VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PŘENOS BEZPEČNOSTNÍCH OPATŘENÍ Z CHROME ZERO DO JAVASCRIPT RESTRICTOR

PORTING OF CHROME ZERO FUNCTIONALITY TO JAVASCRIPT RESTRICTOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETER HORŇÁK**
AUTHOR

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

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Horňák Peter**
Program: Informační technologie
Název: **Přenos bezpečnostních opatření z Chrome Zero do JavaScript Restrictor**
Porting of Chrome Zero Functionality to JavaScript Restrictor
Kategorie: Bezpečnost

Zadání:

1. Seznamte se s projektem JavaScript Restrictor.
2. Analyzujte bezpečnostní opatření realizované nástrojem Chrome Zero z hlediska jejich aktuálnosti.
3. Navrhněte postup přenosu opatření z Chrome Zero do nástroje JavaScript Restrictor.
4. Přeneste bezpečnostní opatření, pište dostatečně kvalitní a komentovaný kód, aby byl vedoucím práce akceptován do nástroje JavaScript Restrictor.
5. Implementaci otestujte.
6. Práci vyhodnoťte a navrhněte možná zlepšení.

Literatura:

- Schwarzm Michael aj. *JavaScript Zero: Real JavaScript and Zero Side-Channel Attacks.* Network and Distributed Systems Security Symposium 2018. San Diego, CA, USA.
- Timko Martin. *Vylepšení rozšíření pro omezení volání JavaScriptu*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. *JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits*. In Network and Distributed Systems Security (NDSS) Symposium.

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 2019
Datum odevzdání: 28. května 2020
Datum schválení: 21. října 2019

Abstrakt

Táto práca sa zaoberá zlepšovaním bezpečnosti a ochranou súkromia užívateľov webových prehliadačov rozšírením funkcionality doplnku webového prehliadača nazývaného JavaScript Restrictor, ktorého prototyp bol vytvorený za účelom ochrany užívateľov na webe. V rámci tejto práce sú analyzované bezpečnostné opatrenia realizované nástrojom Chrome Zero, ktorý implementuje opatrenia proti mikroarchitektúrnym útokom, útokom zneužívajúcich časovače s vysokým rozlíšením a zneužitiu samotného JavaScript enginu. Práca vyhodnocuje aktuálnosť opatrení a vybrané z nich integruje do JavaScript Restrictoru. Opatrenia sú otestované a vyhodnotené z pohľadu každodenného používania.

Abstract

This thesis deals with improvements of security and privacy protection of web browser users by enhancing functionality of web browser extension called JavaScript Restrictor, which prototype was created in order to protect users on web. Within this thesis are analyzed security measures realized by tool Chrome Zero, which implements measures against microarchitectural attacks, attacks abusing high resolution timers and abuse of JavaScript engine. Thesis evaulates topicality of measures and integrates selected into JavaScript Restrictor. Measures are tested and evaluated from every day use point of view.

Kľúčové slová

Bezpečnosť, JavaScript Restrictor, Chrome Zero, JavaScript Zero, rozšírenie webového prehliadača, microarchitektúrne útoky, časovače s vysokým rozlíšením, web

Keywords

Security, JavaScript Restrictor, Chrome Zero, JavaScript Zero, web browser extension, microarchitectural attacks, high resolution timers, web

Citácia

HORŇÁK, Peter. *Přenos bezpečnostních opatření z Chrome Zero do JavaScript Restrictor*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Přenos bezpečnostních opatření z Chrome Zero do JavaScript Restrictor

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Libora Polčáka Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Peter Horňák
27. mája 2020

Poďakovanie

Rád by som poďakoval Ing. Liborovi Polčákovi PhD. za jeho čas, ktorý venoval pri vedení tejto bakalárskej práce a za poskytnutú odbornú pomoc. Tak isto by som rád poďakoval mojim spolužiakom a kamarátom, ktorí ma motivovali pri písaní tejto práce. Nakoniec by som chcel poďakovať mojim najbližším, ktorí ma vždy podporili, keď som to potreboval.

Obsah

Kapitola 1

Úvod

V poslednej dobe je možné pozorovať stále rastúcu popularitu dynamických webových stránok, ktorých neoddeliteľnou súčasťou sú skripty vytvorené v programovacom jazyku JavaScript bežiace v prehliadači návštevníka stránky. Až na 95% webových stránok je možné nájsť práve JavaScript [32], ktorý umožňuje ich prevádzkovateľom spúšťať rôzne výpočty na strane klienta, bez toho aby si to bežný užívateľ čo i len všimol.

Týmto spôsobom je možné určitú záťaž preniesť z webových serverov na zariadenia klientov, avšak v mnohých prípadoch tieto programy monitorujú a získavajú dáta o užívateľoch, pričom nie je úplne známe akým spôsobom sa následne s týmito dátami pracuje. V roku 2017 až 77% načítaní stránok obsahovalo aspoň jeden tracker (program na zbieranie dát) a približne 10% stránok obsahovalo 10 a viac trackerov [23], ktoré tieto dáta môžu posielať tretím stranám.

Ďalšou možnou hrozbou pre užívateľov sú mikroarchitektúrne útoky, ako napríklad útoky na vyrovnávaciu pamäť môžu byť vykonané cez web stránky pomocou JavaScriptu. Tieto časované útoky umožňujú útočníkom sledovať napríklad stlačenia klávesnice, históriu navštívených webov [37] a ďalšie mnohé informácie, ktoré narúšajú súkromie návštevníkov stránok.

Aj keď poskytovatelia prehliadačov reagujú na tieto zraniteľnosti, stále existuje množstvo spôsobov ako zneužiť natívne správanie JavaScriptu, avšak zablokovanie týchto vlastností môže úplne znemožniť kompatibilitu s niektorými webovými stránkami. Následkom toho je možné vytvárať unikátne identity užívateľov (anglicky fingerprint), ktoré vznikajú na základe získaných informáciach o hardvéri ako napríklad rozlíšenie obrazovky, počet jadier procesora alebo dáta o momentálnej výpočtovej sile, ale aj dát o operačnom systéme, nainštalovaných rozšíreniach a používanom prehliadači. Na základe fingerprintov je možné sledovať aktivitu užívateľa naprieč viacerých stránok [35].

Hoci väčšina prehliadačov dodržuje štandardy ako sa majú stránky renderovať alebo správať, čo definuje World Wide Web Consortium (W3C), prehliadače sa v istých častiach rozlišujú vo svojej implementácií, umožňujúc developerom webových prehliadačov ako napríklad Tor, obmedziť niektoré spôsoby sledovania. Ďalším spôsobom pre ochranu môžu byť webové rozšírenia, ktoré sa snažia odstrániť rozdiely medzi jednotlivými implementáciami prehliadačov alebo pridávajú náhodnosť, poprípade zaokrúhľovanie hodnôt aby sa zamedzilo unikátnosti týchto hodnôt.

Cieľom tejto práce bolo naštudovať problematiku bezpečnostných problémov prehliadačov a webu ako takého a zároveň vylepšiť funkcionalitu webového rozšírenia JavaScript Restrictor [38, 39] na základe rozšírenia pre Google Chrome nazývaného Chrome Zero, ktorý implementuje bezpečnostné opatrenia na základe štúdie JavaScript Zero [36].

JavaScript restrictor je webové rozšírenie, ktoré bolo vytvorené za účelom obmedzovať získavanie dát o osobách [31] a to pomocou techník, ktoré nezasahujú do priamej implementácie prehliadača. Rozšírenie je schopné zablokovať prístup k JavaScriptovým objektom, metódam a atribútom alebo oklamať stránky na základe menej presnej implementácie niektorých funkcionalít. Hlavným cieľom je popri chránení užívateľa tak isto nezničiť kompatibilitu s navštívenými stránkami. Podobný prístup pre ochranu užívateľov využíva aj Chrome Zero, ktorý implementuje techniku zapuzdrenia pôvodnej implementácie funkcie. Funkcia, ktorá je obalená je tak odstránená z globálneho priestoru a dá sa k nej a jej atribútom pristúpiť len z obaľovacej funkcie. Túto techniku využíva aj JavaScript restrictor čím webu poskytuje skreslené informácie o užívateľovi, vďaka čomu poskytuje väčšiu anonymitu pri prehliadaní a zároveň aj znemožňuje zneužitie určitých zraniteľností.

Práca je rozdelená do niekoľkých kapitol. Na začiatku v kapitole 2 je predstavený projekt JavaScript Restrictor [38, 39], jeho analýza funkcionalita momentálnych bezpečnostných riešení. V kapitole 3 je popísaná analýza JavaScript Zero [36] a rozšírenia Chrome Zero, jeho funkcionalita, aktuálnosť riešenia a bezpečnostné opatrenia vhodné na prenos do JavaScript Restrictora. Následne v kapitole 4 je predstavený návrh prenosu bezpečnostných opatrení a v kapitole 5 sú popísane detaily implementácie rozšírenia. Nakoniec v kapitole 6 je predstavený priebeh testovania novej funkcionality a dopad na bežné prehliadanie stránok.

Kapitola 2

Sledovanie užívateľov a JavaScript Restrictor

Táto kapitola sa zaoberá projektom JavaScript Restrictor, ktorého prototyp vytvoril Ing. Zbyněk Červinka ako súčasť jeho diplomovej práce. Následne jeho funkcionalitu rozšíril a vylepšil Ing. Martin Timko, tak isto vo svojej diplomovej práci a zverejnil ho pod jeho súčasným názvom, čoho súčasťou bolo aj vytvorenie kompatibility pre prehliadače Google Chrome a Opera.

Prvotný nápad vytvoriť webové rozšírenie, ktoré funguje ako firewall pre JavaScriptové APIs (rozhrania pre programovanie aplikácií) vymyslel Ing. Libor Polčák Ph.D., ktorý je zároveň aj momentálny udržiavateľ projektu a tak isto viedol spomenuté diplomové práce.

V sekcií 2.1 je popísaný účel a motivácia prečo toto rozšírenie bolo vytvorené. Následne popisujem súčasný návrh a implementáciu v sekcií 2.2. V sekcií 2.3 je predstavené užívateľské rozhranie a v poslednej sekcií (sekcia 2.4) sú zhrnuté výsledky, ktoré rozšírenie poskytuje a momentálne nedostatky.

2.1 Techniky využívané na sledovanie užívateľov

Snažíme sa riešiť problém, ktorý sa stáva každoročne väčším, keďže digitálna populácia každým rokom rastie. Odhad je, že v roku 2019 používa internet 4,1 miliardy [11] a počet stále narastá. Sledovanie aktivity užívateľov nemusí automaticky znamenať, že sa jedná o zlé úmysly. Z dôvodu obrovského dostupného obsahu na internete v rozličných formátoch a v rôznych kategóriách, sa sleduje aktivita jednotlivých užívateľov a na základe toho sa prispôsobuje obsah [33], ktorý sa im zobrazuje. Ako všetko, aj táto problematika má svoje pozitíva a negatíva. Navštevovatelia webov nájdu obsah ktorý hľadajú rýchlejšie na základe jemu vytvoreného profilu a poskytovatelia produktov majú väčšiu šancu, že produkty, ktoré ponúkajú sa dostanú k ich cieľovej skupine. Tieto profily vznikajú napríklad na základe analýzy navštívených URL, ako dlho sú jednotlivé stránky prehliadané alebo o aké témy sa daný profil zaujíma[20]. Avšak, často nie je známe, aké konkrétne informácie sú o používateľoch zbierané, ako sú presne používane a veľa krát sa toto deje bez vedomia jedinca o ktorom sa tieto informácie zisťujú. To môže viesť až k situáciám, kedy sa získané dáta používajú k identifikácií osôb namiesto vytvárania anonymnej identity ku ktorej je priradený vhodný obsah. Identitu osoby je možné zistiť napríklad na základe e-mailovej adresy, adresy bydliska, telefónu alebo mena. Ďalej tak isto je potrebné ochraňovať užívateľov aby sa na verejnosť nedostali citlivé dáta ako informácie o jeho momentálnej polohe, čísla

kreditnej karty alebo história navštívených stránok [39]. Všetky tieto informácie môžu byť zneužité na rôzne škodlivé účely, s rôznymi dôsledkami.

Techniky získavania dát o užívateľoch

Všetky tieto dáta je možné získať viacerými spôsobmi, ktoré nie je technicky zložité zakomponovať do webovej stránky, čo naznačuje aj fakt, že štúdia z roku 2018 [25] zistila, že z 500 najnavštevovanejších webových stránok v Spojených Štátoch Amerických až 90% načítaní stránok obsahovalo aspoň jeden tracker.

Medzi najčastejšie spôsoby pre získavanie informácií patria Cookies [3], Session replay scripty, [39], Web Storage[2] a IndexedDB API [9].

Cookies

Cookies sú trojice doména, meno a hodnota, ktoré sú uložené lokálne u užívateľa v jeho prehliadači a manipuluje sa s nimi pomocou jazyka JavaScript alebo HTTP hlavičiek [34]. Sú navrhnuté aby boli schopné ukladať obmedzený objem dát špecifických pre stránku navštívenú užívateľom a môže ku ním pristúpiť buď daný web, alebo užívateľ. Toto správanie umožňuje serveru prispôsobiť web ku danému užívateľovi alebo stránka môže obsahovať skript, ktorý je schopný pracovať s týmito dátami.

Hoci pomocou cookies môžu byť uložené iba dáta, ktoré užívateľ dobrovolné zveril stránke tým, že stránku prehliada alebo, ktoré už stránka vlastní, môžu byť cookies použité k monitorovaniu jednotlivcov cez väčší počet webov a to tým, že môžu byť tieto dáta zdielané ďalším tretím stranám [34]. Napríklad ak užívateľ navštívi stránku, ktorá načítava reklamu z tretej strany, tak užívateľovi táto strana môže poslať HTTP `Set-Cookie` hlavičku, ktorá nastaví cookie do zariadenia návštevníka. Ak potom užívateľ navštívi ďalší web, ktorý využíva tú istú tretiu stranu pre poskytovanie reklamy, tak tejto strane je automaticky v požiadavku poslaná aj cookie, ktorú pred tým nastavila, vďaka čomu je schopná rozpoznať užívateľa naprieč viacerými webmi.

V súčastnosti však existuje veľa známych webových rozšírení, ktoré sú schopné rozpoznať a blokovať takéto sledovanie od tretích strán. Navyše najnovšie verzie prehliadačov majú natívnu podporu zakázať všetky cookies tretích strán [27] a dokonca v niektorých prípadoch majú túto možnosť štandardne zapnutú.

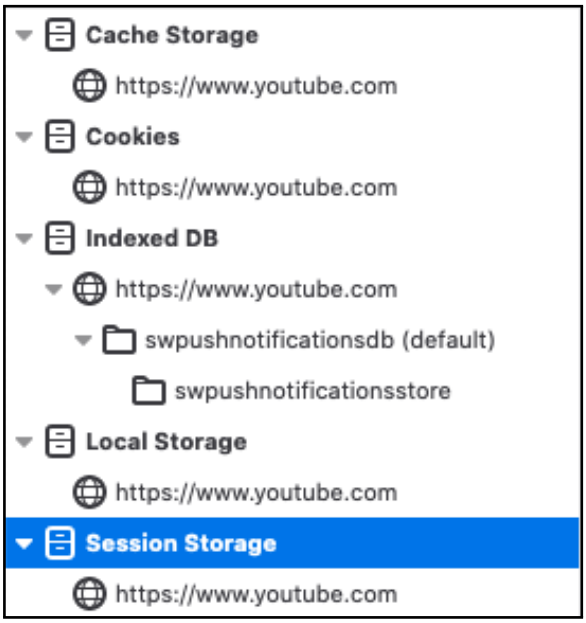
Tento dôvod motivoval poskytovateľov reklám a tvorcov trackerov aby našli lepšie spôsoby ako identifikovať návštevníkov ich webov. Napríklad cookies používané systémom Google Analytics, ktoré je možné nájsť na mnohých weboch, využívajú takzvané neviditeľné pixle [16] pre nastavovanie týchto cookies.

Web Storage

Web Storage je spôsob, pomocou ktorého webové aplikácie môžu ukladať perzistentné dáta v prehliadači. S týmto obsahom sa môže pracovať po koniec momentálnej relácie (t.j. sessionStorage) alebo pomedzi viaceré relácie (t.j. localStorage) [10]. Táto technológia je založená na metóde client-side scripting (vykonávanie programu na strane klienta). Vo Web Storage sa využíva na sledovanie užívateľa rovnakým spôsobom ako HTTP Cookies a navyše umožňuje ukladať väčšie množstvo dát. Tak isto je treba podotknúť, že táto technológia je oveľa menej známa a preto je väčšia pravdepodobnosť, že užívatelia nebudú priebežne mazať uložené dáta ako to môže byť v prípade Cookies.

Indexed Database API

Definuje rozhranie založené na jazyku JavaScript, pre vstavaný transakčný databázový systém. Podobne ako Web Storage, umožňuje ukladanie štrukturovaných dát v prehliadači. Poskytnuté API je jediný možný spôsob, ako webová aplikácia môže pristupovať a upravovať tieto dáta. Hlavnou výhodou je spôsob akým sú dáta uložené, to umožňuje lepšie štruktúrovanie dát a zároveň aj väčší objem. Pomocou IndexedDB je možné vytvoriť niekoľko objektovo orientovaných databáz, priradených k jednému zdroju, každá z týchto databáz umožňuje ukladať objekty, ktorých obsah sa dá zoraďovať a triediť na základe viacerých kľúčov [10]. Rozhranie využíva kľúče a indexovanie, namiesto SQL dotazov. Opäť tu platí rovnaký princíp ako u vyššie spomenutých technológií, čiže web môže pristúpiť k dátam iba ak platí, že pôvod dát sa rovná pôvodu požiadavku.



Obr. 2.1: Reprezentácia údajov uložených na strane klienta, ktoré ukladá prehliadač Firefox/71.0. pre webovú stránku www.youtube.com

Session replay skripty

Pre monitorovanie návštevníkov v reálnom čase sa využívajú takzvané session replay skripty. Tieto nástroje sú typicky napísané v jazyku JavaScript a sú schopné zaznamenávať akcie, ktoré užívateľ v danom momente vykonáva, typicky sú to stlačenia kláves, scrollovanie (listovanie stránky vertikálnym smerom) alebo užívateľské vstupy v reálnom čase [15]. Tento spôsob umožňuje poskytovateľom webových aplikácií zbierať komplexné informácie o správaní užívateľov a na základe toho prispôsobovať obsah, ktorý im poskytujú. Nazbierané dáta sú následne ukladané a spracovávané pre potreby poskytovateľa. Avšak toto sa deje bez znalosti užívateľa a jeho súhlasu. Navyše neexistuje žiadny jednoduchý spôsob pre bežných návštevníkov ako sa tomuto spôsobu sledovania vyhnúť[39]. Príkladom služby pre analýzu webových aplikácií, pomocou monitorovania správania užívateľa je Hotjar[1].

[1] https://www.hotjar.com/

2.2 Aktuálna funkcionalita a implementácia rozšírenia

Fungovanie doplnku je postavené na princípe zapuzdrenia objektov, funkcií a atribútov takým spôsobom, aby externé skripty mohli k ním pristupovať iba pomocou vytvorenej obálky. Tento princíp obmedzuje prístup k zapuzdrenému kódu, vďaka tomu je možné ho upravovať podla našich pravidiel a vracať ho v upravenej podobe. K zapuzdreniu pôvodného kódu prichádza ešte v dobe pred spracovaním kódu načítavanej webovej stránky, čo zaručuje, že ku týmto konštrukciám nie je možné pristúpiť pred ich obalením.

Pomocou užívateľského rozhrania je možné si zvoliť medzi piatimi úrovňami funkcionality rozšírenia. Úroveň 0 predstavuje vypnutie celkovej ochrany, následne je možné si vybrať úrovne 1 až 3, ktoré majú každá vlastnú preddefinovanú funkcionalitu, kde 1 znamená minimálnu ochranu a úroveň 3 maximálnu ochranu. Na koniec existuje možnosť vytvorenia vlastných nastavení, kde je možné povoliť alebo zakázať jednotlivé prvky ochrany [38].

V oficiálne dostupnej verzii rozšírenia sú zapuzdrené 3 konštrukcie, ktoré boli implementované v diplomovej práci Zbyňka Červinku [39]. Obalenie ostatných konštrukcií implementoval Martin Timko vo svojej diplomovej práci [38].

Súčasťou tejto práce je implementácia, ktorá však bude pridaná do pripravovanej verzie 0.3, kde niektoré z aktuálne obalených konštrukcií budú odstránené. Z toho dôvodu, táto sekcia popisuje iba konštrukcie, ktoré budú zachované v novej verzií rozšírenia. Novú verziu pripravuje Ing. Libor Polčák PhD., ktorý je zároveň aj vedúci tejto práce.

- **metóda `window.performance.now()`:** Táto metodá objektu window.performance vracia čas, ktorý ubehol od vytvorenia načítaného dokumentu. Toto časové rozmedzie môže byť vyjadrené až v mikrosekundách, avšak každý prehliadač môže túto hodnotu zaokrúhľovať svojim spôsobom [4]. V prípade JavaScript restrictoru je štandardne nastavené zaokrúhľovanie na 100 milisekúnd, avšak s možnosťou nastavenia na desiatky alebo tisícky milisekúnd. Zaokrúhľovanie umožňuje sa vyhnúť útokom spojených s presným načasovaním [37].

- **metóda `window.performance.getEntries()`:** Táto metóda vracia list objektov typu `PerformanceEntry` definované pre danú stránku. Tieto objekty môžu byť zneužité a preto návratová hodnota je nahradená upraveným listom, v ktorom sú tieto hodnoty zaokrúhlené. Ten istý prístup je aplikovaný pre metódy `getEntriesByName()` a `getEntriesByType()`, ktoré sa rozlišujú iba v spôsobe selektovania vracaných objektov.

- **objekt `window.Date`:** Objekt obsahuje časové údaje, konkrétne dátum, čas s presnosťou na milisekundy a časovú zónu. Presnosť zaokrúhľovania času je možné upraviť zo stoviek milisekúnd až na celé sekundy. Predvolené zaokrúhľovanie je nastavené na stovky milisekúnd.

- **objekt `window.XMLHttpRequest`:** Vďaka zapuzdreniu tohto objektu je možné žiadosti typu `XMLHttpRequest` odchytávať a následne úplne zablokovať alebo sa pre každú žiadosť spýtať užívateľa, či chce danú požiadavku povoliť alebo zablokovať. Tak isto si vie užívateľ jednotlivé požiadavky zobrazovať. Keďže môžu obsahovať osobné dáta, ktoré stránka o užívateľovi nazbierala ale tak isto môžu byť potrebné pre funkčnosť webu, predvoleným nastavením je pýtať sa užívateľa na každú požiadavku.

- **vlastnosť `window.navigator.deviceMemory`:** Pomocou tejto vlastnosti je možné zistiť veľkosť pamäte RAM v gigabajtoch, na základe tejto hodnoty je potom možné deliť užívateľov do kategórií. Táto hodnota je nastavená na hodnotu 4, čo bolo v čase implementácie najčastejšia hodnota veľkosti RAM medzi užívateľmi, čo sa však zmenilo v Novembri 2019, kedy veľkosť 8 GB začala byť najčastejšie používaná. [26]

- **vlastnosť `window.navigator.hardwareConcurency`:** Hodnota určujúca počet logických jadier procesora na zariadení, dostupných pre využitie. Hodnota je nastavená na 2, čo je najčastejšia hodnota užívateľov navštevujúcich web [26] .

- **metóda `window.HTMLCanvasElement.prototype.toDataURL()`:** Získavanie informácií o HTML elementoch typu Canvas môže slúžiť na vytváranie fingerprintov. Keďže jednotlivé zariadenia sa líšia v spôsobe akým tieto elementy vykresľujú, je možné ich rozlišovať na základe hashu týchto elementov. Stránka, ktorá zavolá túto metódu, dostane upravený obrázok tak, že všetky pixly budú mať hodnotu `RGB(255,255,255)`, čiže budú biele. Na základe toho bude výsledný canvas vyzerať rovnako na všetkých zariadeniach.

- **metóda `HTMLCanvasElement.prototype.toBlob()`:** Metóda vytvára falošný Canvas, ktorý nahrádza ten originálny.

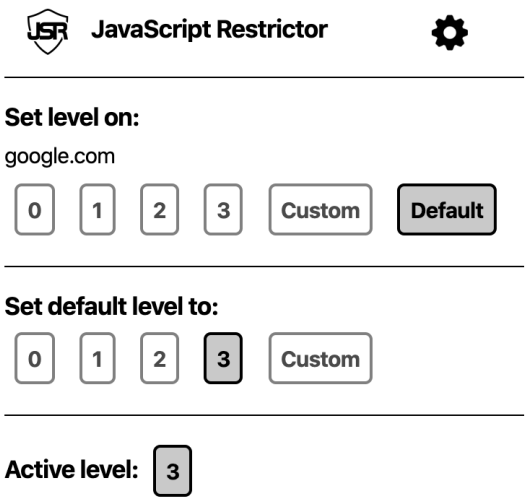
- **metóda `CanvasRenderingContext2D.prototype.getImageData()`:** Namiesto pôvodných dát sa vracia falošný objekt typu `ImageData`.

2.3 Užívateľské rozhranie

Užívateľské rozhranie pozostáva z dvoch častí, vyskakovacie okno 2.2, ktoré sa zobrazí po kliknutí na ikonu a stránku s osobným nastavením. Všetky prvky sú v anglickom jazyku, vďaka čomu je možné zaujať užívateľov z krajín aj mimo Česka a Slovenska. Dizajn je minimalistický, vďaka čomu je veľmi jednoduché sa vo vyskakovacom okne orientovať a hneď pri prvom pohľade je jasné, čo jednotlivé prvky rozhrania robia.

Užívateľ je schopný si zvoliť level ochrany pre momentálne navštívenú doménu, tak isto je možné nastaviť predvolený stupeň ochrany pre stránky u ktorých nie je špecifikované inak. Vyskakovacie okno užívateľovi ukazuje akú momentálnu úroveň ochrany mu rozšírenie poskytuje na otvorenom dokumente. Následne v okne je ikona nastavenia pomocou ktorej sa užívateľ dostane na stránku s nastaveniami.

V nastaveniach je možné upraviť predvolenú hodnotu ochrany, tak isto sa tam nachádzajú užitočné odkazy na stránku projektu, testovaciu stránku a popis jednotlivých úrovní. Ďalšou funkciou je nastavenie protekcie pre konkrétnu doménu, vďaka čomu je možné upraviť úroveň ešte pred tým ako užívateľ túto doménu navštívi. Nakoniec si užívateľ môže nastaviť svoj vlastný level ochrany a vybrať si, ktoré konštrukcie bude rozšírenie zaobaľovať.



Obr. 2.2: Vyskakovacie okno rozšírenia JavaScript Restrictor

2.4 Zhodnotenie rozšírenia

Na základe testovania [38] je možné vidieť, že obaľovanie jednotlivých API skutočne poskytuje väčšiu anonymitu na internete. Nová verzia niektoré objekty prestáva obaľovať, z dôvodu problémov s kompatibilitou stránok, ktoré sa kvôli ním vyskytovali v prechádzajúcich verziách.

Rozšírenie zabezpečuje ochranu pred zneužitím časovačov pomocou zaokrúhľovania, upravuje prístup k elementu typu `Canvas`, čo sťažuje identifikovanie užívateľa a dokáže blokovať `XMLHttpRequest`, vďaka čomu užívateľ dokáže kontrolovať aké požiadavky kód na stránke posiela.

Stretol som sa však s istými problémami. Na prehliadači Google Chrome rozšírenie nefungovalo. Tak isto som narazil na problém pri prepínaní úrovní pomocou vyskakovacieho okna, kde prepnutie na inú úroveň ochrany nemalo žiadny efekt. Posledný problém, nastával pri nastavovaní úrovne ochrany na špecifickú doménu. Ak bola úroveň nastavená na stránku s URL obsahujúcov subdoménu, tak sa používala prednastavená úroveň.

Kapitola 3

Chrome Zero

Táto kapitola sa venuje rozšíreniu pre prehliadač Google Chrome nazývané *Chrome Zero*, ktoré implementuje opatrenia pre ochranu užívateľa predstavené v práci *JavaScript Zero: Real JavaScript and Zero Side-Channel Attacks*. Rozšírenie bolo použité ako dôkaz koncepcie práce [36].

JavaScript Zero predstavuje skupinu opatrení, ktoré redukujú možnosti útokov v moderných webových prehliadačoch. Využíva pri tom pokročilé vlastnosti jazyka JavaScript pre dynamické blokovanie potencionálne zneužiteľných vlastností JavaScriptu.

V sekcií 3.1 sú popísané nebezpečenstvá pre návštevníka webu, ktorým sa táto práca venuje. Autori sa zamerali na dvojicu útokov a to konkrétne na mikroarchitektúrne útoky a útoky postrannými kanálmi. Následne v sekcií 3.2 je rozobratý návrh rozšírenia JavaScript Zero a jeho štruktúra. Potom sa pozrieme na implementáciu samotného rozšírenia 3.3 a ktoré protiopatrenia boli v rozšírení zavedené a to v sekcii 3.4. Na konci kapitoly v sekcii 3.5 sú zanalyzované výsledky tejto práce a rozšírenie Chrome Zero.

3.1 Možnosti zneužitia jazyka JavaScript

Iba v nedávnej dobe bolo odhalené, že je možné vykonávať mikroarchitektúrne útoky pomocou JavaScriptu [37]. Keďže kód v JavaScripte je samotne oddelený vo vlastnom prostredí a je vo svojej podstate beží iba v jednom vlákne, útočníci sa stretávajú s istými problémami v porovnaní so strojovým kódom [36].

V tejto sekcií sú predstavené jednotlivé koncepty, ktoré sa využívajú a sú popísané spôsoby ako je možné ich zneužiť.

Adresa pamäte

V JavaScripte neexistuje spôsob akým by programátor mohol priamo pristupiť do pamäte. Hoci ukazovatele sa interne používajú, nie sú sprístupnené, čo znemožňuje útočníkovi zistiť informácie o virtuálnej adrese programu. Avšak `ArrayBuffer` je objekt, ktorý reprezentuje pole o fixnej veľkosti, obsahujúce binárne dáta. S týmito dátami však nie je možné priamo manipulovať [7], namiesto toho je potrebné vytvoriť reprezentáciu binárnych dát v určitom formáte, ktorá umožňuje do pamäte zapisovať, alebo znej čítať. To znamená, že ak 1 virtuálna adresa v `ArrayBufferi` je identifikovaná, zbytok adries je tak isto známy, keďže adresy v pamäti a indexy pola sú lineárne závislé.

Mikroarchitektúrne útoky ako napríklad útoky na cache [29] alebo útoky založené na DRAM [30] sa však nespoliehajú na virtuálne adresy ale na fyzické adresy. Avšak pre mo-

derné operačné systémy platí, že neprivilegovaný užívateľ nemá možnosť pristúpiť k mapovaniu virtuálnych adries na fyzické. Preto sa útočníci musia uchýliť k tomu, získať tieto informácie z bočných kanálov. `ArrayBuffer` je prehliadačom vždy alokovaný zarovno stránky, čo znamená, že prvý byte je alokovaný na začiatku novej fyzickej stránky a jeho 12 najmenej významových bitov je nastavených na hodnotu 0 [36].

Druhú vlastnosť, ktorú je možné zneužiť, je fakt, že alokácia pamäte na halde prebieha na základe požiadavku (anglicky on demand), to znamená, že prehliadač požiada operačný systém o pridelene ďalšej pamäte na halde až vtedy, keď je to potrebné. Pre alokovanie väčšej časti pamäte, prehliadač zvykne využívať `mmap`[36], ktorý je optimalizovaný pre alokáciu stránok o veľkosti 2 MB. Tieto stránky sa nazývajú *transparent huge pages* (THP) a majú väčšiu veľkosť oproti typickejším stránkam o veľkosti 4 KB [37]. Pre získanie indexu, kde nová stránka začína, útočník iteruje cez pole a meria čas ako dlho trvá pristúpiť k danému prvku. Keďže fyzické stránky sú alokované dynamicky na požiadavku, čo znamená až pri prvom prístupe do pamäte, tak čas pre pristúpenie do prvku na danom indexe je významne väčší. Na základe tohto času útočník je schopný rozoznať, že sa dostal na novú stránku o veľkosti 2 MB. Na základe spôsobu akým THP fungujú vie, že táto stránka má 21 najmenej významných bitov fyzickej a virtuálnej stránky nastavených na hodnotu 0.

Viacvláknovosť

S príchodom HTML5 prišiel tento koncept do JavaScriptu, vo forme tzv. `WebWorkers`. Umožňujú jednoduchým spôsobom spustiť skript na pozadí vo vlastnom vlákne, bez toho aby narušili užívateľské rozhranie. Každé vlákno má svoju synchrónnu frontu udalostí, ktoré sa majú vykonať. Komunikácia s procesom, ktorý vytvoril vlákno prebieha na základe správ, čo sú opäť udalosti.

Web workers tak isto predstavujú isté nebezpečenstvo, keďže boli predstavené útoky [22], ktoré na základe paralelných procesov sú schopné získať informácie o užívateľoch. Na základe prerušení CPU dokáže takýto proces bežiaci na pozadí detegovať, ktoré tlačidlá klávesnice boli stlačené.

S týmto konceptom JavaScript uviedol tak isto `SharedArrayBuffer`. Je to objekt, ktorý je veľmi podobný `ArrayBuffer`u, s rozdielom, že k jeho prvkom môže pristupovať viac vlákien naraz. V práci [37] je ukázaný spôsob, ako môže byť `SharedArrayBuffer` zneužitý na vytvorenie časovaču s presnosťou na nanosekundy. Tento spôsob vyžaduje aby jedno vlákno zvyšovalo hodnotu v tomto poli a hlavné vlákno k nemu môže jednoducho pristúpiť a využiť ho napríklad ako časovú značku. Táto metóda umožňuje vytvárať časové značky s rozdielom iba 2 ns, čo je dostatočné pre vykonávanie časovaných útokov.

3.2 Návrh JavaScript Zero

Základným konceptom JavaScript Zero [36] je vytvoriť abstraktnú vrstvu medzi samotným enginom a rozhraním, ktoré je poskytnuté potencionálnemu útočníkovi. Základnou predstavou je ochrániť objekty, funkcie, atribúty a rozhrania pomocou tejto vrstvy. Týmto spôsobom nie je nutné upravovať zdrojový kód v prehliadači a je jednoduchšie ho v prípade potreby upraviť alebo rozšíriť. Medzivrstva môže zablokovať, upraviť alebo iba prepustiť dané volania na základe ochranných pravidiel, ktoré sú definované. Avšak samotné konštrukcie jazyka ako podmienky, cykly alebo dátové typy nie sú týmito pravidlami nijako ovplyvnené. Každý prípad s ktorým sa má zaobchádzať, má presne definovanú štruktúru ktorá popisuje čo sa má vykonať v prípade volania.

Pre rozšírenie flexibility existujú 4 možnosti, ktoré ovplyvňujú vykonanie funkcie:

- **Povoliť:** Funkcia je explicitne povolená a rozšírenie jej behu nebude brániť ani ho upravovať.
- **Upraviť:** Namiesto pôvodnej funkcie sa vykoná telo funkcie špecifikovanej v ochranných pravidlách.
- **Blokovať:** Invokácia funkcie je zablokovaná a je nahradená predom definovanou hodnotou.
- **Povolenie užívateľa:** Užívateľ musí dať povolenie pre vykonanie funkcie, ktoré sa mu zobrazí pomocou oznámenia, inak je volanie blokované.

```
1  {
2    "function": {
3      "window.performance.now": {
4        "action": "ask",
5        "return": "0"
6      }
7    },
8    "object": {
9      "window.*Array": {
10       "action": "modify",
11       "file": "array_aslr.js"
12     }
13   }
14 }
```

Výpis 3.1: Pravidlá Chrome Zero na úrovni 1, upravujúce funkciu `window.performance.now` a objekt `Array`

Výpis 3.1 popisuje príklad pravidiel, ktoré sú implementované v rozšírení Chrome Zero[1]. Pravidlá definujú nové správanie pre dve konštrukcie, funkcia `window.performance.now`, pri ktorej volaní užívateľ rozhodne či chce funkciu povoliť alebo zablokovať. Druhý modifikovaný sú objekty typu `TypedArray`, ktoré sú upravené na základe kódu v súbore `array_aslr.js`.

Cieľom tohto zápisu je, že ostatní užívatelia sú schopný napísať si vlastné bezpečnostné opatrenia a zdielať tieto opatrenia medzi ostatnými užívateľmi. V rozšírení sú implementované rôzne úrovne zabezpečenia a takýmto spôsobom je veľmi jednoduché vytvoriť ďalšiu úroveň na základe vlastných požiadaviek užívateľa, za predpokladu, že má isté skúsenosti s programovaním. Ďalšou výhodou je v prípade novo objavenej zraniteľnosti možné jednoducho implementovať nové opatrenie oddelené od zvyšku kódu a zároveň jednotlivé opatrenia prispôsobiť pre konkrétny hardware a software.

Autori zaviedli do rozšírenia koncept ochranných úrovní, pre zjednodušenie každodenného používania. Užívateľ si vyberie jeden z možných stupňov ochrany, ktoré sa líšia počtom a striktnosťou pravidiel. To je možné na základe jednoduchého užívateľského rozhrania, ktoré rozšírenie poskytuje. Toto však zamedzuje niektorým výhodám spomenutým v predchádzajúcom odseku.

[1] https://github.com/IAIK/ChromeZero

3.3 Implementácia rozšírenia Chrome Zero

Pri implementovaní rozšírenia boli určené 3 hlavné ciele, ktoré majú byť splnené:

- Obmedzenia nemôže byť možné obísť použitím samo modifikujúcim sa kódom (anglicky self-modifying code), ako napríklad funkcie vyššieho rádu (anglicky higher-order scripts)
- Zamedzenie zneužitiu potencionálne nebezpečných vlastností jazyka, musí byť nezvratné na každej stránke.
- Kompatibilita a užívateľské rozhranie nemôžu byť príznačne ovplyvnené.

```
1 var original_function = window.performance.now;
2 window.performance.now = function() {
3     return 0;
4 }
5 console.log(window.performance.now); // 0
6 console.log(original_function.call(window.performance)); // Spravna hodnota
```

Výpis 3.2: Príklad použitia techniky virtual machine layering aplikovanú na funckiu `window.performance.now`. Do konzoly sa vypíše hodnota novej funkcie, ktorá bude vždy 0 a tak isto aj pôvodná funckia[36].

Virtual machine layering

Zabezpečenie obchádzania prepísaných funkcií môže byť problém, pretože v JavaScripte je možné dynamicky vytvárať volanie funkcií a tým pádom v zdrojovom kóde sa toto volanie nemusí nachádzať. Je potrebné zabezpečiť volania všetkých funkcií, pretože vynechanie jednej by narušilo celý koncept. Preto sa pre zaistenie prístupu k upravenej funkcií a toho aby žiadna časť kódu nemohla pristúpiť k pôvodnej funkcii, je použitá technika nazývaná virtual machine layering. Pri tomto spôsobe sú referencie na pôvodné funkcie nahrádzané, novými funkciami viď. výpis 3.2.

Ukázaný spôsob je možné použiť, avšak je potrebné tento princíp rozšíriť na objekty a atribúty. Aby bolo možné aplikovať pravidlá na objekty, sa autori rozhodli zmeniť implementáciu pôvodného objektu pomocou *Proxy* objektu.

```
1  var handler = {
2      get: function(obj, prop) {
3          return prop in obj ?
4              obj[prop] :
5              37;
6      }
7  };
8  var p = new Proxy({}, handler);
9  p.a = 1;
10 p.b = undefined;
11 console.log(p.a, p.b); // 1, undefined
12 console.log('c' in p, p.c); // false, 37
```

Výpis 3.3: Príklad, ktorý ukazuje definovanie nového správania objektu pomocou `Proxy` [6]. V prípade, že objekt neobsahuje atribút, je navrátená hodnota 37.

Tento objekt bol predstavený v štandarde ES6 [14]. Obsahuje originálny objekt a posiela funkcie, ktoré nie sú upravené do originálneho objektu. Pomocou objektu proxy je možné predefinovať správanie a vlastnosti celého objektu 3.3. Alternatívou pre proxy objekt by mohlo byť implementovať celý objekt od znova, avšak tento spôsob je časovo náročný a zložitý na udržiavanie.

JavaScript umožňuje nahradiť všetký atribúty vstavanou funkciou *accessor properties*. Táto funkcia spôsobuje, že každý prístup do atribútu, invokuje definovanú funkciu. V prípade, že atribút už je typu accessor property, tak Chrome Zero iba nahradí funkciu, ktorá je invokovaná. V opačnom prípade je možné všetky atribúty, pretypovať pomocou funckie `Object.defineProperty` [36].

```
1 (function (){
2 var original_function = window.performance.now;
3 window.performance.now = function() {
4     return 0;
5 }
6 })();
7 console.log(window.performance.now(); // 0
8 console.log(original_function.call(window.performance)); // Uncaught ReferenceError
```

Výpis 3.4: Príklad zapuzdrenia novej implementácie do anonymnej funkcie.

Zabezpečenie nezvratnosti

Dôležitou súčasťou je zabezpečiť aby útočník naše bezpečnostné opatrenia nemohol obísť alebo zvrátiť. Preto je potrebné zaručiť aby útočník nemohol pristúpiť k ľubovolnej referencií k pôvodnej konštrukcií. Chrome Zero preto využíva princíp zapuzdrenia pôvodnej referencie, pomocou vytvorenia anonymných odborov platnosti premenných nepriradených k žiadnemu objektu, čo znemožňuje prístup ku kódu z vonkajšieho odboru platnosti premenných. Ako bolo ukázané vo výpise 3.2, útočník by mohol pristúpiť k pôvodnej implementácií funkcie, čomu zapuzdrenie zabraňuje, čo je možné vidieť vo výpise 3.4.

```
1  // Vymazanie novej implementacie
2  (function (){
3  window.performance.now = function() {
4      return 0;
5  }
6  })();
7  console.log(window.performance.now()); // 0
8  delete(window.performance.now); // true
9  console.log(window.performance.now()); // Spravna hodnota
10 //Pouzitie Object.freeze pre zabranenie vymazania
11 (function (){
12 window.performance.now = function() {
13     return 0;
14 }
15 Object.freeze(window.performance);
16 })();
17 console.log(window.performance.now()); // 0
18 delete(window.performance.now); // false
19 console.log(window.performance.now()); // 0
```

Výpis 3.5: Ukážka ako zabrániť zmazaniu novej implementácie.

Stále však existuje jeden možný spôsob, ktorý nie je ošetrený. Útočník by mohol zmazať konkrétnu funkciu pomocou funkcie `delete`, čoho následkom by bolo nahradenie novej implementácie naspäť pôvodnou. Tomu zabraňuje vstavaná funkcia `Object.freeze`, ktorá znemožňuje zmene "zamrazeného"(anglicky freeze) objektu. Fungovanie tejto funkcie je možné vidieť vo výpise 3.5.

3.4 Bezpečnostné opatrenia

Táto sekcia popisuje vybraté bezpečnostné opatrenia a akým spôsobom boli implementované, ktoré ochraňujú užívateľa pred potencionálnymi nástrahami predstavenými v sekcii 3.1. JavaScript Zero predstavuje 10 návrhov, ktoré spoločne plne chránia užívateľa a čiastočne zamedzujú zneužitie exploitov nultého dňa.

Opatrenie \Útok	*Rowhammer.js [19]*	*Page Deduplication [18]*	*DRAM Covert Channel [37]*	*Anti-ASLR [17]*	*Cache Eviction [19] [37]*	*Keystroke Timing [22]*	*Browser [22]*
Buffer ASLR	❍	◗	❍	●	●	❍	❍
Predčítanie pola	●	❍	●	❍	❍	❍	❍
Náhodný prístup do pola	●	◗	◗	●	●	❍	❍
Náhodné mapovanie indexov	❍	●	❍	●	❍	❍	❍
Zníženie presnosti časovej značky	❍	◗	❍	❍	❍	◗	◗
Náhodnosť časových značiek	❍	◗*	❍	❍*	❍	●*	●*
WebWorker polyfill	❍	❍	●	●	●	●	❍
Oneskorenie správ	❍	❍	❍	❍	❍	◗	◗
Spomalenie SharedArrayBuffer	❍	❍	●	◗	●	❍	❍
Zákaz SharedArrayBuffer	❍	❍*	●	●*	●	❍*	❍*

Tabuľka 3.1: Všetky návrhy a konkrétne útoky, ktoré sú riešené. Symbol ● znamená, že návrh úplne bráni útoku. Symbol ◗ indikuje, že útoku je čiastočne zabránené, je ťažšia jeho realizácia a symbol ❍ ukazuje, že opatrenie nemá na útok žiadny vplyv. Znak * označuje opatrenia, ktoré musia byť spoločne implementované k ochrane proti útoku. [36]

V tabuľke 3.1 je možné vidieť, vzťahy medzi navrhnutými opatreniami a útokmi, ktoré práca JavaScript Zero [36] berie do úvahy.

Zabezpečenie polí a stránkovania

Aby sa predišlo zarovnávaniu polí podľa stránok, autori navrhli použiť takzvaný **buffer ASLR**, čo v preklade znamená náhodnosť rozloženia adresného priestoru. Táto technika umožňuje posunúť začiatok pola hocikam na danej stránke, pomocou prepísania konštruktora aby alokoval o 4 KB viac pamäte a následným vygenerovaním náhodného ofsetu v rozmedzí 0 až 4096 B vrátane. To znamená, že útočník sa nemôže spoliehať na fakt, že po-

sledných 12 najmenej významných adresných bitov prvého `ArrayBuffer`u sú nastavené na hodnotu 0.

Útočník však stále môže iterovať cez dostatočne veľké pole a pomocou časovania výpadku stránky detegovať hranice stránky. Jednoduchý spôsob na mitigovanie tohto útoku, je **predčítať** pole po jeho vytvorení, čím sa všetky stránky namapujú do jednotky správy pamäti (MMU), následkom čoho útočník nevie detegovať výpadky stránok.

Namiesto predčítania, je možné upraviť pole tak, aby pri prístupe k prvku sa pristúpilo k ďalšiemu náhodnému prvku do pamäti. Pred tým útočník mohol, počkať pokiaľ sa stránky nedostanú preč z MMU a následne by mohol opäť zistiť hranice stránok. Pri čítaní pamäte je možné pristúpiť k ďalšiemu **náhodnému prvku pola** čo znemožní toto riešenie, pretože útočník sa nemôže spoľahnúť či výpadok stránky nastal na základe náhodného alebo klasického prístupu.

Posledný útok, ktorý súvisí z poliami a predchádzajúce techniky ho neošetrujú sa nazýva de duplikácia stránok (anglicky page deduplication). Útočník musí mať kontrolu nad obsahom aspoň jednej stránky, čo vie zaručiť pomocou vytvorenia dostatočne veľkého pola. Aby sa tomuto útoku dalo zabrániť je potrebné znemožniť útočníkovi deterministicky vybrať obsah celej stránky. Autori pomocou lineárnej funkcie upravujú **náhodné mapovanie indexovanie pola na pamäť**. Fungovanie pola je upravené aby pristúpilo k pamäti s hodnotou $f(x)$, kde x je index pola. Táto funkcia má tvar $f(x) = ax + b \bmod n$, pričom hodnoty a a b musia byť nesúdelitelné čísla a n sa rovná veľkosť pola. Tak isto a a n musia byť nesúdelitelné pre vygenerovanie unikátneho mapovania z indexov na hodnoty pamäte.

Presné časovanie

Jednoduchým spôsobom je **znížiť presnosť** časovačov s veľkou presnosťou. Typicky stačí zaokrúhliť výsledok funkcie na 100 ms, poprípade na sekundy.

Zaujímavejší spôsob je pridanie **náhodného šumu** k časovej značke, čo na prvý pohľad pôsobí, že časovač funguje tak ako by mal. Implementácia spočíva vo vygenerovaní náhodného čísla x v rozsahu 1 ms a momentálnej hodnoty originálnej funkcie n. Pričom sa vypočíta nová hodnota čísla n pomocou funkcie $n = n - (n \bmod x)$, následne sa táto hodnota vráti. Navyše táto technika prináša lepšiu ochranu pred útokmi viď. 3.1.

Pomocou `WebWorker`ov je možné použiť časovače, ktoré majú dostatočnú presnosť na zneužitie. Radikálnym riešením je úplné zabklovanie paralelizmu v prehliadači. Toto je možné dosiahnuť nahradením `WebWorker`ov takzvanými **polyfillmi**, ktoré sa používajú pre pridanie podpory noviniek do starších prehliadačov [5] a pomocou tohto kódu sa pokúsiť napodobniť funkcionalitu na hlavnom vlákne. Tento spôsob blokuje všetky útoky spojené s paralelizmom.

Možnou alternatívou je oneskoriť funkciu `postMessage`. Takéto oneskorenie je možné implementovať pomocou pridania náhodného šumu, podobne ako vo vyššie spomenutom prípade.

Zdielané dáta

Pomocou `SharedArrayBuffer` je možné vytvoriť vlastný časovač, ktorý má v porovnaní s ostatnými časovačmi veľkú presnosť.

Prvou možnosťou je ho **zablokovať**. Keďže v čase písania tejto práce ho využívajú iba 2 prehliadače pre počítače a 1 prehliadač pre zariadenia so systémom Android, čo tvorí približne 30% užívateľov [12], preto by sa momentálne stránky nemali na túto funkcionalitu spoliehať.

Ďalším prístupom ako ochrániť užívateľov je **spomaliť** prístup o náhodnú hodnotu. Takto sa stane časovač nepoužiteľný pre mikroarchitektúrne útoky.

3.5 Zhrnutie práce

Práca [36] predstavuje jednotlivé opatrenia a následne sú implementované v rozšírení Chrome Zero. Hoci rozšírenie bolo vytvorené ako dôkaz koncepcie, z testovania vyplýva, že riešenie je prakticky využiteľné pre denné používane.

Pre testovanie autori poskytli toto riešenie 24 účastníkom a vysvetlili im zámer a funkcionalitu rozšírenia. Následne zúčastnení mali rozoznať či je Chrome Zero na danej stránke zapnutý alebo nie. Celková úspešnosť správneho určenia bola 50.2%. Tak isto bolo rozšírenie podrobené testovaniu výkonnosti, pričom priemerné spomalenie prehliadania bolo 1.82%.

Pri mojom testovaní som narazil na problém na stránke `https://amiunique.org/fp` pri dvoch najvyšších úrovniach ochrany. V tomto prípade nastal problém pri vytváraní nového pola, ktorý následne očakávala funkcia `crypto.getRandomValues`. Avšak po vytvorení pola pomocou konštrukcie `new Uint8Array(1)` sa do tejto funkcie posielal objekt `Proxy`, čo spôsobuje chybu a nefunkčnosť stránky. Podobná chyba sa vyskytuje aj na platforme YouTube[2], kde sa autori rozhodli pri tejto stránky rozhodli neobaľovať objekty.

[2] `www.youtube.com`

Kapitola 4

Návrh rozšírenia implementácie

V tejto kapitole je popísaný návrh implementácie rozšírenia JavaScript Restrictor na základe bezpečnostných opatrení predstavených v práci JavaScript Zero [36]. Hoci rozšírenie Chrome Zero bolo navrhnuté aby mohlo fungovať nezávisle na prehliadači, implementované a testované bolo pre prehliadač Google Chrome. Preto bude potrebné tieto opatrenia implementovať a následne otestovať takým spôsobom aby boli kompatibilné s ostatnými prehliadačmi, ktoré JavaScript Restrictor podporuje. V sekcií 4.1 sú popísané, ktoré opatrenia budú implementované a dôvod prečo ostatné implementované nebudú. Sekcia 4.2 popisuje návrh špecifickej časti opatrení a to konkrétne senzorom zariadenia, možných k zneužitiu. Nakoniec sekcia 4.3 v krátkosti predstaví dopad tejto práce na užívateľské rozhranie rozšírenia.

4.1 Analýza bezpečnostných opatrení

Táto sekcia sa zaoberá analýzou opatrení, ktoré sú vhodné na implementáciu do JavaScript Restrictoru. Z tabuľky 3.1 je možné vidieť, že všetky opatrenia dohromady ochraňujú užívateľa pred všetkými spomenutými útokmi. Avšak niektoré bezpečnostné opatrenia prekrývajú svoju funkcionalitu, čo umožňuje isté opatrenia vynechať.

Zníženie presnosti časových jednotiek je už implementované v JavaScript Restrictore a tak isto môžme vidieť, že nebráni niektorým útokom viď. 3.1. Namiesto toho pridanie šumu umožňuje užívateľa ochrániť napríklad pred sledovaním úderov klávesnice. Preto by malo toto opatrenie plne nahradzovať zaokrúhľovanie hodnôt na preddefinovaných úrovniach.

Náhodné mapovanie indexov ako jediné je schopné zabrániť zneužitiu de duplikácii stránok, avšak ako bolo spomenuté v sekcii 3.5 na najvyšších dvoch úrovniach tento prístup spôsoboval nefunkčnosť stránky. To isté platí aj pre náhodný prístup do pola, ktorý sa využíva pri druhom najvyššom stupni ochrany. Obidve opatrenia budú implementované pre zaručenie bezpečnosti. V prípade podobných problémov s kompatibilitou, aké nastali v Chrome Zero, navrhujem aby sa tieto opatrenia dali použiť pri vlastnom nastavení zabezpečenia a boli použité iba na najvyššej preddefinovanej úrovni bezpečnosti.

Na základe toho sa rozhodne či bude potrebné implementovať opatrenie Buffer ASLR, keďže náhodný prístup do pola rieši rovnaké problémy a bolo by zbytočné zabezpečovať tú istú hrozbu 2 krát.

Predčítanie pola a oneskorenie správ, sú jednoduché spôsoby ako sťažiť útočníkovi zneužitie zraniteľností a zároveň obmedziť jeho možnosti, preto navrhujem aby tieto opatrenia boli implementované a zároveň používane od prvej úrovne bezpečnosti.

Nahradenie `WebWorker` je radikálne riešenie, ktoré síce bráni všetkým útokom spojeným s paralelizmom, ale paralelizmus úplne odstráni. `WebWorkeri` sú dostupný v prehliadačoch, ktoré používa až 97% [13] užívateľov, a preto navrhujem aby toto opatrenie bolo implementované ale použité bude iba na najvyššej úrovni ochrany. Možné následky tohto opatrenia by mohlo byť spomalenie stránok, keďže niektoré sa môžu spoliehať na výpočty na pozadí. Implementovaním ostatných opatrení je možné docieliť rovnaké výsledky a preto na nižších úrovniach oneskorenie správ bude dostačujúcim opatrením.

Spomalenie a zakázanie `SharedArrayBuffer` je možné implementovať avšak je možné tento objekt odstrániť na každej úrovni ochrany a to z dôvodu, že nie je dostupný v populárnych prehliadačoch ako Firefox a Safari. Vo Firefoxe je implementovaný od verzie 57 a v Safari od verzie 10.1, avšak je potrebné ho povoliť v nastaveniach a to z dôvodu možného zneužitia podobnému ako v prípade útoku Spectre [21]. V prípade, že užívateľ chce `SharedArrayBuffer` spomaliť, bude možné tak spraviť vo vlastných úrovniach.

Nakoniec je potrebné volať metódu `Object.freeze()`, ktorá momentálne v JSR chýba. Táto metóda zaručuje nezvratnosť upravených objektov ako bolo popísané v sekcií 3.3.

4.2 Dáta zo senzorov

Súčasťou opatrení navrhnutých v JavaScript Zero [36] je obalenie objektov, ktoré poskytujú informácia priamo zo senzorov daného zariadenia a to konkrétne:

- **Battery status API** - Poskytuje informácie o momentálnom stave batérie zariadenia, času do úplného nabitia/vybitia a stavu pripojenia k nabíjačke.
- **Senzor okolitého svetla** - K tomuto senzoru je možné pristúpiť pomocou správcu udalostí.
- **Pohyb zariadenia** - Podobne, zmenu stavu senzorov poskytujúcich informácie o polohe, orientácií a pohybe zariadenia je možné sledovať pomocou správcu udalostí.

Doteraz tieto opatrenia neboli spomenuté v práci a to preto, že JavaScript Restrictor nie je rozšírenie, ktoré bolo testované pre mobilné zariadenia a ich verziách prehliadačov. Senzory o pohybe zariadenia sa na bežne používaných osobných počítačoch nevyskytujú a senzor okolitého svetla je vo všetkých prehliadačoch, pre ktoré je toto rozšírenie oficiálne dostupné, štandardne zablokované. Hoci sa všetky tieto dáta dajú zneužiť [28, 24], tak navrhujem vo výslednej implementácií zablokovať iba `Battery status API`. Tento senzor je totižto možné zneužiť aj u cieľových užívateľov tohto rozšírenia a API je blokované v prehliadači Firefox od verzie 52 [8] čo nám umožní ho zablokovať a predpokladať, že funkcionalita stránky nebude závislá na tomto rozhraní.

4.3 Užívateľské rozhranie

Implementácia bude priamo naväzovať na zatiaľ nepublikovanú verziu 0.3. V tejto verzií bolo prepracované užívateľské rozhranie a preto táto práca ho nebude iným spôsobom upravovať. Jediná modifikácia, ktorá bude uvedená je pridanie novo-implementovaných opatrení do možností pri vytváraní vlastných úrovní a informácie o ich následkoch.

Kapitola 5

Implementácia opatrení

Kapitola 5 vysvetľuje techniky a postupy, ktoré boli použité pri implementovaní navrhnutých bezpečnostných opatrení. Zaoberá sa tak isto problémami, s ktorými som sa stretol a analyzuje aké nedostatky sa vyskytujú vo výslednej funkcionalite. Sekcia 5.1 sa venuje najkomplexnejšej skupine opatrení z pohľadu náročnosti realizácie a to ochrane proti zneužitiu stránkovania. Druhá sekcia 5.2 zhŕňa všetky potrebné úkony k obmedzeniu presných časovačov. V sekcií 5.3 je vysvetlené riešenie problematiky zdieľania dát a blokovanie senzoru batérie.

5.1 Zabezpečenie stránkovania a bytových polí

Táto sekcia popisuje ako aké techniky boli použité pre zabezpečenie `ArrayBuffer`, často nazývaný aj bytové pole, a pamäte pred možným zneužitím. Avšak tento objekt nie je upravovaný ani obaľovaný žiadnym spôsobom. Pre prácu s dátami uloženými v bytovom poli, je potrebné vytvoriť jeden z dostupných objektov typu `TypedArray` alebo objekt `DataView`. Tieto objekty predstavujú reprezentáciu binárnych dát v špecifickom formáte a preto je potrebné zaobaliť a kontrolovať manipuláciu práve s nimi. V rámci projektu JavaScript Restrictor je možné tieto opatrenia nájsť v súbore `common/wrappingS-ECMA-ARRAY.js`.

Obaľovanie TypedArray

`TypedArray` je prototyp pre skupinu 9 objektov. Každý z týchto objektov je obalený pomocou objektu typu `Proxy` s rovnakými obslužnými rutinami (anglicky handlers) a referencie na ich konštruktér sú nahradené rovnakou funkciou. Obaľovacia funkcia nahrádza pôvodnú, pričom hu pri vykonávaní zavolá pre vytvorenie objektu, to sa však deje iba v prípade, že argument nie je typu `Proxy`.

Následne sa definuje funkcia `offsetF()`, ku ktorej môže pristúpiť iba daný objekt. Ak si užívateľ zvolí, že chce aby bytové pole malo náhodné mapovanie indexov na pamäť, tak sa vygenerujú náhodné vstupy, ktoré každý index mapujú na unikátne miesto v pamäti viď. 3.4 a každý prvok pola zo vstupu sa posunie na nový index. V opačnom prípade táto funkcia iba vracia ten istý index, ktorý jej príde na vstup.

Vytvorený objekt, sa potom obalí pomocou `Proxy` objektu, pričom sú definované 2 handlery, pre prístup k atribútom a handler pre ich nastavovanie. Obidva reflektujú funkcionalitu pôvodného objektu a to využitím objektu `Reflect`, ktorý je prispôsobený k tomu aby sa dal jednoducho použiť pri obalení pomocou `Proxy`, avšak v obidvoch prípadoch, sa vy-

koná navyše jeden náhodný prístup do pamäte. Tak isto týmto spôsobom, je možné interne zamieňať pôvodné indexy na index v pamäti pomocou funkcie `offsetF()`.

Po obalení objektu, je potrebné do objektu pridať metódy `from()` a `of()`, ktoré sa využívajú pri tvorbe nových `TypedArray`s. Keďže je možné ako argument konštruktéra použiť `TypedArray`, obaľovacia metóda zavolá tú pôvodnú a následne hodnotu pošle do nového konštruktéra.

V prípade, že je zapnuté náhodné mapovanie, tak sa každá metóda, ktorá je závislá na správnom poradí prvkov prepíše. Celkovo je potrebné upraviť 19 metód, ktoré rozdeľujeme na 4 typy.

- 1. Funkcia vracia referenciu na pôvodný objekt.
- 2. Funkcia vracia nový objekt, ktorý je samostatne obalený.
- 3. Funkcia vracia špecifickú hodnotu.
- 4. Na rozdiel od predchádzajúcich typov, je funkcia volaná nad novou kópiou pôvodného objektu, s ktorým nie je manipulované. Návratová hodnota je rovnaká ako funkcie typu 2.

Správna funkcionalita je docielená spätným mapovaním indexov na ich pôvodné hodnoty a následným zavolaním pôvodnej funkcie nad bytovým polom. Po úspešnom ukončení volanej funkcie sa opäť prvky uložia do svojho náhodného indexu.

Tu však nastáva prvý nedostatok výslednej implementácie. Metóda `subarray()` vracia referenciu na časť pôvodného pola na ľubovolnom ofsete o zvolenej veľkosti. V prípade, že sa pole upraví na pôvodné poradie, je možné dostať správnu časť pola, avšak po opätovnom mapovaní, sa upraví aj táto referencia, ktorá sa z funkcie vracia. Tým pádom užívateľovi nie je navrátená časť pola od indexu I po index I + N, kde N predstavuje veľkosť pola, ale hodnoty priamo uložené v pamäti na ofsete I až I + N. Toto by bola kritická chyba a to z dôvodu, že by útočník mohol odhaliť mapovanie a toto opatrenie by nebolo funkčné. Riešením by bolo vytvoriť pole, ktoré ukazuje na jednotlivé hodnoty na ofsetoch vypočítaných pomocou funkcie `offsetF()`. Avšak v jazyku JavaScript neexistujú vstavané ukazovatele do pamäte na primitívne dátové typy, v tomto prípade integer, nie je možné predávať referenciou ale iba hodnotou. V momentálnom riešení sa vráti pôvodná referencia na pole. Tým pádom metóda neplní rovnakú funkcionalitu ako pôvodná metóda ale zároveň neodhaľuje vytvorené mapovanie. Pri tomto probléme narážame na problém, kde by bolo potrebné využiť ukazovatele, ktoré však JavaScript neposkytuje a preto pre správne fungovanie, je potrebné zmeniť koncepčný návrh obaľovania.

Na koniec obaľovacej funkcie, sa preiteruje cez celé bytové pole, čím sa všetky stránky načítaju do jednotky správy pamäti a následne sa obalený objekt vráti.

DataView objekt

Druhou možnosťou pre pracú s bytovým polom je objekt typu `DataView`, ktorý poskytuje metódy pre nastavovanie a pristupovanie k číslam na určitom ofsete v pamäti. V tomto prípade som sa rozhodol nepoužiť obaľovanie pomocou `Proxy`. Keďže pre prístup do pamäti sa využívajú iba metódy na základe reprezentácie čísiel, tak sú prepísané jednotlivé metódy, spoločne s konštruktérom objektu.

Metódy, ktoré pracujú s klasickými integermi, či už so znamienkom alebo bez využívajú metódu `getUint8` pre prístup na jednotlivé ofsety v pamäti. Následne sa tieto hodnoty

posunú pomocou bitovej operácie posunu vľavo, sčítajú sa a výsledok sa vráti. Pre prepisovanie hodnôt sa podobne používa metóda `setUint8`. Číslo na vstupe sa konvertuje do jeho bitovej reprezentácie a následne je rozdelené do častí o veľkosti 8 bitov. Potom sú tieto časti jednotlivo nastavené na správne indexy. Podobne ako pri `TypedArrays` na vypočítavanie indexov sa využíva funkcia `offsetF()`, ktorá je definovaná pri vytváraní nového `DataView` objektu.

Desatinné čísla sú v JavaScripte reprezentované pomocou štandardu IEEE 754 [1]. Kde sa pri čítaní pamäte, načíta každý byte čísla a je uložený do pola. Potom je toto pole podľa štandardu transformované na desatinné číslo. Pri funkciách set, sa použije podobný prístup, len v opačnom poradí, kde najprv sa číslo transformuje na pole bytov a následne sa jednotlivé byty vložia do pamäte.

Posledným typom číselnej reprezentácie je objekt `BigInt`, ktorý v `DataView` umožňuje reprezentovať čísla až o veľkosti 64 bitov. Tieto čísla sa odlišujú znakom `n` na konci. Podobne ako pri desatinných číslach sa u čítania z pamäte každý byte uloží do pola. Následne sa využije konštruktér objektu `BigInt`, kde sa pošle toto pole konkatenované ako reťazec v hexadecimálnej sústave. Opačný postup je použitý pri nastavovaní hodnôt, kde sa najprv číslo transformuje na jeho hexadecimálnu reprezentáciu, rozdelí sa na 8 bitové čísla a tie sa uložia do pamäte.

Všetky tieto metódy počítajú so vstupným argumentom predstavujúcim endianitu, ktorý sa používa pri výbere spôsobu písania do pamäti.

Detegovanie objektu Proxy

Ako bolo spomenuté vyššie, pre zaručenie správnej funkcionality náhodného mapovania indexov na pamäť je potrebné detegovať, či parameter funkcie je objekt typu `Proxy`. V JavaScripte toto však prirodzene nie je možné, pretože objekt zaručuje transparentnú virtuálizáciu. Avšak, je to možné docieliť pomocou zabalenia objektu `Proxy` pomocou objektu `Proxy`. Prepísaním konštruktéra, v ktorom je definovaný `Symbol` nazývaný `is_proxy` a následným využitím handleru `has`, je možné túto funkcionalitu docieliť. Handler umožňuje použiť operátor `in`, pomocou ktorého je možné zistiť, či `is_proxy` sa nachádza v objekte.

Symbol `is_proxy` je definovaný v rámci obaleného objektu, tým pádom jeho referencia nie je dostupná pre kód na stránke a tak mimo obaľovacej funkcie nie je možné zistiť, či sa symbol v objekte nachádza. Prepisovanie objektu `Proxy` sa deje iba vrámci lokálneho odboru platnosti premenných objektov `TypedArray`.

5.2 Časovače s vysokým rozlíšením

V tejto sekcií je vysvetlený spôsob akým bolo integrované pridávanie náhodného šumu do presných časovačov, využitím už vytvorenej obálky pre funkciu `performance.now()` a objekt `Date`. Následne je popísané akým spôsobom je pre definovaný objekt `PerformanceEntry`, ktorého atribúty môžu byť tak isto zneužité na časovanie s vysokým rozlíšením.

Náhodný šum

Pre pridávanie náhodného šumu je využité už existujúce obaľovanie pre zaokrúhľovanie výsledkov. Do obálky som pridal parameter, ktorý určuje či sa budú výsledné hodnoty zaokrúhľovať, alebo sa ku ním bude pridávať náhodný šum. Na jeho základe sa prepíše referencia na funkciu, ktorá hodnotu z pôvodnej funkcie upraví. Vygeneruje sa náhodná

hodnota v rozsahu, ktorý si určuje užívateľ a následne sa odčíta od pôvodnej hodnoty, výsledok po operácií modulo s náhodnou hodnotou. Navyše si táto funkcia zapisuje poslednú vrátenú hodnotu a v prípade, že pri ďalšom volaní, je nová hodnota menšia ako tá stará, tak sa vráti predchádzajúca hodnota. To sa deje z toho dôvodu aby naše časovače nepôsobili na užívateľa, že cestuje späť v čase, pričom by ich správanie na stránke bolo rozličné od ostatných užívateľov, vďaka čomu by bolo jednoduchšie rozoznať ich identitu.

Objekt PerformanceEntry

Tento objekt je používaný ako prototyp pre ďalšiu skupinu časovačov. Pri jeho vytvorení sa zapíše do atribútu `startTime` časová značka, kedy bol objekt vytvorený. Tak isto v prípade vytvorenia `PerformanceMark` je v atribúte `duration` uložená hodnota časovej značky, ktorý predstavuje čas ako dlho daná udalosť trvala. Preto je potrebné prepísať deskriptor daných atribútov a to konkrétne prístup k ich hodnotám. Najprv sa načíta pôvodná hodnota, ktorá sa pošle ako parameter do jednej z funkcií, ktoré boli použité aj pri vyššie spomenutých časovačoch. Tento prepísaný deskriptor atribútu následne nahradí ten originálny v prototype objektu.

5.3 Blokované objekty a polyfilly

Sekcia popisuje objekty, ktoré sú úplne zablokované, alebo sú úplne nahradené ich polyfillom, ktorý napodobňuje ich správanie a funkcionalitu.

Úplnému zablokovanie je možné u objektov `SharedArrayBuffer` a `BatteryManager`. Obidva tieto objekty sú zablokované na najnovších verziách prehliadaču Firefox a Safari. Tento fakt nám umožňuje ich zablokovať aj na prehliadači Chrome, keďže obidva prehliadače sú bežne používané a stránky, ktoré by boli závislé na týchto objektoch, by neboli kompatibilné. V prípade, že užívateľ si neželá zablokovať `SharedArrayBuffer`, rozšírenie je schopné pridávať náhodne spomalenie pri prístupoch a pri upravovaní hodnôt zdielaného poľa. Tento prístup aspoň čiastočne dokáže mitigovať, niektoré útoky. Spomalenie je vykonávané pomocou objektu `Proxy`, ktorý definuje handler generujúci náhodne číslo v rozmedzí 0 až 10000 a následne vykoná toľko opakovaní cyklu.

Objekt `Worker` sa na vyššej úrovni ochrany nahrádza polyfillom, ktorý kopíruje vonkajšie správanie objektu, avšak interná funkcionalita sa líši. Najdôležitejším rozdielom je, že je znemožnený skutočný paralelizmus, ktorý môže byť zneužitý na vytváranie vlastných časovačov s vysokým rozlíšením. V prípade, že sa užívateľ rozhodne použiť nižšiu úroveň bezpečnosti, tak je nahradená metóda `postMessage`, ktorá sa používa na posielanie správ a komunikovanie. Táto funkcia pridáva náhodné spomalenie podobným spôsobom ako v `SharedArrayBuffer` a následne zavolá pôvodnú funkciu bez dodatočných zmien.

Kapitola 6

Testovanie funkcionality a spomalenia

Táto kapitola sa zaoberá testovaním implementovaných opatrení. Keďže JavaScript Zero ukazuje, že dané opatrenia sú funkčné a ochraňujú užívateľa pred zneužitím útokov, ktoré sú spomenuté v sekcií 3.4, tak testovanie bude zamerané, či JavaScript Restrictor naozaj opatrenia implementuje a je zaručená ich správna funkcionalita. Výsledky a postup týchto testov je možné nájsť v sekcií 6.1 a zároveň popisuje objavené chyby nájdené vo výslednej implementácií. Následne sekcia 6.2 analyzuje spomalenie stránok dôsledkom týchto opatrení, ktoré je očakávané. Nakoniec v 6.3 je popísané testovanie kompatibility novej implementácie s najznámejšími stránkami, ktoré stránky nefungujú a z akých dôvodov sa to deje.

6.1 Testovanie funkcionality

Obaľovanie objektov, je navrhnuté takým spôsobom aby nebolo možné na stránke ho zvrátiť alebo upraviť, preto pre testovanie správnosti vnútornej funkcionality je potrebné prepísať zdrojový kód priamo v rozšírení. Z tohto dôvodu bolo testovanie prevádzané ručne priamo v konzole prehliadača za pomoci vypisovania testovacích hlášok. Tie umožňujú sledovať zmenu správania objektov, pretože zmena funkcionality by sa mala diať na pozadí a na prvý pohľad by nemala byť viditeľná, či už pre užívateľa alebo pre skripty zo strany servera.

Testované boli jednotlivé opatrenia, najmä upravený prístup do pamäti, náhodnosť časovania a spomalenie komunikácie. Testovanie prebiehalo na operačnom systéme macOS Catalina 10.15.4 a v prehliadačoch Google Chrome verzie 81.0.4044.138 a Firefox 76.0.1.

Najprv som testoval jednotlivé opatrenia postupne a to tak, že bolo vždy aktívne iba obaľovanie jedného objektu a následne na preddefinovanej úrovni 3, kde sú zapnuté všetky opatrenia naraz a poskytujú najvyššiu možnú ochranu.

Pre zachovanie pôvodnej funkcionality objektov som vytvoril jednotkové testy, ktoré ukazujú či výsledná implementácia nemení správanie obalených objektov. Na testovanie som využil známy testovací framework QUnit [1]. Spustiť testy a skontrolovať ich výsledky je možné otvorením súboru `docs/test/unit_test.html` pomocou prehliadača, na ktorom je nainštalované rozšírenie JSR. Frontend je generovaný samotným frameworkom, ktorý navyše importuje súbor `docs/test/tests.js`, ktorom sa nachádzajú jednotlivé testovacie sady. Jednotkové testy navyše umožnia jednoduchší a rýchlejší vývoj pri budúcich úpravách rozšírenia a zaručia zachovanie funkcionality.

[1] https://qunitjs.com/

```
1 // Bug: Interval 0 - target.length - 2
2 var random_idx = Math.floor(Math.random() * (target['length'] - 1));
3 // Opraveny kod: Interval 0 - target.length - 1
4 var random_idx = Math.floor(Math.random() * target['length']);
```

Výpis 6.1: Off-by-one error pri generovaní náhodného indexu

Opravené chyby

- Pri testovaní som objavil off-by-one error viď. 6.1, kde pri generovaní náhodného indexu som zmenšil množinu čísel o 1. Táto chyba (anglicky bug) vznikla kvôli môjmu predpokladu, že funkcia `Math.random()` môže vrátiť aj číslo 1 a v prípade násobenia dĺžkou poľa, by sa mohlo pristupovať na index mimo rozsah.
- U pridávania náhodného šumu do časových značiek, som objavil chybu pri značkách, ktoré nevracajú desatinné čísla, spôsobujú, že šum do výsledného čísla pridá aj náhodnosť za desatinnou čiarkou. z pohľadu ochrany pred zneužitím to nespôsobuje problém, avšak v prípade, detekcie návštevníkov stránok, by používatelia JSR boli jednoducho rozlíšiteľní od bežného návštevníka a mohlo by to byť zneužité na ich stopovanie.
- U `DataView` objektu čísla, ktoré boli reprezentované na 8 alebo 16 bitoch, boli nesprávne ukladané do pamäte. Toto sa dialo z dôvodu, že sa interne každé číslo konvertovalo na svoju 32 bitovú reprezentáciu, ale nastavovalo sa iba 8 alebo 16 bitov. To spôsobovalo pri big endiane, že menej signifikantné bity neboli zapísané do pamäte.
- Tak isto nebolo správne implementované ukladanie do pamäte čísiel typu `BigInt`, kde pri konvertovaní čísla sa k binárnej podobe dopĺňali nulové byty na koniec čísla a nie pred číslo. Toto spôsobovalo, že v prípade 2 bytovej verzie by sa zapísalo do pamäte číslo 0x1100 namiesto čísla 0x0011.
- Nakoniec testovanie odhalilo chybu špecifickú pre prehliadač Firefox. Tu som narazil na vyhadzovanie výnimky pri načítaní stránok, čo spôsobovalo, že obalený kód nebol injektovaný. Rozšírenie sa snažilo obaľovať konštrukcie, ktoré vývojári Firefoxu z bezpečnostných dôvodov odstránili, ako napríklad `navigator.getBattery()`,pričom volanie funkcie `toString()` na hodnotu `undefined` vyvolávalo výnimku.

Testovanie odhalilo iba jednu chybu implementácie, ktorú nebolo možné opraviť, pretože táto chyba je spojená s návrhom a bolo by potrebné prepracovať celé obalenie objektu.

Pri oboch typov objektov pracujúcich s objektom `ArrayBuffer` nastáva problém, keď viaceré reprezentácie pracujú s jedným `ArrayBuffer`om. Každá reprezentácia si pri inicializácií vytvára vlastné mapovanie a tým pádom mapovanie už inicializovaného objektu je nesprávne a pracuje s inými hodnotami. Tento problém by sa čiastočne dalo vyriešiť priradením unikátneho mapovania k bufferu, avšak funkcia navrhnutá v JavaScript Zero, je priamo závislá na veľkosti bufferu, na základe ktorého sa generujú ostatné vstupy funkcie. V prípade, že by sa pracovalo iba s určitou časťou bufferu, tak by musela byť vytvorená nová funkcia, ktorá by mala iné mapovanie a opäť by nastal rovnaký problém.

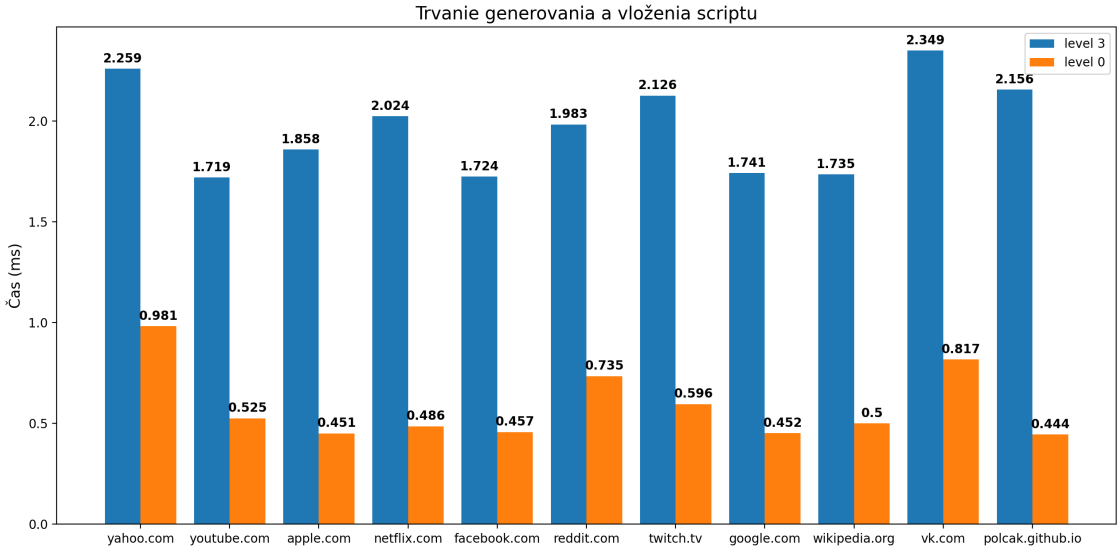
Posledným výsledkom, ktoré testovanie prinieslo, je chýbajúca implementácia metódy `dispatchEvent()` v polyfille objektu `Worker`, ktorý bol prebratý z https://github.com/nolanlawson/pseudo-worker.

6.2 Dopad na výkon stránok

Meranie prebiehalo na rovnakom operačnom systéme ako predchádzajúce testy na prehliadači Google Chrome. Testovaná bola rýchlosť vytvorenia skriptu a jeho vloženie na stránku pri zapnutej vlastnej úrovni, ktorá mala zapnuté všetky opatrenia predstavené a implementované ako súčasť tejto práce. Následne bol tento čas porovnávaný s vkladaním skriptu pri úrovni 0. Týmto spôsobom je možné zistiť, ako môžu tieto opatrenia spomaliť načítanie stránky, avšak neurčujú spomalenie prehliadania. To je totižto priamo úmerné s frekvenciou používania obalených objektov a funkcií, pričom spomalenie je často krát náhodné.

Pre testovanie bola vybratá testovacia stránka rozšírenia a ďalších 9 náhodných stránok, ktoré patria do zoznamu Alexa top 50[2] najnavštevovanejších stránok. Pre meranie bola použitá ešte neobalená metóda `performance.now()`, ktorej výsledok sa následne po vložení na stránku poslal na lokálne bežiace web API, implementované v Python3 pomocou frameworku Flask[3], ktoré ukladalo výsledky na základe URL do súborov. Pre účel testov som upravil rozšírenie a toto správanie nie je pre užívateľov voľne dostupné a replikovatelné.

Z grafu 6.1 je možné vidieť viditeľné spomalenie pri načítaní. Najväčšie priemerné spomalenie je možné vidieť u domény `polcak.github.io`, naopak najmenšie na domovskej stránke domény `youtube.com`. Z dát vyplýva, že priemerné spomalenie je približne 1,38ms čo sa rovná 354.08% nárastu oproti úrovni 0. Čas načítania stránok sa zvýšil avšak táto hodnota je dostatočne malá na to, aby pri bežnom prehliadaní nebola rozpoznateľná.



Graf 6.1: Priemerné spomalenie pri používaní všetkých implementovaných opatrení predstavených v tejto práci, v porovnaní s úrovňou bez opatrení.

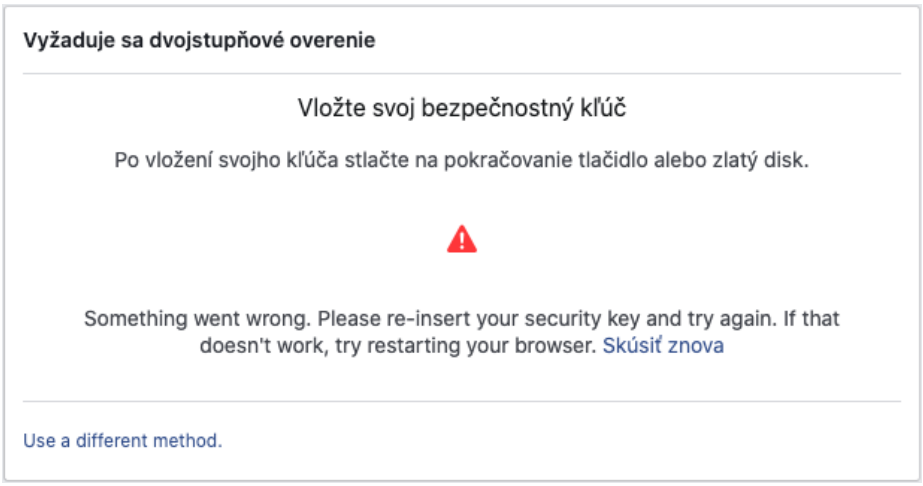
[2] https://www.alexa.com/topsites
[3] https://flask.palletsprojects.com/

6.3 Kompatibilita

Testovanie kompatibility prebiehalo popri mojom bežnom prehliadaní, pričom som používal rozšírenie na vlastnej úrovni, kde boli zapnuté iba nové opatrenia. Pri istých stránkach, popísaných v tejto sekcií, nastávali problémy, podobne ako pri používaní Chrome Zero, avšak pri bežnom prehliadaní som si nevšimol žiadne rozdiely.

Prvý krát som sa stretol s problémom pri sledovaní videí na webe youtube.com,kde sa video nikdy nespustí. Tento problém sa vyskytoval aj v riešení Chrome Zero, kde tvorcovia jednoducho pri tejto doméne obaľovanie objektu `Uint8Array` vynechali. Rovnaká chyba nastáva pri obalení objektu `DataView` v prípade ak je zapnuté náhodné mapovanie indexov na pamäť. Jednotkové testy neodhalili chybu v mapovaní a preto si myslím, že porucha nastáva pri reprezentácií jedného bufferu viacerými objektami. Tento problém je popísaný v sekcií 6.1. V developerskej konzole sa nevyskytujú žiadne chybové hlášky, preto je tento problém komplikované debuggovať. Zaujímavosťou je, že videá v HTML tagu `<iframe>` na iných stránkach je možné spustiť bez problémov.

Pri Google maps[4] som najprv objavil bug pri vytváraní objektu `Date`. V prípade pridávania náhodného šumu, sa posledná hodnota ukladá a následne porovnáva. V konštruktéri je možné určiť parametre, ktoré definujú hodnotu vytvoreného dátumu. Ak parametre predstavovali čas v budúcnosti, tak sa táto hodnota uložila a tým pádom sa dátum vrátil vždy keď sa skript snažil zistiť momentálny čas. Na mapách sa v jednom prípade volá konštruktér s rokom 9999 a tým pádom sa pri ďalších volaniach vždy vracala táto hodnota. Tento problém bol jednoducho vyriešený upravením správania ak konštruktér obsahuje parametre.



Graf 6.2: Chyba pri autentizácií pomocou hardvérového autentifikačného zariadenia.

Tak isto som objavil problém pri autentizácií pomocou YubiKey[5] na sociálnej sieti facebook.com viď. 6.2, kde nebolo možné tento hardvérový autentizátor použiť, čo spôsoboval opäť objekt `Uint8Array`, pričom nezáležalo, či sa používalo mapovanie alebo nie.

Problémy s kompatibilitou nastávali najmä pri prehrávaní videí a pri komplexnejšom využití JavaScriptu ako napríklad mapy. Avšak ostatné testované mapy fungovali bez problémov. Najčastejší problém spôsoboval objekt `Uint8Array`, s ktorým sa stretli aj tvorcovia Chrome Zero a je náročné zistiť čo presne tieto problémy spôsobuje.

[4] https://www.google.com/maps/
[5] https://www.yubico.com/

Kapitola 7

Záver

V rámci tejto práce som sa zoznámil s technikami pre tvorbu webových rozšírení, spôsobmi pre rozpoznávanie návštevníkov webu a ich sledovaní naprieč viacerými stránkami. Táto problematika je popísaná v kapitole 2 a bola jeden z dôvodov pre vznik rozšírenia JavaScript Restrictor. Ďalej som sa zoznámil s prácou JavaScript Zero [36], ktorá popisuje možnosti zneužitia jazyka JavaScript, najmä na mikroarchitektúrne útoky a útoky pomocou časovačov s vysokou presnosťou. Práca taktiež predstavuje bezpečnostné opatrenia, pomocou ktorých je možné predísť útokom ukázaných v tabuľke 3.1.

Niektoré opatrenia boli implementované vrámci projektu Chrome Zero, slúžiace ako dôkaz koncepcie, ktoré sú popísané v rámci kapitoly 3. Cieľom tejto práce bolo analyzovať jednotlivé opatrenia a následne vybrané z nich preniesť do rozšírenia JavaScript Restrictor, pre zvýšenie bezpečnosti užívateľov, pričom nenarušiť kompatibilitu a výkon webových stránok. V kapitole 4 je popísaná analýza opatrení a môj návrh, ktoré opatrenia mali byť implementované a do ktorých základných úrovní ochrany budú zaradené. Postup pri implementácií novej funkcionality rozšírenia je popísaný v kapitole 5. V rámci tejto práce boli implementované všetky potrebné opatrenia, pre zabezpečenie maximálnej bezpečnosti kombinovaním jednotlivých opatrení, hoci v Chrome Zero tieto opatrenia chýbali. Okrem Buffer ASLR boli implementované všetky opatrenia ukázané v tabuľke 3.1. Mimo bezpečnostných opatrení bola opravená chyba, kvôli ktorej rozšírenie nefungovalo na prehliadačoch Google Chrome. Ďalej problém s nastavovaním levelov pre špecifické subdomény a úprava súboru `Makefile` aby fungoval na operačných systémoch macOS. Kapitola 6 popisuje testovanie funkcionality, kompatibility a dopad na načítanie stránok.

Zdrojové kódy je možné nájsť v repozitári projektu[1]. Bezpečnostné opatrenia bude možné používať vrámci verzie rozšírenia 0.3.

V rámci ďalšej práce na rozšírení sa ponúka optimalizácia generovania injektovaného kódu, pretože v momentálnom riešení sa redundantne generujú tie isté časti kódu. Ďalšou možnosťou je úprava mapovania indexov bufferu na pamäť, keďže toto opatrenie spôsobuje zmenu funkcionality objektov.

Možné je rozšíriť JavaScript Restrictor o ďalšie obalenie objektov, ktoré by užívateľa chránili pred ďalšími technikami získavania otlačkov zariadenia, ako napríklad fonty alebo zoznam webových rozšírení. Rozšírenie nie je testované a prispôsobené na mobilné zariadenia, na základe toho neboli niektoré opatrenia implementované a práve to by mohlo byť dobré vylepšenie a námet pre ďalšiu prácu.

[1] https://github.com/polcak/jsrestrictor

Literatúra

[1] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*. 2008, s. 1–70.

[2] *Web Storage in HTML Living Standard*. [online]. Web Hypertext Application Technology Working Group., 2017 [cit. 2019-12-29]. Dostupné z: https://html.spec.whatwg.org/multipage/webstorage.html.

[3] *HTTP cookies* [online]. Mozilla and individual contributors., 2019 [cit. 2019-12-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

[4] *Performance.now()* [online]. Mozilla and individual contributors., 2019 [cit. 2020-01-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Performance/now.

[5] *Polyfill* [online]. Mozilla and individual contributors., 2019 [cit. 2020-01-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Polyfill.

[6] *Proxy* [online]. Mozilla and individual contributors., 2019 [cit. 2020-01-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy.

[7] *ArrayBuffer* [online]. Mozilla and individual contributors., 2020 [cit. 2020-01-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer.

[8] *Battery Status API* [online]. Mozilla and individual contributors., 2020 [cit. 2020-05-09]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Battery_Status_API.

[9] Alabbas, A. a Bell., J. *Indexed Database API 2.0* [online]. World Wide Web Consortium., január 2018 [cit. 2019-12-29]. Dostupné z: https://www.w3.org/TR/IndexedDB-2/.

[10] Belloro, S. a Mylonas, A. I Know What You Did Last Summer: New Persistent Tracking Mechanisms in the Wild. *IEEE Access*. Institute of Electrical and Electronics Engineers (IEEE). 2018, roč. 6, s. 52779–52792. Dostupné z: https://doi.org/10.1109/access.2018.2869251.

[11] Biernátová, O. a Skůpa, J. *Measuring digital development* [online]. Geneva: International Telecommunication Union, 2019 [cit. 2019-12-29]. Dostupné z: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2019.pdf.

[12] Deveria, A. a Schoors, L. *Can I Use* [online]. [cit. 2020-01-15]. Dostupné z: https://caniuse.com/#feat=sharedarraybuffer.

[13] Deveria, A. a Schoors, L. *Can I Use* [online]. [cit. 2020-01-15]. Dostupné z: https://caniuse.com/#search=WebWorker.

[14] Ecma International. *ECMAScript 2015 Language Specification*. 6th. Geneva: [b.n.], June 2015. Dostupné z: http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf.

[15] Englehardt, S. *No boundaries: Exfiltration of personal data by session-replay scripts* [online]. Center for Information Technology Policy (CITP), Princeton University, 2017 [cit. 2020-01-07]. Dostupné z: https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts/.

[16] Fouad, I., Bielova, N., Legout, A. a Sarafijanovic-Djukic, N. Tracking the Pixels: Detecting Web Trackers via Analyzing Invisible Pixels. *CoRR*. 2018, abs/1812.01514. Dostupné z: http://arxiv.org/abs/1812.01514.

[17] Gras, B., Razavi, K., Bosman, E., Bos, H. a Giuffrida, C. ASLR on the Line: Practical Cache Attacks on the MMU. In:. Február 2017.

[18] Gruss, D., Bidner, D. a Mangard, S. Practical Memory Deduplication Attacks in Sandboxed Javascript. In:. September 2015.

[19] Gruss, D., Maurice, C. a Mangard, S. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In:. Júl 2016, s. 300–321. ISBN 978-3-319-40666-4.

[20] Kaplas, J. *Possibilities and usability of various privacy preservertion browser add-ons*. Lappeenranta, 2016. Diplomová práca. Lappeenranta University of Technology. Degree Program in Computer Science. Vedoucí práce Jari Porras.

[21] Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D. et al. Spectre Attacks: Exploiting Speculative Execution. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, s. 1–19.

[22] Lipp, M., Gruss, D., Schwarz, M., Bidner, D., Maurice, C. et al. Practical Keystroke Timing Attacks in Sandboxed JavaScript. In:. August 2017, s. 191–209. ISBN 978-3-319-66398-2.

[23] Macbeth, S. *Tracking the Trackers: Analysing the globaltracking landscape with GhostRank* [online]. 2017 [cit. 2019-12-26]. Dostupné z: https://www.ghostery.com/wp-content/themes/ghostery/images/campaigns/tracker-study/Ghostery_Study_-_Tracking_the_Trackers.pdf.

[24] Mehrnezhad, M., Toreini, E., Shahandashti, S. a Hao, F. TouchSignatures: Identification of User Touch Actions and PINs Based on Mobile Sensor Data via JavaScript. *Journal of Information Security and Applications*. Január 2016, roč. 26.

[25] Molly Hanson, P. L. a Macbeth, S. *The Tracker Tax: the impact of third-partytrackers on website speed in the United Statesk* [online]. Máj 2018 [cit. 2019-12-29]. Dostupné z: https://www.ghostery.com/wp-content/themes/ghostery/images/campaigns/tracker-tax/Ghostery_Study_-_The_Tracker_Tax.pdf.

[26] Mozilla.org. *Hardware Across the Web.* [online]. [cit. 2020-01-08]. Dostupné z: https://data.firefox.com/dashboard/hardware.

[27] Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F. et al. Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In:. Máj 2013, s. 541–555. ISBN 978-1-4673-6166-8.

[28] Olejnik, L., Acar, G., Castelluccia, C. a Diaz, C. The Leaking Battery. In:. Január 2016, s. 254–263. ISBN 978-3-319-29882-5.

[29] Oren, Y., Kemerlis, V. P., Sethumadhavan, S. a Keromytis, A. D. The Spy in the Sandbox. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. ACM Press, 2015. Dostupné z: https://doi.org/10.1145/2810103.2813708.

[30] Pessl, P., Gruss, D., Maurice, C., Schwarz, M. a Mangard, S. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, August 2016, s. 565–581. Dostupné z: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/pessl. ISBN 978-1-931971-32-4.

[31] Polčák, L. *JavaScript Restrictor* [online]. [cit. 2019-12-31]. Dostupné z: https://polcak.github.io/jsrestrictor/.

[32] Q Success. *Historical yearly trends in the usage of client-side programming languages for websites* [online]. 2019 [cit. 2019-12-26]. Dostupné z: https://w3techs.com/technologies/history_overview/client_side_language/all/y.

[33] Rajul, Y., Babu, D. S. a Anuradha, K. Analysis on Periodic Web Personalization for the Efficiency of Web services. In: *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*. IEEE, Apríl 2018. Dostupné z: https://doi.org/10.1109/icicct.2018.8473276.

[34] Roesner, F., Kohno, T. a Wetherall, D. Detecting and defending against third-party tracking on the web. In:. Apríl 2012, s. 12–12.

[35] Schwarz, M., Lackner, F. a Gruss, D. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. Dostupné z: https://www.ndss-symposium.org/ndss-paper/javascript-template-attacks-automatically-inferring-host-information-for-targeted-exploits/. ISBN 1-891562-55-X.

[36] Schwarz, M., Lipp, M. a Gruss, D. JavaScript Zero: Real JavaScript and Zero Side-Channel Attacks. In:. Január 2018.

[37] Schwarz, M., Maurice, C., Gruss, D. a Mangard, S. Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript. In:. Január 2017, s. 247–267. ISBN 978-3-319-70971-0.

[38] Timko, M. *Vylepšení rozšíření pro omezení volání JavaScriptu.* Brno, 2019. Diplomová práca. Vysoké učení technické v Brně. Fakulta informačních technologií. Vedoucí práce Libor Polčák.

[39] Červinka, Z. *Rozšíření pro webový prohlížeč zaměřené na ochranu soukromí.* Brno, 2018. Diplomová práca. Vysoké učení technické v Brně. Fakulta informačních technologií. Vedoucí práce Libor Polčák.