu došlo na testovací adresy 309 emailů od třetích stran, jejichž domény nebyly navštíveny použitým web crawlerem.

Za zmínku také stojí kategorizace stránek, u kterých docházelo k únikům. Ze získaných dat bylo zjištěno, že mezi pravidelné hříšníky patří novinové a mediální stránky, které způsobily jedny z nejčastějších úniků z nedbalosti tak mezi nejčastější úmyslné úniky. Naopak stránky s obsahem pro dospělé se umístily nízko v obou případech úniků.

3.2 Obranná opatření ve Formlocku

Jelikož kontrola interního kódu webových stránek by byla časově náročná a pro obyčejného uživatele webu i obtížná, tak autoři výzkumu přišli s rozšířením Formlock do webového prohlížeče, které by všechny podobné kontroly provedlo za uživatele.

Jako první způsob ochrany uživatele bylo zvoleno zvýraznění formulářů, které by mohly prozradit jejich zadané informace. Formulář je považován za nebezpečný rozšířením Formlock, pokud posílá svá data doménám třetích stran a nebo pokud používá metodu GET. Výběr právě těchto dvou parametrů byl opodstatněn tím, že nezpochybnitelně vystavují poslané informace útočníkům a zároveň jsou lehké k nalezení v HTML kódu, takže uživatel není zavalen chybně identifikovanými hrozbami. Uživatel má taky možnost si prohlédnout jaká rizika provázejí vyplňování daného formuláře.



Obrázek 3.2: Kontextové menu Formlocku na prohlížeči Chromium

Jsou však i situace, kdy by uživatel chtěl i přes bezpečnostní problémy daný formulář odeslat. Únik dat nemusí být vždy způsoben pouze kódem zahrnutým přímo v daném formuláři, ale skripty třetích stran se také mohou pokoušet získat data z formuláře ještě

před tím, než by je uživatel poslal. Těmto situacím se Formlock snaží zabránit pomocí defakto uzamčení prostředí, ve kterém uživatel pracuje. Po zvolení možnosti *Set LOCK* v kontextovém menu jsou blokovány všechny dotazy na domény jiné než je doména v URL stránky, na které se formulář nachází, nebo URL v parametru *action* zvoleného formuláře. Zahrnutí domény z parametru *action* bylo přidáno právě kvůli předem zmíněným Form builderům, bez jejichž povolení by uživatelův formulář nikdy nedorazil ke svému cíli.

Dále si Formlock uloží čas, ve kterém došlo k uzamčení formuláře, aby po jeho odemčení mohl vymazat všechna data o prohlížení včetně cookies, která byla během tohoto časového intervalu uložena v HTML5 uložišti dat. Tímto se autoři snaží bránit proti skriptům, které by si údaje ukládaly a odeslaly je až po odemknutí formuláře, kdy by už dotazy na cizí domény byly povoleny.

V případě že formulář, který chce uživatel odeslat, používá metodu GET, není možné změnit metodu na POST, jelikož by to s největší pravděpodobností rozbilo komunikaci se vzdáleným serverem, který očekával první možnost. Formlock tedy povolí odeslání, ale díky zamčení formuláře budou všechny dotazy na domény třetích stran zablokovány a navíc Formlock odstraní parametry z výsledné URL a znovu načte danou stránku, aby nebylo možné po odemknutí formuláře získat informace pomocí hlavičky referer (hlavička, která obsahuje adresu stránky, ze které byl dotaz poslán) [15] a parametru *location* objektu *document*.

Po odeslání formuláře může uživatel vypnout zamknutí formuláře, což vede k znovunačtení stránky, vymazání všech záznamů v HTML5 úložišti, které vznikly během uzamčení a k ukončení blokování požadavků na domény třetích stran. Tímto Formlock uvede stránku do původního stavu, aniž by nějak výrazně narušoval její primární funkčnost.

3.3 Realizace opatření

Formlock byl implementován autorem primárně pro Google Chrome. Skládá se z jednoho skriptu, který běží na pozadí a několika skriptů, které pracují s obsahem stránky a posílají různé informace skriptu v pozadí pro zhodnocení. Hlavními aspekty implementace jsou:

- Blokování požadavků na domény třetích stran
- Kontrola vlastností formuláře
- Obnovení stránky do původního stavu

Blokování požadavků na domény třetích stran

Tento krok je nutný, aby skripty nemohly komunikovat s třetími stranami během uzamknutí formuláře a tím i vyzradit zadávané údaje například průběžnou kontrolou polí. K tomuto účelu se zde využívá rozhraní WebRequest, které umožňuje přidat funkce typu *EventListener* na jednotlivé fáze HTTP požadavků. Při zamknutí formuláře se uloží URL dokumentu, kde se formulář nachází a případné URL v parametru *action*. Obě jsou následně uloženy do pole *lockDomains*, které následně slouží pro porovnání při odchycení požadavku. Tuto hlavní funkci pro blokování požadavku lze vidět ve výpisu 3.1. Pokud došlo k uzamčení formuláře, tak je zkontrolována URL požadavku a pokud se daná URL nenachází v seznamu povolených domén, tak je požadavek blokován.

```
chrome.webRequest.onBeforeRequest.addListener(
    function(details) {
        var fCancel = false;
        if (lockDomains.length > 0) {
            fCancel = true;
            var current_domain = getRootDomain(getHostname(details.url));
            if (lockDomains.indexOf(current_domain) !== -1) {
                fCancel = false;
            }

            if (fCancel) {
                blocked.push(details.url);
            }
        }
        return {cancel: fCancel};
    },
    {urls: ["<all_urls>"]},
    ["blocking"]
);
```

Výpis 3.1: Zachycení požadavku

Kontrola vlastností formuláře

Informace o formuláři jsou získány tak, že funkce typu *EventListener* v *background.js* čeká na úspěšnou navigaci na novou stránku. Poté provede vložení skriptů, které pracují s obsahem včetně *form_check.js*, ve kterém se také hlavní funkcionalita nachází. Kód ve výpisu 3.2 následně projde všechny dostupné formuláře na dané stránce a přiřadí všem, které používají metodu GET a nebo v atributu *action* mají jiné URL než současného dokumentu, červený rámeček.

```
for (var f = 0; f < document.forms.length; ++f) {

    var current_url = getRootDomain(getHostname(document.forms[f].getAttribute('action')));

    var violation = "";

    if (global_url !== current_url) {
        violation += "> Third-party: " + current_url + "\n";
    }

    if (document.forms[f].method === "get") {
        violation += "> Submit with GET\n";
    }

    if (violation !== "") {
        document.forms[f].style.border = "medium solid red";
    }
}
```

Výpis 3.2: Kontrola formuláře

Obnovení stránky do původního stavu

K obnově dojde poté, co uživatel vybere možnost pro odemčení formuláře. Obsluha kliknutí v souboru *background.js* vypíše URL všech požadavků, které byly během zamčení zabloko-

vány, provede injekci kódu, který odstraní parametry dotazu GET z URL, a vymaže data v HTML5 úložišti včetně cookies.

3.4 Nalezené nevýhody

Testování rozšíření Formlock jsem prováděl manuálně na prohlížečích Firefox verze 84.0 a Chromium verze 87.0 na operačním systému Ubuntu verze 20.04.1. Cílem testování bylo zhodnocení uživatelské přívětivosti a hlavně kontrola funkčnosti klíčových prvků rozšíření jako blokování požadavků na domény třetích stran během uzamčení, zvýraznění nebezpečných formulářů a promazání HTML5 úložiště po odemčení. K testování funkcionality byla primárně využita testovací stránka autorů[3], ale pro úplnost jsem manuálně kontroloval funkčnost jednotlivých opatření i na dalších stránkách, jak je popsáno níže.

Byť byl Formlock implementován primárně pro Google Chrome a tedy i pro prohlížeče založené na projektu Chromium, tak je částečně funkční i na prohlížeči Firefox, který je založený na jádru Gecko.

Už při instalaci Formlocku do obou prohlížečů jsem narazil na první problémy. V souboru *manifest.json* byly špatně pojmenovány soubory s ikonami, kdy podřetězec názvu "lock"byl napsaný s malým l, zatímco název odkazovaného souboru měl dané L velkým písmem. Toto se sice může zdát jako naprostá banalita, avšak Chromium odmítlo rozšíření kvůli této chybě načíst. Firefox byl sice schopen ikony vynechat a načíst rozšíření, ale instalace byla doprovázena několika chybami.

Po opravě názvu souboru jsem pokračoval v testování pomocí technického dema od autorů. Demo obsahuje stránku s formulářem, který používá metodu GET, a skript, který se snaží na pozadí uchovat hodnotu v poli pro email jak pomocí HTML5 úložiště, tak pomocí poslání stejného údaje na doménu třetí strany. Při odeslání formuláře je prohlížeč přesměrován na další stránku, kde jsou vypsány zachycené údaje. Na Chromiu proběhlo všechno v pořádku. Formulář byl opatřen červenou hranou, po zamčení byly blokovány požadavky mířící na doménu třetí strany, bylo zabráněno útoku pomocí hlavičky referer a po odemčení byla vymazána i schovaná data v lokálním úložišti.

Při stejném pokusu na prohlížeči Firefox se však prokázalo mnoho chyb, které silně omezovaly funkcionalitu tohoto rozšíření. Kontextové menu se sice vytvořilo a bylo detekováno, že formulář používá metodu GET, avšak nefungovala žádná vyskakovací okna, která měla informovat o uzamčení formuláře nebo k výpisu bezpečnostních problémů. Při odeslání uzamknutého formuláře sice byly dotazy na třetí stranu zablokovány, ale při odemčení byla zároveň načtena pouze prázdná stránka a při manuálním návratu na demo stránku byl zadaný mail uložen jak v cookies, tak v lokálním úložišti.

Následně jsem s aktivním rozšířením na obou prohlížečích procházel volně internet a navštěvoval typické stránky jako `www.google.com`, `www.facebook.com` a `bookdepository.com`. Při tomto volném testování bylo zřejmé, že rozšíření doopravdy kontroluje všechny formuláře, jelikož jsem náhle viděl některá tlačítka s novými červenými hranami a mnoho vyhledávacích polí bylo také označeno. Toto se dalo očekávat a není to nutně chyba, jelikož to splňuje původní kritéria pro nebezpečný formulář, kterými byly metoda GET a URL v *action* parametru rozdílná než URL dokumentu, kde se nachází. Při tomto testování jsem však narazil na to, že Formlock kontroluje všechny požadavky včetně těch, které nesouvisí se stránkou, na které jsem zamkl formulář. Pokud jsem tedy odemkl formulář na jedné kartě prohlížeče, zatímco jsem měl na dalším otevřený například Reddit, tak mi bylo vy-

[3] `http://anonymous-demo.github.io/`

psáno, že byly zablokovány i dotazy na Reddit, byť stránka s formulářem žádné reference na Reddit neobsahovala. Toto již je chyba, jelikož blokování požadavků stránek na jiných tabech vede k porušení jejich funkcionality včetně nemožnosti danou vedlejší stránku načíst během zamčení formuláře na jiné stránce.

Posledním neopomenutelným faktem je ten, že kontextové menu se vytváří pouze po instalaci, takže pokud zavřete a znovu otevřete rozšíření, tak není dostupné.

3.5 Zhodnocení Formlocku

Formlock slouží jako bezpečnostní rozšíření, které se snaží informovat uživatele a bezpečnostních nedostatcích formuláře, který vyplňuje a pokud se uživatel rozhodne, tak se ho snaží ochránit před vyzrazení osobně identifikujících informací třetím stranám.

Motivací pro jeho vytvoření byl průzkum autorů práce *Are You Sure You Want to Contact Us? Quantifying the Leakage of PII via Website Contact Forms* [25], kteří zjistili, že nezanedbatelný počet stránek ze 100 000 nejnavštěvovanějších obsahoval nějaká bezpečnostní rizika spojená s vyplňováním formulářů, která by mohla vést k identifikaci anonymního uživatele třetími stranami.

I přes předem zmíněné chyby poskytuje Formlock užitečné funkce k ochraně uživatele při vyplňování webových formulářů. Téměř všechna slibovaná funkcionalita je implementována a až na problémy s kompatibilitou je většina nedostatků snadno opravitelná.

Kapitola 4

JavaScript Restrictor

JavaScript Restrictor je webové rozšíření, které funguje jako firewall pro různá JavaScriptová rozhraní. Údržbu provádí Libor Polčák a vývoj probíhá na FIT VUT také formou bakalářských a diplomových prací. Funkční prototyp vyvinul Zbyněk Červinka v rámci své diplomové práce [27]. Další rozšíření funkcionality přišla od Martina Timka (první veřejná verze a podpora Google Chrome a Opera) [26], Pavla Pohnera (Network Boundary Shield)[22], Petera Horňáka (Přenos opatření Chrome Zero) [7] a Martina Bednáře, který přidal automatické testy pro JavaScript Restrictor [3].

Sekce 4.1 slouží k popsání jednotlivých ovlivněných rozhraní a jak se proti zneužití jejich slabin JavaScript Restrictor brání. Sekce 4.2 rozebírá více rozšíření Network Boundary Shield, které stejně jako Formlock pracuje WebRequest rozhraním a sekce 4.3 popisuje, jakým způsobem úrovně ochrany fungují a jak se dají přidávat nové.

4.1 Současné ochranné prvky

Stránky využívající JavaScript mají v současné době k dispozici nemalý počet rozhraní prohlížeče, které mohou nějak využívat. Uživatelé mají bohužel malou kontrolu nad tím, jaká rozhraní budou a nebudou dostupná a jak budou využívána, což může vést k jednoduchému zneužití poskytované funkcionality. JavaScript Restrictor se s tímto snaží bojovat tak, že kontroluje poskytovaná rozhraní a dává uživateli prostředky k jejich omezování.

JavaScript Restrictor tohoto dosahuje pomocí omezení přístupu k JavaScriptovým objektům, funkcím a vlastnostem a nebo poskytnutím méně přesných implementací jejich funkcionalit. Cílem je poskytnout webovým stránkám buď falešná a nebo žádná data bez toho, aby se daná stránka nějak rozbila. K tomuto zapouzdření také dochází před vykreslením stránky, aby žádný z přítomných skriptů nemohl využít zneužitelných funkcionalit a jejich původní podobě.

V současné době umožňuje JavaScript Restrictor modifikaci nebo omezení následujících:

- **metoda window.performance.now():** Slouží k získání objektu časového razítka, které má přesnost v řádu mikrosekund, ale prohlížeče většinou toto číslo zaokrouhlí (Firefox zakrouhluje na milisekundy). JavaScript Restrictor tuto hodnotu zaokrouhluje na čím dál větší řády s rostoucí úrovní zvoleného zabezpečení. Podobná opatření platí u objektů *window.PeformanceEntry* a *window.Date*.

- **HTMLCanvasElement:** Metody z rozhraní Canvas se dají využít k identifikaci uživatele tím, že před načtením stránky skripty využijí Canvas elementu k vykreslení

nějakého textu a pomocí jeho metod jako například *ToDataURL()* získají informace, které jsou závislé na hardwaru uživatele. Restrictor tomu zabraňuje tím, že pomocí úprav nebezpečných funkcí je vždy vrácen pouze bílý obraz.

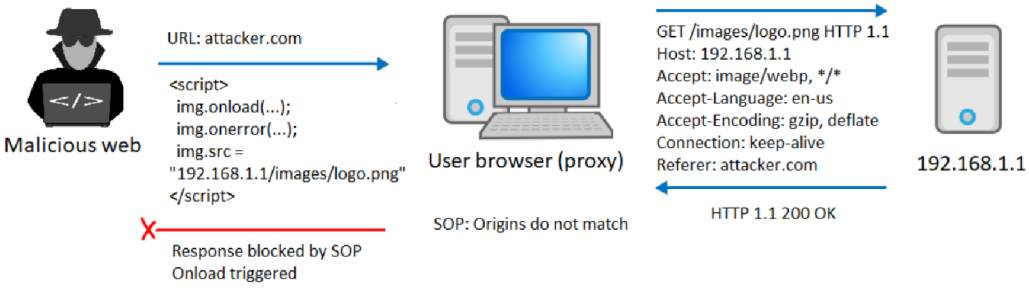
- **navigator.deviceMemory a navigator.hardwareConcurrency:** Tyto atributy obsahují velikost RAM daného zařízení a počet dostupných logických jader procesoru. Zde je obrana implementována podvržením falešných hodnot.
- **XMLHttpRequest:** XHR umožňuje posílání HTTP požadavků od klienta na server. Většinou se používá pro věci jako dynamické získávání obsahu ze serveru, ale posílání požadavků může být i zneužito k posílání identifikujících dat z jiných rozhraní serveru. V současné době brání Restrictor proti tomuto útoku tak, že na nejvyšší úrovni opatření se uživatele dotazuje, jestli mají být jednotlivé požadavky povoleny.
- **ArrayBuffer:** Tyto buffery jsou souvislými bloky dat, které fungují podobně jako pole, ale jsou rychlejší. Bohužel kvůli způsobu jejich alokace prohlížečem se však útočník může dozvědět fyzické adresy a ty využít k mikroarchitekturálním útokům [24]. Zde použitou ochranou je náhodné mapování indexů pole do paměti.
- **SharedArrayBuffer:** Tento objekt je podobný objektu *ArrayBuffer*, ale byl určený pro zrychlení předávání dat mezi běžícími vlákny. Dá se však využít i k časovaným útokům, kdy potřebnou práci odvede do sdíleného bufferu vlákno na pozadí a hlavní vlákno si jej poté lehce přečte [24]. Restrictor se tomuto útoku brání tak, že nepovoluje, aby hlavní skript běžící v rámci stránky mohl přistupovat k *SharedArrayBuffer*.
- **WebWorker:** Web Workers jsou objekty, které umožňují spuštění JavaScriptového kódu na pozadí a na jiném vlákně mimo hlavní. Toto vede k využití vláken běžících na pozadí k časovacím útokům, jelikož nespadají pod stejná omezení jako hlavní vlákno [24]. Implementovaným bezpečnostním opatřením je nahrazení nativního Workera napodobeninou, která není doopravdy paralelní.
- **Geolokační rozhraní:** Restrictor umožňuje uživateli snížit přesnost poskytovanou tímto rozhraní a nebo dokonce jej i přímo zakázat. Také řeší časová razítka, která toto rozhraní poskytuje stejným způsobem jako u výše zmíněných časovačů.

4.2 Network Boundary Shield

Dalším důležitým prvkem JavaScript Restrictoru je také Network Boundary Shield [22], který má za úkol zabránit webovým stránkám na internetu využívat počítač uživatele jako proxy do lokální sítě. V rámci této práce je vhodné se s ním seznámit, jelikož také používá WebRequest rozhraní, které je klíčové pro správné fungování Formlocku.

Motivací pro vytvoření této vrstvy ochrany byla zranitelnost politiky stejného původu. Politika stejného původu slouží k omezení způsobů, jak mohou webové stránky přistupovat k datům jiných webových stránek a aplikací [2]. Zatímco v rámci této politiky jsou zápisy napříč původy povoleny kvůli funkcionalitám jako posílání formulářů nebo používání hypertextových odkazů, tak čtení je převážně zakázáno až na pár výjimek. Mezi tyto výjimky patří případy jako načítání kaskádových stylů, spouštění skriptů a načítání obrázků z jiných stránek. Díky povolení čtení z jiných zdrojů se ulehčí zátěž na server, který nemusí mít všechny pomocné soubory uložené u sebe a zároveň je tedy nemusí neustále stahovat, aby byly aktuální.

Politika stejného původu má však několik nedostatků, které umožňují ji obejít. Network Boundary Shield řeší zneužití, při kterém lze provádět dotazy na zařízení v lokální síti z webové stránky umístěné na internetu. Tento útok využívá faktu, že prohlížeč při porušení politiky stejného původu sice nezpřístupní požadovaný obsah stránce s jiným původem, ale zpřístupní útočníkovi informaci, jestli se požadovaný obsah podařilo načíst a tedy, jestli se na dané adrese a portu opravdu nachází nějaká služba.



Obrázek 4.1: Schéma využití prohlížeče jako proxy. Převzato z [22]

Útočník poté mohl posílat dotazy na běžné privátní adresy (192.168.1.1), získat pomocí nich nejen představu o struktuře lokální sítě, ale mohl také zkoušet posílat požadavky na různé služby na daných zařízeních, jelikož díky tomuto útoku vystupoval pro daná zařízení jako uživatel dané sítě a u některých služeb si mohl potvrdit, jestli byla operace úspěšná například pomocí dotazu na vlastní DNS server.

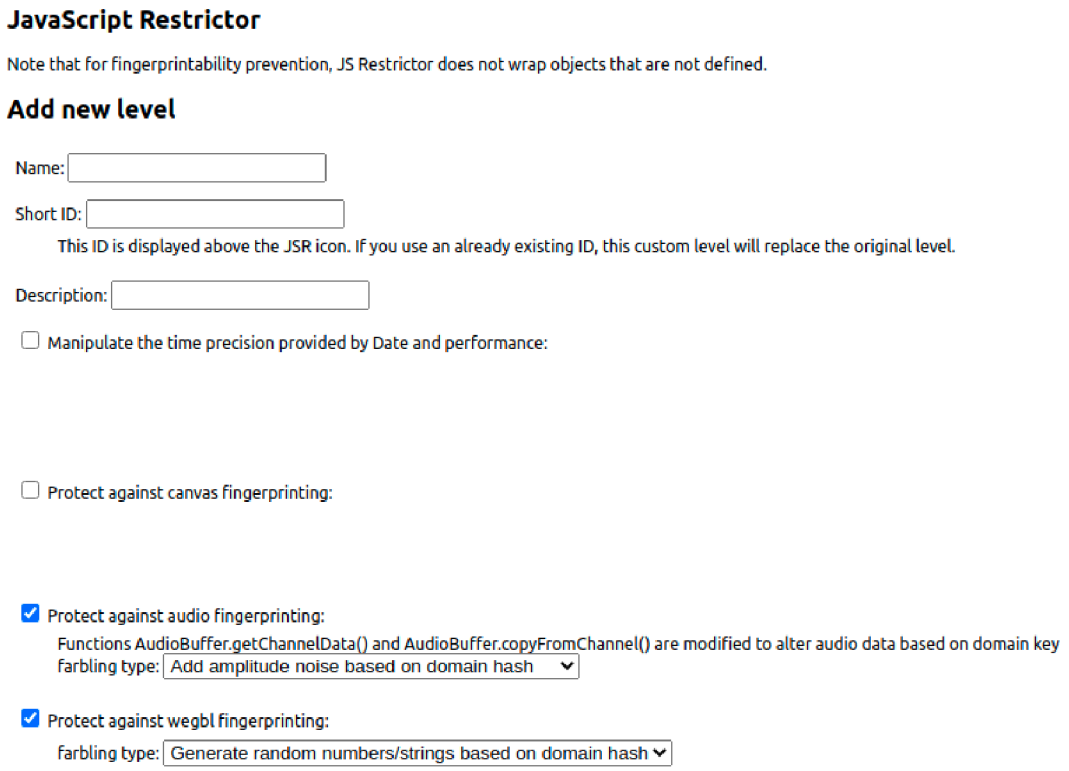
Hlavní myšlenkou ochrany před tímto typem útoků je rozpoznání, jestli je cílová adresa privátní a zdrojová je zároveň veřejná. Na Firefoxu je rozpoznávání adres jednodušší, jelikož je poskytnuto DNS API, zatímco na prohlížečích s jádrem Chromium je současným řešením vlastní DNS cache, která je v JavaScript Restrictoru implementována od verze 0.4.2. Po získání obou adres je následně pomocí regulárních výrazů kontrolováno, jestli se nejedná o komunikaci z veřejné sítě do lokální. Pokud ano, tak je o tom uživatel informován a má možnost blokování daného dotazu.

4.3 Úrovně ochrany

JavaScript Restrictor obsahuje 4 úrovně zabezpečení, které ovlivňují, do jaké míry budou předem zmíněná rozhraní omezována nebo dokonce blokována. Network Boundary Shield je zapnutý na každé úrovni, jelikož dotazy z veřejných adres na privátní jsou vždy podezřelé, ale je možné jej vypnout v nastavení. Uživatel má však také možnost vytvořit si vlastní úroveň ochrany, kterou si může sestavit ze seznamu funkcionalit dostupném v nastavení, který lze vidět na obrázku 4.2. Tyto úrovně zabezpečení se vždy vážou na domény. Uživatel si sice může zvolit základní úroveň zabezpečení, která se bude aplikovat na všechny nově navštívené stránky, ale JavaScript Restrictor si zároveň ukládá změny úrovní na jednotlivých doménách, takže při opětovném navštívení dané stránky se zvolí poslední použitá úroveň.

O fungování nastavení úrovní se starají převážně soubory *options.js* a *levels.js*. Soubor *levels.js* je skript běžící na pozadí, ve kterém jsou ve formátu JSON předdefinované konfigurace základních úrovní zabezpečení a také všechny skupiny tzv. wrapperů (objekt obalující jiný objekt) s popisky, možnostmi ochrany (například varianty přesnosti času u wrapperu, který obaluje rozhraní *Date*) a názvy objektů, které obalují. Dále jsou poskytnuty funkce

pro práci s jednotlivými úrovněmi, uložení nových a získání poslední použité úrovně pro danou doménu. Soubor *options.js* poté slouží jako frontend, pomocí kterého může uživatel s úrovněmi manipulovat.



Obrázek 4.2: Stránka pro vytvoření nové úrovně

Kapitola 5

Návrh integrace a vylepšení

Tato kapitola rozebírá návrh integrace rozšíření Formlock [25] do JavaScript Restrictoru a možná vylepšení jak v podobě úpravy stávajících mechanik, tak v přidání nové funkcionality. Sekce 5.1 rozebírá, jak by současná bezpečnostní opatření v rámci Formlocku mohla být integrována do JavaScript Restrictoru. Sekce 5.2 předkládá návrhy na vylepšení současných opatření a přidání nové funkcionality. Sekce 5.3 pojednává o útocích, které nepřímo souvisí se zaměřením tohoto rozšíření, a jak by se jim případně dalo alespoň částečně předejít.

5.1 Zahrnutí bezpečnostních rozšíření

Vzhledem k tomu, že JavaScript Restrictor již obsahuje některé mechaniky obrany využité ve Formlocku, tak je nutné zvážit, jaká opatření bude vůbec vhodné zahrnout a jaká z již implementovaných půjdou využít s menšími změnami. Nutné je i vzít v potaz, na jakých úrovních opatření budou jednotlivé mechaniky fungovat.

Jedním z opatření, která bude určitě nutné zahrnout je vizuální označení nebezpečných formulářů. Na úrovni 0 by toto opatření mělo být vypnuto, jelikož pokud uživatel plně důvěřuje dané stránce, tak se metoda GET dá považovat za pouhou chybu programátora a rozdílná URL v atributu *action* může znamenat využití předem zmíněných *Form builders*.

Uzamykání formulářů je opatření, které by bylo vhodné umístit od druhé úrovně výš, jelikož první úroveň slouží k minimální úrovni ochrany, takže ji uživatel bude spíš používat na stránkách, kterým důvěřuje. Na stejné úrovně jako uzamykání musí být ovšemže umístěna i detekce formulářů.

Momentálně je tato funkcionalita implementována pomocí zvýraznění daného formuláře červeným okrajem, což sice upozorní uživatele, ale činí tak pomocí změny elementu, tudíž by to mohlo vést k identifikaci Formlocku. Kvůli možnosti identifikace obranných prostředků by bylo vhodné nahradit zvýraznění formuláře nějakou formou notifikací, aby útočník nemohl odhalit ochranná opatření při změně elementu na stránce.

Odemknutí formuláře by mohlo zůstat téměř nezměněné, jelikož pročištění lokálních dat a opětovné načtení stránky by nemělo narušovat ostatní již implementované funkcionality.

Používání kontextového menu pro zamknutí a odemknutí formuláře může být pro uživatele zdlouhavé. Při konzultacích s vedoucím práce jsme se shodli na novém způsobu, který by vyžadoval menší počet uživatelských akcí a využil by již existující vyskakovací okno JavaScript Restrictoru. Uživatel by si v daném okně mohl podobně jako u Network Boundary Shield navolit, jestli chce blokovat požadavky na třetí strany během vyplňování formuláře.

Formlock by následně při události *focusin* na nebezpečném formuláři zkontroloval nastavení Formlocku pro danou stránku a poté zamkl formulář.

5.2 Vylepšení a přidání mechanismů

Formlock byl původně implementován pouze jako tzv. *proof of concept* (implementace konceptuálního řešení, která má nastínit validitu daného návrhu), takže mnoho aspektů zůstalo přehlédnuto a některé myšlenky zůstaly nedokončeny, což naznačuje i nemalý počet poznámek v původním kódu.

Ukládání stavu stránky

Současná implementace neřeší jakkoliv uložení dat, která má stránka uložená na klientské straně. Toto se nemusí zdát zprvu jako velký problém, ale je nutné vzít v potaz, že při vymazávání dat z úložišť, databází apod. v prohlížeči maže i položky, které byly před zadanou časovou značkou vytvořeny a po dané značce modifikovány, místo toho aby byly navráceny do svého původního stavu. Toto může způsobovat problémy od pouhého zdržení kvůli opětovnému získání informací ze serveru hostujícího danou stránku až po narušení funkcionality dané stránky. V momentální verzi jsou také data vymazávána plošně bez omezení na doménu, vůči které bylo provedeno uzamknutí, což může vést k předem zmíněným problémům na všech ostatních stránkách, které bude mít uživatel během zamknutí formuláře otevřené. Tento problém půjde vyřešit přidáním několika argumentů do metody *remove()* rozhraní *browsingData*.

Obnovení dat do stavu před uzamknutím formuláře by mohl obstarat vložený skript do dané stránky, který by na pokyn od skriptu běžícího na pozadí načetl stavy všech důležitých úložišť. Načtené informace by byly zaslány skriptu na pozadí, který by si je uložil a následně je poslal zpět, aby mohla být data obnovena po odemknutí formuláře a opětovném načtení stránky.

Kompatibilita s prohlížečem Firefox

Z testování v kapitole 3 je jasné, že bude nutné stávající implementaci upravit, jelikož nefunguje na prohlížeči Firefox. Toto je důležitý aspekt, jelikož JavaScript Restrictor funguje i na prohlížeči Firefox a bylo by nevhodné vynechávat funkcionalitu z jednoho z nejpoužívanějších prohlížečů. Problémy s nekompatibilními nebo přímo chybějícími rozhraními půjde jednoduše vyřešit umístěním kódu z těchto problematických částí do jednotlivých skriptů na pozadí, které by následně poskytovaly funkce společnému skriptu na pozadí. Tento postup byl využit například v bakalářské práci Petera Horňáka, který do JavaScript Restrictoru integroval funkcionalitu z rozšíření Chrome Zero [7].

Změna detekce formulářů

Dalším vážným aspektem ke zvážení je způsob, jakým by měly být detekovány potenciálně nebezpečné formuláře. Současná metoda je sice rozumná, ale při její aplikaci v praxi může být pro uživatele nepřívětivá, jelikož se svými současnými kritérii detekuje i bezpečné formuláře, které slouží jako například formulář na obrázku 5.1 k vyhledávání apod. Obzvlášť otravný by tento způsob vyhodnocování byl v kombinaci s předem zmíněným nutným převodem zvýrazňování formulářů na notifikace, jelikož by byl člověk zahrnut upozorněními na každé stránce, která by používala formulář pro zadávání položek k vyhledávání.

Bylo by tedy vhodné vynechat všechny formuláře, které mají roli *search* a zároveň formuláře, které obsahují pouze prvky jako tlačítka apod., jejichž hodnota by při případném úniku měla zanedbatelnou hodnotu.

Dálé by bylo vhodné vylepšit i způsob, jakým je uživatel na dané formuláře upozorněn. Stránka může obsahovat několik formulářů, takže by si uživatel nemusel být jistý, o jaký se jedná. Zároveň by bylo vhodné uživatele informovat pouze v případě, že chce daný formulář využít. Detekce formulářů by tedy mohla být, podobně jako je zamknutí v předchozí sekci, spojena s událostí *focusin* na formuláři, takže k upozornění by došlo až, když by uživatel klikl na nějaké políčko daného formuláře.

```
1 <form action="/search/" autocomplete="off" method="get" role="search">
2     <input type="search" id="header-search-bar" name="q" class="_2xQx4j6lBnDGQ8QsRnJEJa"
          placeholder="Search" value="">
3 </form>
```

Výpis 5.1: Formulář vyhledávání na stránce Reddit.com

5.3 Nepřímé zneužití formulářů

K zvážení při návrhu nových funkcionalit je také otázka, jestli by měla být přidána ochrana proti nepřímému úniku dat pomocí formulářů. Příkladem tohoto útoku je zneužití automatického doplnění přihlašovacích údajů pomocí neviditelného formuláře a následné poslání údajů třetím stranám [1]. Podstata tohoto napadení spočívá v tom, že člověk vyplní svoje přihlašovací údaje na přihlašovací stránce, kde se žádný škodný kód nenachází, prohlížeč mu následně nabídne uložení těchto údajů pro budoucí použití a na další stránce (kam se uživatel dostane pomocí přesměrování nebo manuálně) je neviditelný přihlašovací formulář, který je automaticky vyplněn prohlížečem a z něho jsou následně odeslány informace třetí straně.

Rozhodl jsem se tento problém v rámci této práce neřešit na základě několika teoretických a praktických důvodů. Jedním z těchto důvodů je fakt, že různé prohlížeče vyvolávají událost, na základě které by automatické vyplnění šlo zachytit, různě a některé vůbec. Rozdíly však nejsou pouze mezi prohlížeči samotnými, ale i mezi jednotlivými verzemi daných prohlížečů. Pro spolehlivé odchycení této události by tedy bylo nutné prostudovat hlouběji současné implementace a napsat vlastní polyfill (kód, který by tuto funkcionalitu doplnil na prohlížečích, které ji neposkytují), jelikož v tento moment existující polyfilly jsou několik let staré a neudržované. Ani přímé blokování odeslání formuláře, dokud by nebylo vyhodnoceno, jestli je skrytý a chce přihlašovací údaje, by nepomohlo, jelikož formuláře by stejně jako v uvedeném demo příkladu[1] nemusely být odesílány vůbec. Útočník by mohl periodicky kontrolovat jejich obsah a po vyplnění poslat údaje třetí straně. Ochrana proti této metodě by se tedy stala nejen závislá na čase (což není spolehlivé), ale zároveň by bylo nutné zasahovat do obsahu, což by JavaScript Restrictor prozradilo.

Jako alespoň částečné řešení tohoto problému by bylo možné filtrovat požadavky na domény poskytující tyto služby (v době psaní zmíněného článku byly pouze dvě), což by však už nebylo přímo zodpovědností Formlocku, ale spíše wrapperu obalujícího XHR požadavky.

[1] `https://senglehardt.com/demo/no_boundaries/loginmanager/`

Kapitola 6

Implementace

Tato kapitola rozebírá, jak funguje implementace nových bezpečnostních opatření a popisuje rozdíly v částech, kde se kód liší kvůli rozhraním poskytovaným jinými prohlížeči. Sekce 6.1 probírá změny v souborech, které byly nutné pro integraci Formlocku. Sekce 6.2 je zaměřena na postupy, pomocí kterých byly jednotlivé druhy úložišť zálohovány a následně obnoveny. Sekce 6.3 shrnuje vylepšení spojená se zamykáním formulářů. Sekce 6.4 zmiňuje za jakých podmínek jsou nově použita bezpečnostní opatření a jak kód bere v potaz úroveň bezpečnosti. Sekce 6.5 vysvětluje nová kritéria pro identifikaci potenciálně nebezpečného formuláře.

6.1 Integrace do JavaScript Restrictoru

Implementace byla od počátku realizována jako rozšíření JavaScript Restrictoru. Aby byl průběžný vývoj transparentní a nově přidaný kód mohl být kontrolován a následně opraven, tak byl vytvořen Pull request v repositáři projektu na GitHubu[1].

Integrace kódu z Formlocku probíhala bez problému a v průběhu nevznikaly žádné funkční konflikty s jinými komponenty JavaScript Restrictoru, avšak kromě pouhého přidání jmen skriptu do souboru *manifest.json* bylo i nutné deklarovat nová práva, aby přidaný kód mohl využívat rozhraní, která předtím nebyla dostupná. Těmito přidanými právy byly *contextMenus* (ve Firefox verzi pouze *menus*), které slouží k práci s rozhraním pro kontextová menu, *browsingData*, které zpřístupňuje rozhraní pro manipulaci s daty vzniklými během prohlížení stránek a je nutné pro realizaci mazání těchto dat, *webNavigation*, které bylo použito pro vyhodnocení nově načtené stránky, což rozhodlo nejen o vytvoření kontextového menu, ale také o vložení skriptů do dané stránky a právo *cookies*, které bylo využito pro práci s cookies daleko sofistikovanějším a efektivnějším způsobem než by bylo možné prací s atributem *cookies* objektu *document*.

Při integraci bylo také nutné přejmenovat a vytvořit několik nových souborů. Původní soubor *background.js* byl kvůli adekvátnějšímu pojmenování a možnému vytknutí kódu rozdělen na 3 soubory: *formlock_common.js*, který obsahuje hlavní funkcionalitu a soubory *formlock_chrome.js* a *formlock_firefox.js*, které poskytují funkce spravující části kódu, které jsou specifické pro dané skupiny prohlížečů. Dalším přidaným souborem byl *data_backup.js*, který je *content script* a stará se o zálohu dat z dané stránky při zamčení a jejich obnovou po odemčení.

[1] `https://github.com/polcak/jsrestrictor/pull/94`

Posledním aspektem nutným pro řádnou integraci bylo přidání nového kódu do systému úrovní obrany a dát uživateli možnost funkcionalitu Formlocku přidat nebo vynechat z vlastních nadefinovaných úrovní ochrany. K přidání funkcionality do předem definovaných úrovní stačilo modifikovat proměnné *level_2* a *level_3* v souboru *levels.js*. Pro přidávání Formlocku do uživatelsky vytvořených úrovní stačilo přidat jedno políčko k zaškrtnutí do souboru *options.js*. Byť jsou v tomto souboru všechny možnosti generovány automaticky na základě definovaných wrapperů v souboru *levels.js*, tak jsem se rozhodl tuto možnost dát zvlášť, jelikož Formlock neobaluje žádné rozhraní.

6.2 Ukládání a obnova dat

K obnově dat stránky po odemknutí formuláře slouží již předem zmíněný skript *data_backup.js*. Tento *content script* je umístěn na každou stránku a očekává zprávu *BackupStorage* od skriptu *formlock_common.js*, který ji vyšle při zamknutí formuláře. Při přijetí této zprávy jsou zazálohována data z *localStorage* a *sessionStorage* dané stránky.

Následně je zavolána funkce *backup_databases* pro zálohu všech databází, která je však dostupná pouze ve verzi pro Chromiové prohlížeče, jelikož ty na rozdíl od svého protějšku Firefoxu obsahují v tento moment experimentální metodu *indexedDB.databases*, která vrací seznam jmen všech databází spojených s touto stránkou a jejich verze. Všechny následující informace ohledně zálohování databází z *indexedDB* jsou tedy relevantní pouze pro Chromium verzi. Při zálohování databází bylo nutné vzít v potaz to, že rozhraní *IndexedDB* sice je asynchronní, ale používá callback funkce navázané na události DOM místo mechaniky *promises*, což není úplně vhodné pro tento případ, jelikož při asynchronním ukládání jednotlivých objektů v databázi by se mělo počkat na dokončení všech dílčích operací. Toto čekání by bylo chaotické, kdybychom museli volat několik zanořených asynchronních funkcí, což bylo vyřešeno obalením těchto volání do *promises*. Takže například místo toho, aby funkce *load_db_contents* pouze zavolala asynchronní funkce pro zálohování jednotlivých *object stores* a ukončila svoje provádění, tak nyní zavolá tyto funkce a až po všech jejich úspěšných či neúspěšných provedeních, obalující *promise* se vyhodnotí a dá tak volající funkci *backup_databases* najevo, že se všechny dílčí operace ukončily.

Záloha databází byla implementována tak, aby se nezastavila neúspěchem dílčí operace. Pokud se nepodaří uložit data z nějakého *object store* a nebo databáze, tak se pokračuje dál a mechanika *reject* v *promises* je tedy používána pouze pro indikaci chyby, o které je uživatel informován obecnou zprávou o neúspěchu. Upozornění na inkonzistenci uložených dat obdrží uživatel na konci zálohy. Všechna uložená data jsou poslána ve formátu JSON do skriptu na pozadí, který je po odemknutí pošle zpět.

Mezi data uložená skriptem však nepatří cookies, jelikož k jejich záloze bylo využiro *Cookies* rozhraní, které poskytuje WebExtensions. Jejich záloha a obnova je řešena ve funkcích *backup_cookies* a *restore_cookies*, které mají svoje verze jak pro Chromium, tak Firefox, jelikož Firefoxová verze funguje pouze pomocí *promises*.

Obnovení dat prochází stejně jako jejich záloha, až na to, že databáze jsou vymazány a opět vytvořeny. Jejich vymazání je nutné, protože promazání dat pomocí rozhraní *browsingData* způsobí i to, že *object stores*, jejichž hodnoty byly během zámku upraveny, se celé vymažou. Pro obnovu těchto *object stores* by však bylo nutné vyvolat událost aktualizace dané databáze, což však není možné, pokud danou databázi neotevřeme s vyšší verzí. Avšak otevření s vyšší verzí by mohlo způsobit rozbití skriptů stránky samotné, protože by se mohly pokoušet otevřít dané databáze s menší verzí, což vede k chybě, a nebo by

nebyly řádně aplikovány aktualizace při nové verzi. Databáze jsou tedy vymazány a znovu vytvořeny se stejnou verzí, aby se nerozbil kód na straně klienta.

6.3 Úprava mechanismu uzamykání

V původní implementaci bylo blokování požadavků na domény třetích stran řešeno tak, že funkce, která naslouchala fázi požadavku *onBeforeRequest*, pouze zkontrolovala, jestli je momentálně aktivní zámek a jestli daný požadavek míří na doménu třetí strany. Tento způsob však nebral v potaz (jak i autor v komentářích původního kódu upozornil) požadavky, které byly posílány v rámci jiných tabů prohlížeče. Zároveň se při testování projevilo, že na prohlížeči Chromium byla tato kontrola vyvolána i při požadavcích, které byly v rámci rozšíření samotného (požadavek na získání ikony uložené ve složce). Do funkce tedy byla přidána kontrola, jestli se požadavek neprovádí v rámci jiného tabu a také jestli požadavek nesouvisí s rozšířením. Tato změna však umožňuje potenciální únik dat v případě, že by uživatel měl danou stránku otevřenou ve dvou tabech. Stránka na uzamčeném tabu by mohla ukládat údaje například do *localStorage*, což by je zpřístupnilo i stránce na nezamčeném tabu. Stránka by následně z nezamčeného tabu mohla poslat informace třetí straně. Přesto jsem se rozhodl tuto změnu ponechat, protože značně zlepšuje uživatelovu zkušenost s rozšířením a dle mého názoru není tato situace příliš častá. Aby daná situace mohla nastat, tak by uživatel buď musel mít otevřené dva taby, které by byly na stejné stránce, nebo by druhá otevřená stránka musela periodicky posílat data z formuláře třetí straně. I přes dle mého názoru malou pravděpodobnost tohoto jevu by bylo vhodné při pokračování práce vzít tuto situaci v potaz a adekvátně upravit návrh.

Při zamykání formulářů již není nutné používat kontextové menu. Místo toho je při detekci uživatelova kliknutí na potenciálně nebezpečný formulář poslána zpráva skriptu na pozadí, která zkontroluje, jestli doména, na které momentálně uživatel je, není v seznamu stránek, kde uživatel povolil požadavky na domény třetích stran během vyplňování formuláře. Pokud se doména stránky na tomto listu nachází, tak není umístěn zámek na daný formulář.

Nově byla také přidána kontrola nově otevřených tabů během uzamknutí formuláře. Tato kontrola je realizována ve funkci naslouhající události *created* rozhraní *tabs*, která zavře jakékoliv nové taby ve fázi načítání, které měly jako svého původce tab, na kterém je momentálně uzamčený formulář.

Situace, kdy by se na stránce nacházelo více formulářů a uživatel se rozhodl kliknout na formulář, zatímco jiný na stejné stránce byl zamknutý byla vyřešena tak, že se zámek přemístí na nově zvolený formulář a neprovede se nová záloha dat. To reálně znamená, že v uzamknutých doménách se pouze změní doména z atributu *action*.

6.4 Reakce kódu na změnu úrovně

Reakce na změnu bezpečností úrovně JavaScript Restrictor probíhá převážně ve funkci *handle_tab*, která analyzuje informace získané naslouháním události *completed* rozhraní *webNavigation*. Vždy po navigaci na novou stránku se získá úroveň, jaká je pro danou doménu uložená. Prvně se na základě úrovně rozhodne, jestli se vytvoří kontextové menu, což řeší funkce *decide_context_menu*, a poté se rozhodne, jestli by se měly vůbec vložit skripty na danou stránku. Pokud Formlock není definovaný v dané úrovni, tak se ještě provede kontrola, jestli uživatel nezměnil úroveň během zámku na nějakou, která Formlock

neobsahuje. Pokud Formlock byl definovaný v této úrovni na této stránce, tak se ještě provede kontrola, aby se kód nepokoušel vložit skripty do interních stránek prohlížeče (např. *chrome://version*), což vede k vyvolání chyby. Pokud daný požadavek prošel, tak jsou do dané stránky vloženy skripty pro reagování na uživatelské akce a pro detekci potenciálně nebezpečných formulářů.

Pro kontrolu úrovně při přepínání mezi jednotlivými taby byla přidána funkce typu *EventListener*, která volá funkci *decide_context_menu* při každé změně aktivního tabu.

6.5 Změna detekce formulářů

Nová implementace detekce formulářů v souboru *form_check.js* nebere jako nebezpečné formuláře, které obsahují pouze prvky *input* s typem bez užitečné informační hodnoty. Jako tyto typy byly vybrány *submit*, *reset*, *button*, *color* a *hidden*. Typy *submit*, *reset*, *button* a *hidden* nemají samy o sobě žádnou informační hodnotu a hodnota typu *color* je pro identifikaci uživatele irelevantní. Pro kontrolu *input* prvků formuláře je určena funkce *check_element*, která zkontroluje, jestli předaný prvek není *input* a pokud je, tak jestli nepatří mezi typy, které nejsou bezpečné. Následně je rekurzivně zavolána na všechny děti daného prvku.

Kapitola 7

Testování

Tato kapitola popisuje postup, který jsem zvolil při testování nové implementace na reálných stránkách, dále popisují jaké poznatky jsem si z tohoto testování odnesl a také jak byly případné chyby zhodnoceny a eventuálně opraveny. Také je popsána vlastní testovací stránka s odůvodněním jejího vytvoření a z ní získaných poznatků.

Manuální testování

Cílem manuálního testování bylo zjistit, jestli během trvání zámku na formuláři neunikají žádná data nepovoleným třetím stranám. Při testování jsem se rozhodl držet následujícího postupu na každé testované stránce:

1. Uzamknutí formuláře
2. Zapnutí zaznamenávání HTTP požadavků
3. Vyplnění a odeslání formuláře
4. Kontrola záznamu HTTP požadavků

Zvolil jsem tento postup, protože se dá předpokládat, že pokud budou během vyplňování formuláře zablokovány všechny požadavky na domény třetích stran a před opětovným povolením těchto požadavků budou všechna data vytvořena po aplikaci zámku smazána, tak útočník nebude mít žádný způsob, jak data z formuláře odeslat třetím stranám. Za bezpečné vyplnění formuláře se tedy dá považovat situace, kdy během vyplňování formuláře neodešly žádné požadavky na domény, které nesouvisí s parametrem *action* nebo URL dané stránky a kdy nebyly otevřené žádné nové taby skriptem na dané stránce a URL nově otevřené stránky neobsahuje informace z daného formuláře.

Při výběru stránek jsem se řídil zejména návštěvností daných stránek. Ale kromě výběru stránek z českých[1] a zahraničních[2] zdrojů jsem přidal i některé osobně navštěvované stránky – jako například `www.bookdepository.com` nebo `www.pijumate.cz`. Při výběru z žebříčku českých stránek jsem zvolil stránky, které měly udávanou návštěvnost alespoň v řádu desítek tisíc za den.

I když studie, na základě které byl původně vytvořen Formlock [25], se zaměřila pouze na kontaktní formuláře, rozhodl jsem se zahrnout i jednoduché formuláře, jako například

[1] `https://toplist.cz/`

[2] `https://www.digitaltrends.com/web/top-100-websites-how-are-they-tracking-you/`

formuláře k přihlášení k on-line newsletteru, neboť únik emailové adresy by mohl být zneužit k posílání spamu na danou adresu, nebo by se útočník mohl pokusit danou adresu napadnout. Testoval jsem i formuláře, které neslouží přímo ke kontaktování stránky, ale mohou sloužit jako deterministický identifikátor při budování profilu uživatele.

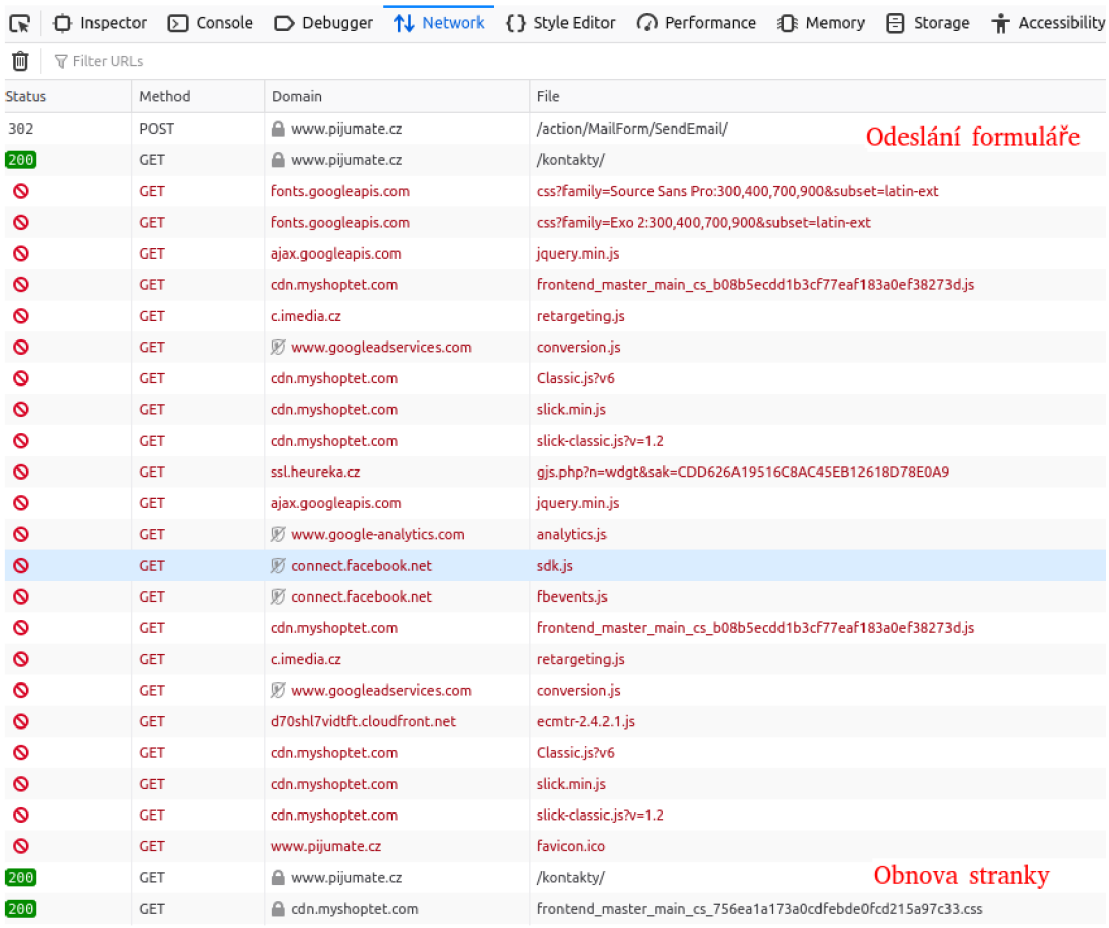
Při procházení testovaných stránek jsem zjistil, že většina z těchto stránek měla svoje formuláře napsané tak, že je Formlock neoznačí za nebezpečné. To však ještě neznamená, že se zadávanými údaji nemohou manipulovat reklamní skripty přítomné na daných stránkách. Rozhodl jsem se tedy ponechat kontextové menu pro případ, pokud by uživatel chtěl blokovat požadavky na domény třetích stran, i když Formlock neoznačil formulář za potenciálně nebezpečný. Uživatel tedy může zamknout formulář manuálně, popřípadě automaticky při kliknutí na něj, pokud byl daný formulář dle nových kritérií označen jako potenciálně nebezpečný. Rozhodl jsem se tedy pro opětovné zavedení kontextového menu, protože uživatele tato možnost neobtěžuje (jako by například vyskakovací okna), a protože je preciznější než povolení posílání požadavků na URL všech *action* parametrů na stránce, které by mohlo být zneužito umístěním neviditelného formuláře na stránku.

Výsledky manuálního testování jsou umístěny na paměťovém médiu, celkově bylo otestováno 22 stránek. Jednotlivé testy jsou umístěny ve složce *manualni_testovani*, kde je každý soubor pojmenován ve formátu `testN-nazevdomeny.format`, kde `N` je pořadové číslo testu a `format` je formát souboru. Z počátku jsem výsledky síťové komunikace ukládal ve formátu *.har*, ale výsledky měly velikosti až 15MB, nebyly přehledné a někdy chyběly záznamy o zablokovaných požadavcích, nebo byly označené jako *finished*. Přesto jsem se rozhodl již existující záznamy ponechat, protože z nich lze vyčíst prodlevu mezi odesláním dat formuláře a obnovením stavu stránky. Od testu č. 7 jsem použil snímky obrazovky z vývojářských nástrojů jednotlivých prohlížečů. Jako příklad uloženého snímku slouží obrázek 7.1. Tyto snímky obrazovky jsou doplněny krátkými popisky, které označují, kdy se odeslal formulář a kdy došlo k obnově stránky, aby čtenář lépe pochopil posloupnost událostí a nemusel záznamy dlouze zkoumat.

Složka obsahuje pouze 20 záznamů, neboť na dvou testovaných stránkách nedošlo k žádné síťové komunikaci. Tyto stránky se před odesláním formuláře pokoušely otevřít nový tab, což Formlock nepovoluje.

Testování jsem prováděl na prohlížečích Chromium verze 90.0.4430.93 a Firefox verze 88.0.

Při testování jsem zjistil, že většina formulářů byla odeslána v pořádku a během zámku byly blokovány požadavky na všechny nepovolené třetí strany. Blokování požadavků na domény třetích stran proběhlo úspěšně i v případě formulářů, které se nepodařilo odeslat.



Obrázek 7.1: Blokování požadavků na stránce `pijumate.cz`

Z manuálního testování jsem získal následující poznatky:

- Některé stránky při načtení okamžitě umístí *focus* na vyhledávací formulář (ekvivalent uživatelova kliknutí na formulář). Pokud je daný formulář vyhodnocený Formlockem jako nebezpečný, tak je na něj okamžitě umístěn zámek a uživatel může mít problémy s používáním dané stránky, protože jsou požadavky omezené na URL dané stránky a *action* parametr vyhledávacího formuláře. Pokud by se uživatel pokusil daný formulář odemknout, tak by po načtení stránky byl opět umístěn *focus* na daný formulář. Implementoval jsem tedy do kontextového menu možnost *Unlock without refresh* umožňující uživateli uvolnit zámek bez nutné obnovy celé stránky.

- Nelze odesílat formuláře, které vyžadují na stejné stránce potvrzení *captcha* od třetích stran. Nejčastěji jsem nalezl mechanismus potvrzení poskytovaný společností Google. Tento problém spočívá v návrhu obranných opatření. Pro řešení tohoto problému by bylo nutné přehodnocení části návrhu, které se zabývá blokováním požadavků na domény třetích stran. Povolení dotazů na Google *captcha* by sice uživatelovi zpříjemnilo používání rozšíření a umožnilo používat dané formuláře, ale také by to bylo v rozporu s ochranou dat, neboť bychom povolovali další třetí straně potenciální přístup k jeho informacím.

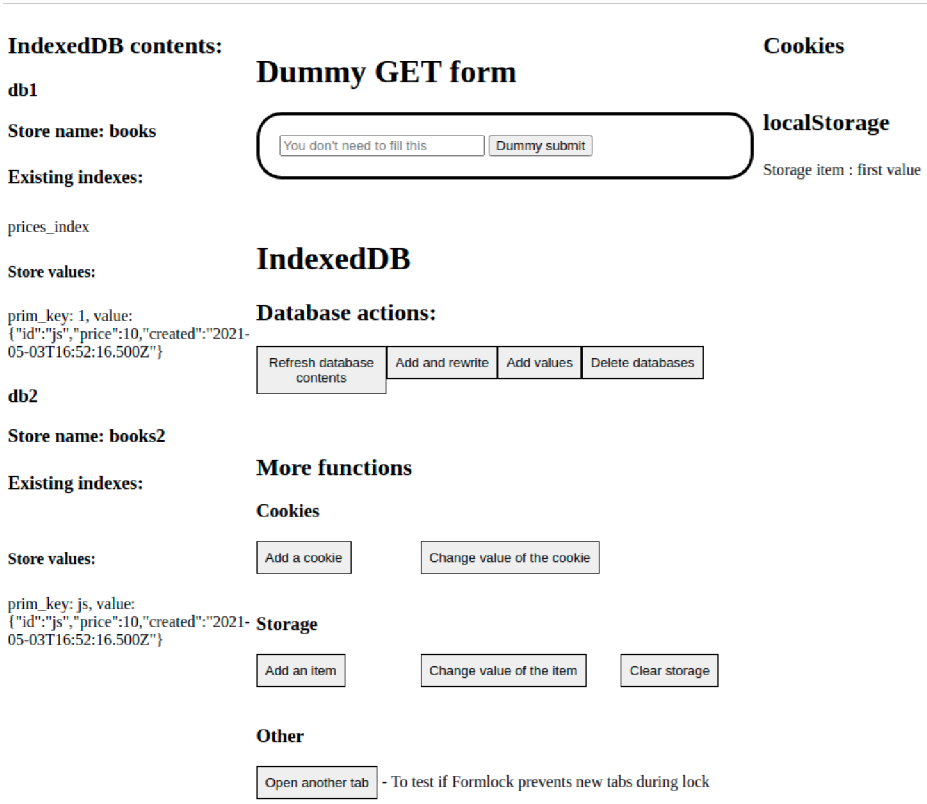
- Formuláře nemusí být vždy využívány ke svému hlavnímu účelu. Setkal jsem se s několika stránkami, kde nedošlo k požadavku POST nebo GET souvisejícímu s posláním formuláře, ale místo toho byly využity XHR požadavky. Úspěch poslání informací tedy záleží, zda autoři posílají požadavky třetí straně nebo ne. Tento problém se projevil na stránce `https://www.belowthefold.news/`. Formulář na této stránce byl nastaven s metodou POST a *action* parametr obsahoval URL třetí strany. Stránka se po kliknutí na odeslání pokoušela otevřít nový tab a následně byly vyvolány pouze požadavky na neznámé třetí strany. Po odeslání formuláře jsem očekával požadavek na doménu v parametru *action*, ale žádný jsem v zachycené síťové komunikaci nenalezl. Tento problém, bohužel, nelze řešit automatizovaně, jelikož není možné vyhodnotit zda požadavky posílají data validním cílům anebo marketingovým serverům.

Testovací stránka

Při testování v praxi by bylo příliš obtížné hledat stránky, které by se pokoušely o zneužití všech krajních případů použití, proto jsem si pro účely testování teoretických útoků vytvořil jednoduchou testovací stránku, viz. obrázek 7.2. Tato stránka se zaměřuje hlavně na kontrolu funkčnosti zálohy a následné obnovy dat, neboť stránky v praxi mohou zaplňovat úložiště velkým množstvím dat, které se těžce sleduje. Na útoky pomocí posílání dat třetím stranám se tato testovací stránka nezaměřuje, tento druh testování je již přítomný na stránce autorů původního Formlocku.[3] Při testování jsem tuto stránku umístil na server na lokální adresu pomocí Node.js s balíčky *Express* a *Nodemon*. Při otevření HTML souboru stránky (*test.html*) pomocí prohlížeče bez využití serveru budou stále fungovat všechny vlastnosti až na přidávaní a změnu cookies.

Stránka obsahuje jednoduchý formulář, který slouží pouze k uzamknutí a údaje neposílá žádné třetí straně. Po levé straně se nachází sloupec, který slouží k výpisu všech databází a jejich *object stores* včetně obsahu a jmen jednotlivých indexů, které k nim patří. Podobně slouží pravá strana pro výpis obsahu cookies a *localStorage*. Střed kromě formuláře obsahuje i ovládací prvky k databázi, například přidání obsahu, aktualizace výpisu anebo vymazání databází. Dále se uprostřed nachází podobné ovládací prvky pro cookies a *localStorage*. Dole se nachází tlačítko pro otevření nového tabu. Jedním z možných vektorů útoku, na který upozorňoval i autor původního Formlocku v komentářích, je otevření nového tabu, kterému by mohly být předány informace k odeslání. V původní implementaci Formlocku byl sice tento způsob útoku nemožný, jelikož byly blokovány požadavky na všech tabech, ale v nové implementaci by teoreticky mohl být možný útok, kdy by skript umístil údaje například do *localStorage*, aby k nim mohla mít přístup stejná stránka otevřená v jiném tabu a mohla je odeslat třetím stranám. O tento útok se testovací stránka nepokouší, slouží pouze ke kontrole, aby se zabránilo testovací stránce otevřít nové taby v průběhu zámku.

[3] http://anonymous-demo.github.io/



Obrázek 7.2: Jednoduchá testovací stránka

Tuto stránku jsem používal průběžně při vývoji nové implementace a pomohla mi odhalit problémy převážně s rozhraním *IndexedDB*. Také jsem při testování na této stránce získal následující poznatky:

- Rozhraní *browsingData* na prohlížeči Firefox při mazání dat *IndexedDB* ignoruje argument *since* (aniž by o tom byla uvedena informace v dokumentaci), který udává časový okamžik (od kterého mají být data odstraněna) a maže jednoduše data všechna. Tento problém nelze v této době implementačně vyřešit, neboť Firefoxové rozhraní *IndexedDB* neobsahuje metodu pro výpis jmen databází, tudíž není možné je jakkoliv zálohovat pro budoucí obnovu.

- Implementace pro prohlížeče Chromium neumožňuje uživateli otevřít nový tab během zámku, pokud je aktivní tab zamknutý. Z jiných tabů je to možné. Tato situace nastává, protože implementace rozhraní *tabs* na Chromiových prohlížečích udává číslo otevírajícího tabu (tabu, ze kterého přišel požadavek na otevření) i v případě, že nový tab byl otevřen uživatelem a ne skriptem z daného tabu. Tento problém byl částečně vyřešen upozorněním uživatele v dané situaci.

- Obnova dat z *IndexedDB* může selhat, pokud se o obnovení dat skript pokusí hned po načtení stránky. Při hledání příčiny tohoto problému jsem zjistil, že v takovýchto situacích může připojení k databázi uváznout bez vyvolání jakékoliv události definované v dokumentaci. Obnovu databází jsem tedy přesunul před obnovení stránky před odemknutím. Tento přesun by neměl vytvářet jakékoliv nové příležitosti pro uložení dat,

jelikož rozhraní *IndexedDB* není kvůli své časové náročnosti vhodné pro operace s vysokou frekvencí (například skript, který by každý zlomek sekundy chtěl uložit údaje z formuláře). Implementaci pro Firefox také tímto způsobem nelze obejít, neboť kvůli chování zmíněnému v prvním bodě jsou vymazána všechna data a implementace pro Chromiové prohlížeče musí kvůli udržení konzistence všechny databáze před obnovou vymazat. Žádná data uložená po záloze vytvořené při zamčení tedy nepřežijí.

Kapitola 8

Závěr

Díky získaným zkušenostem z této práce jsem se naučil jak tvořit rozšíření prohlížeče, včetně toho jak fungují interně a jaké funkcionality poskytují vývojářovi prostřednictvím rozhraní. Tyto poznatky rozebírá kapitola 2 včetně rozdílů mezi prohlížeči a přicházejícími změnami.

Dále jsem se v kapitole 3 zaměřil na únik osobních údajů z formulářů, který byl také zkoumán v rámci studie ze Stony Brook University [25] a kvůli kterému bylo vytvořeno autory studie rozšíření Formlock. Rozšíření Formlock jsem důkladně otestoval a sepsal všechny problémy včetně problematické kompatibility s prohlížečem Firefox.

Následně jsem se v rámci kapitoly 4 obeznámil s fungováním rozšíření JavaScript Restrictor, jeho ochrannými prvky, částmi, které by mohly ovlivnit i budoucí integraci Formlocku a úrovněmi ochrany, abych mohl rozhodnout, do jakých úrovní by nově přidaná funkcionalita měla patřit.

Integraci Formlocku do JavaScript Restrictoru a vylepšení obranných prvků jsem navrhl v kapitole 5. V rámci této kapitoly jsem zároveň nastínil problematiku nepřímého úniku dat pomocí formulářů, její relevantnost k této práci a časové a implementační aspekty, které znemožnily řešení těchto úniků v rámci této práce.

Implementační detaily jsem popsal v rámci kapitoly 6. Kromě doladění a integrace Formlocku do JavaScript Restrictoru jsem také implementoval zálohu a obnovu dat stránek, kde bylo použito zamčení formuláře. Díky této záloze byl minimalizován dopad zámku na běžné používání dané stránky. Dále jsem změnil způsob detekce potenciálně nebezpečných formulářů, aby nebrala v potaz neškodné vyhledávací formuláře a neobtěžovala tedy uživatele na bezpečných stránkách. Také jsem změnil mechaniku zamčení formuláře, aby byla automatizována u nebezpečných formulářů a vyžadovala tedy méně činnosti od uživatele.

Průběh důkladného manuálního testování byl popsán v kapitole 7 spolu s popisem vlastní testovací stránky, kterou jsem napsal pro kontrolu obrany proti útokům, které jsem nenalezl implementované na webu.

Při možném pokračování tohoto projektu by bylo vhodné zaměřit se na nepřímé úniky dat z formulářů, což by však nezahrnovalo pouze úpravu kódu z Formlocku, ale zároveň i jiných wrapperů v rámci JavaScript Restrictoru. Kód pro detekci formulářů by poté mohl být rozšiřován o detekci nových hrozeb a mohl by komunikovat i s jinými skripty než původními z Formlocku. Dalším vylepšením při pokračování by bylo převést kód na verzi kompatibilní s Chrome Manifest V3, což momentálně není v rámci této práce možné, jelikož zbytek JavaScript Restrictoru je v současné době napsaný pro Manifest V2, takže by bylo nutné přepsat i všechny ostatní wrappery a komponenty rozšíření.

Literatura

[1] Acar, G. *No boundaries for user identities: Web trackers exploit browser login managers* [[online]]. Prosinec 2017. Revidováno 27-12-17 [cit. 2021-4-22]. Dostupné z: https://freedom-to-tinker.com/2017/12/27/no-boundaries-for-user-identities-web-trackers-exploit-browser-login-managers.

[2] Barth, A. *The Web Origin Concept* [Internet Requests for Comments]. RFC 6454. IETF, prosinec 2011. Dostupné z: https://tools.ietf.org/html/rfc6454.

[3] Bednář, M. *Automatické testování projektu JavaScript Restrictor*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=217224.

[4] Chrome Developers. *Extensions platform vision* [[online]]. Listopad 2020 [cit. 2021-4-23]. Dostupné z: https://developer.chrome.com/docs/extensions/mv3/intro/platform-vision/.

[5] Chrome Developers. *Migrating to Manifest V3* [[online]]. Listopad 2020. Revidováno 9-11-20 [cit. 2021-4-26]. Dostupné z: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-migration/.

[6] Chrome Developers. *Overview of Manifest V3* [[online]]. Listopad 2020 [cit. 2021-4-23]. Dostupné z: https://developer.chrome.com/docs/extensions/mv3/intro/mv3-overview/.

[7] Horňák, P. *Přenos bezpečnostních opatření z Chrome Zero do Javascript Restrictor*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=215288.

[8] Krishnamurthy, B. a Wills, C. On the Leakage of Personally Identifiable Information Via Online Social Networks. *Computer Communication Review*. Leden 2010, sv. 40, s. 112–117.

[9] MDN contributors. *Porting a Google Chrome extension* [[online]]. Prosinec 2018. Revidováno 19-8-19 [cit. 2021-4-12]. Dostupné z: https://extensionworkshop.com/documentation/develop/porting-a-google-chrome-extension/.

[10] MDN contributors. *Building a cross-browser extension* [[online]]. 2019. Revidováno 19-2-21 [cit. 2021-4-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Build_a_cross_browser_extension.

[11] MDN CONTRIBUTORS. *Chrome incompatibilities* [[online]]. Září 2020. Revidováno 24-3-21 [cit. 2021-4-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Chrome_incompatibilities.

[12] MDN CONTRIBUTORS. *Differences between API implementations* [[online]]. Září 2020. Revidováno 15-9-20 [cit. 2021-4-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Differences_between_API_implementations.

[13] MDN CONTRIBUTORS. *Extension pages* [[online]]. Září 2020. Revidováno 19-2-21 [cit. 2021-4-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/user_interface/Extension_pages.

[14] MDN CONTRIBUTORS. *IndexedDB API* [[online]]. Září 2020. Revidováno 8-4-21 [cit. 2021-4-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.

[15] MDN CONTRIBUTORS. *Referer - HTTP* [[online]]. Prosinec 2020. Revidováno 2020-12-15 [cit. 2020-12-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer.

[16] MDN CONTRIBUTORS. *Service Worker API* [[online]]. Září 2020. Revidováno 8-2-21 [cit. 2021-4-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.

[17] MDN CONTRIBUTORS. *Using Promises* [[online]]. Září 2020. Revidováno 29-3-21 [cit. 2021-4-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises.

[18] MDN CONTRIBUTORS. *Web Storage API* [[online]]. Září 2020. Revidováno 2-2-21 [cit. 2021-4-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API.

[19] MDN CONTRIBUTORS. *WebRequest* [[online]]. Září 2020. Revidováno 1-2-21 [cit. 2021-4-13]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest.

[20] MEHTA, P. *Creating Google Chrome Extensions*. 1. vyd. New Delhi, India: Apress, červen 2016. ISBN 1484217748.

[21] NIKIFORAKIS, N., INVERNIZZI, L., KAPRAVELOS, A., ACKER, S. V., JOOSEN, W. et al. You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions. In: *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. New York, Washington, USA: [b.n.], 2012. Dostupné z: https://www.kapravelos.com/publications/jsinclusions-CCS12.pdf.

[22] POHNER, P. *Detekce podezřelých síťových požadavků webových stránek*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/129272.

[23] Q-SUCCESS. *Usage statistics of JavaScript as client-side programming language on websites* [[online]]. Duben 2021 [cit. 2021-4-26]. Dostupné z: https://w3techs.com/technologies/details/cp-javascript.

[24] Schwarz, M., Lipp, M. a Gruss, D. JavaScript Zero: Real JavaScript and Zero Side-Channel Attacks. In: *Network and Distributed Systems Security (NDSS) Symposium 2018*. San Diego, CA, USA: Sciendo, únor 2018, s. 3 – 4. DOI: 10.14722/ndss.2018.23094. ISBN 1-1891562-49-5. Dostupné z: https://doi.org/10.14722/ndss.2018.23094.

[25] Starov, O., Gill, P. a Nikiforakis, N. Are You Sure You Want to Contact Us? Quantifying the Leakage of PII via Website Contact Forms. In: *Proceedings on Privacy Enhancing Technologies*. Darmstadt, Germany: Sciendo, Září 2016. DOI: 10.1515/popets-2015-0028. Dostupné z: https://doi.org/10.1515/popets-2015-0028.

[26] Timko, M. *Vylepšení rozšíření pro omezení volání JavaScriptu*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=197781.

[27] Červinka, Z. *Rozšíření pro webový prohlížeč zaměřené na ochranu soukromí*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=181826.

Příloha A

Seznam testovaných stránek

České stránky

- https://www.databazeknih.cz/napiste-nam
- https://horoskopy.najdise.cz/
- https://slovnik-cizich-slov.abz.cz/web.php/kontakt
- https://uloz.to/kontaktni-formular
- https://www.femina.cz/
- https://www.seznamka.cz/
- https://www.living.cz/
- https://www.pokerstars.cz/help/articles/la-chat-instructions/171138/
- https://www.yelp.cz/contact
- https://www.pijumate.cz/kontakty/

Zahraniční stránky

- https://www.bookdepository.com/contactus#
- http://anonymous-demo.github.io/
- https://help.imgur.com/hc/en-us/requests/new
- https://join1440.com/
- https://www.thegistsports.com/subscribe/
- https://marvelapp.com/contact-us
- https://bizzarodevs.com/
- https://www.ben-evans.com/newsletter
- https://www.belowthefold.news/

- https://www.brainpickings.org/newsletter/
- https://www.hackernewsletter.com/
- https://www.newyorker.com/newsletter/the-climate-crisis