VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

BĚHOVÁ PROSTŘEDÍ PRO TESTOVÁNÍ ČINNOSTI ROZŠÍŘENÍ PRO WEBOVÝ PROHLÍŽEČ

RUN-TIME ENVIRONMENTS FOR BROWSER EXTENSION TESTING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JANA PETRÁŇOVÁ**
AUTHOR

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

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

155960

Ústav:	Ústav informačních systémů (UIFS)
Studentka:	**Petráňová Jana, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Počítačové sítě
Název:	**Běhová prostředí pro testování činnosti rozšíření pro webový prohlížeč**
Kategorie:	Web
Akademický rok:	2023/24

Zadání:

1. Seznamte se s integračními a systémovými testy projektu JShelter a porovnejte možnosti využití různých konfigurací s jinými možnostmi virtualizace a podobnými přístupy.
2. Nastudujte obranné možnosti rozšíření JShelter a porovnejte je s rozšířeními zaměřenými na bezpečnost a ochranu soukromí (např. uBlock Origin, PrivacyBadger, NoScript, DuckDuckGo Privacy Essentials, Ghostery, Netcraft, Decentraleyes). Vyberte rozšíření, která JShelter dobře doplňují a identifikujte rozšíření, která se funkcionalitou překrývají.
3. Navrhněte prostředí, které by umělo otestovat činnost JShelteru v různých konfiguracích, jak v integračních, tak systémových testech. Vytvořte obecný návrh, aby v běhových prostředích byla otestovatelná i funkcionalita jiných rozšíření.
4. Po konzultaci s vedoucím návrh implementujte.
5. Implementaci otestujte a porovnejte s původní implementací integračních a systémových testů. Vyhodnoťte činnost JShelteru v kombinaci s jinými rozšířeními.
6. Svůj přínos vyhodnoťte a navrhněte možná pokračování projektu.

Literatura:

- Libor Polčák, Marek Saloň, Giorgio Maone a kol. JShelter: Give Me My Browser Back. *Proceedings of the 20th International Conference on Security and Cryptography*. Řím: SciTePress - Science and Technology Publications, 2023, s. 287-294. ISBN 978-989-758-666-8
- Martin Bednář. Automatické testování projektu JavaScript Restrictor. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií
- A. Datta, J. Lu a M. C. Tschantz. Evaluating Anti-Fingerprinting Privacy Enhancing Technologies. *The World Wide Web Conference*. San Francisco, USA: ACM, 2019, s. 351–362.

Při obhajobě semestrální části projektu je požadováno:
První 3 body zadání včetně konzultace návrhu s vedoucím práce.

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:	1.11.2023
Termín pro odevzdání:	17.5.2024
Datum schválení:	30.10.2023

Abstrakt

Tato práce přibližuje problematiku sledování uživatele při pohlížení internetu a jmenuje nejvíce populární rozšíření prohlížeče se zaměřením na bezpečnost a soukromí. Soustředí se na projekt *JShelter* a jeho testy, jejichž ideologií se inspiruje při implementaci praktické části práce. Zmiňuje projekt *PETInspector*, jehož implementační části převezme a upraví pro potřeby výsledného běhového prostředí. Definuje požadavky na běhové prostředí a popisuje jeho návrh. Implementaci návrhu realizuje a popisuje její zajímavé aspekty. Pro výslednou implementaci testuje splnění definovaných podmínek a nakonec shrnuje testy provedené pomocí výsledného běhového prostředí a popisuje jejich výsledky.

Abstract

This thesis briefly describes how users are being tracked across the internet while browsing and names the most popular browser extensions focusing on security and privacy. It focuses on the *JShelter* project and its testing, which is used as a fundamental basis for the proposed implementation of the run-time environment. It also mentions the *PETInspector* project, which is redone and used as a framework for testing. It presents a possible design of the run-time enviroment implementation, which is later implemented. The implementation is summarized and used for various tests, including those that affirm its ability to fulfil defined conditions. Finally, the tests results are presented and interpreted.

Klíčová slova

rozšíření pro prohlížeč, bezpečnost, soukromí, internet, sledovací prvky, automatizace, testování, běhové prostředí

Keywords

browser extensions, security, privacy, internet, tracking, automatization, testing. run-time environment

Citace

PETRÁŇOVÁ, Jana. *Běhová prostředí pro testování činnosti rozšíření pro webový prohlížeč.* Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Běhová prostředí pro testování činnosti rozšíření pro webový prohlížeč

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením pana Ing. Libora Polčáka Ph.D. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Jana Petráňová
14. května 2024

Poděkování

Ráda bych poděkovala panu Ing. Liboru Polčákovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování diplomové práce.

Obsah

Kapitola 1

Úvod

Proces prohlížení internetových stránek vyvolává v uživatelích pocit soukromí a bezpečí. I když tomu tak ve fyzické situaci může opravdu být, na internetu je opak pravdou [46]. Při návštěvách stránek po sobě uživatel zanechává stopy, které mohou vést k jeho opětovné identifikaci mezi sezeními prohlížení [36]. Použití internetu přestává být anonymní. Co víc, nemusí jít pouze o navštívené stránky. I třetí strany při prohlížení získávají osobních údaje uživatele. Prohlížením nedobrovolně sdělená data jsou poté zneužita, typicky pro tvorbu cílené reklamy, která uživatele po internetu pronásleduje [46].

Přestože v dnešní internetové sféře není soukromí standardem, má na něj každý uživatel v rámci prohlížení internetu právo [2]. Navzdory tomu není kvůli problémům v jeho interpretaci takové právo zcela uplatnitelné v praxi [35]. Je potřeba převzít vlastní iniciativu a aktivně vyvíjet a používat prostředky, jako jsou například rozšíření prohlížeče, které omezí třetím stranám nabourávat soukromí uživatele a zneužívat jej pro vlastní zisk.

Pokud uživateli nevyhovuje, jak je nakládáno s jeho osobními údaji, může využít dostupných webových rozšíření, která se specializují na ochranu jeho soukromí a bezpečí. Taková rozšíření vznikají proprietárně, nebo i pro použití zdarma. Jejich vývojáři však čelí neustále se měnícímu prostoru internetu. S novou verzí prohlížeče přichází další úskalí spojená s vývojem rozšíření, stejně tak se rapidně vyvíjí techniky, které implementují sledování uživatele za pomoci sledovacích prvků na stránkách, které navštěvuje.

Je tedy žádoucí, aby existoval nástroj, který dovolí vývojářům a uživatelům otestovat webová rozšíření a to, jak se konkrétní rozšíření ovlivňují. Nástroj by měl být odolný vůči rychlým změnám ve verzi ovladačů prohlížečů. Měl by být dobře oddělený na funkční celky pro jednodušší úpravy a univerzálně použitelný pro jakákoliv rozšíření a jejich kombinace. Výstupy testů nástroje by měly být konzistentní, deterministické a předvídatelné, což dovolí vývojářům analyzovat chování jejich implementace.

Cílem této práce je uvést čtenáře do problematiky sledování uživatele na internetu a implementace vhodného běhového prostředí, které vývojářům i uživatelům bezpečnostních webových rozšíření usnadní testování a porovnání rozšíření a jejich kombinací.

Kapitola 2

Bezpečnostní rozšíření prohlížeče

Tato kapitola uvede čtenáře do problematiky tvorby otisku prohlížeče a nastíní, proč je třeba se před takovými metodami v rámci prohlížení internetu chránit. Dále představí pro práci stěžejní projekt *JShelter* a jeho metodiku při boji proti tvorbě otisku prohlížeče uživatele. Uvede několik dalších bezpečnostních rozšíření dostupných na internetu a shrne jejich přístupy k ochraně uživatele v průběhu prohlížení. Nakonec zmíní projekt *PETInspector*, jehož implementace je pro tuto práci myšlenkově zajímavá.

2.1 Tvorba otisku prohlížeče

Termínem tvorba otisku prohlížeče, v angličtině *fingerprinting*, označujeme proces vytvoření unikátního otisku prohlížeče uživatele při každodenním prohlížení webu pomocí identifikace hardware použitého zařízení, operačního systému uživatele a využitého prohlížeče (*Google Chrome*, *Mozilla Firefox* a další), společně s jeho konfigurací [36]. Takový otisk může být využit za účelem identifikace uživatele při opakovaných návštěvách webů a domén. Identifikace probíhá i bez uživatelova přihlášení k prohlížené stránce. Vytváření otisku často probíhá bez uživatelova souhlasu [53], který dokonce nemusí být ani nutný v případě, že vytváření otisku probíhá při snaze zkvalitnění služeb zprostředkovaných uživateli, jako je ochrana uživatele proti podvodům.

Otisk se skládá z více charakteristik udávaných nejen prohlížečem, ale i zařízením, které uživatel k prohlížení internetu využívá. Charakteristiky samy o sobě jsou sdíleny velkým počtem uživatelů internetu, ale jejich kombinace vytváří pro něj unikátní otisk, který lze při opakované návštěvě rozpoznat.

Metody tvorby otisku

Otisk je vytvořen pomocí sesbírání atributů uživatele – prohlížeč, operační systém a jiné další speciální atributy. Ty jsou čteny aktivně, například pomocí webových rozhraní *API – Application Programming Interface*, která vytvářejí vstupní bod pro specifické funkce a objekty. Díky využití jazyka *JavaScript* a jeho jednoduchých volání je sběr informací triviální. Otisk lze vytvořit i staticky zkoumáním hlaviček protokolu *HTTP* nebo *HTTPS*. Dále jsou agregovány, analyzovány a přetvořeny do otisku. Nejčastěji sbírané atributy jsou například:

- *canvas element* a *WebGL*,
- seznam písem dostupných v zařízení,

- zásuvné moduly prohlížeče,
- operační systém a jeho verze,
- prohlížeč a jeho verze,
- rozlišení obrazovky,
- obsah hlaviček protokolu *HTTP/HTTPS*, včetně výpočtu MD5 hashe informací získaných pomocí extrakce dat z TLS paketů při ustanovení spojení [38],
- nastavení jazyka a GPS souřadnice.

Výsledkem je unikátní otisk s dostatečnou úrovní entropie [54] k rozeznání uživatele skrze vícero nezávislých sezení prohlížeče.

Využití otisku prohlížeče

Otisk prohlížeče může být využit k rozeznání sítí botnet, nebo odhalení ilegálních praktik například v bankovním sektoru [23][1]. Často je však využit za účelem tvorby reklamy cílené na uživatele webu [46]. Provozovatel služeb, které uživatel navštíví, má přístup k historii prohlížení uživatele. Sledující prvky stránek jsou schopny rozpoznat jeho zájmy, koníčky, v mnoha případech dokonce věkovou skupinu a další citlivé informace. Tyto informace mohou provozovatelé stránek použít pro svoje vlastní účely nebo je prodat obchodníkům s informacemi, kteří následně vytváří uživatelův profil [51]. Profil dál prodávají tvůrcům internetových reklam, které se při prohlížení uživateli v prohlížeči zobrazují. Cílená reklama je velice efektivní. Až 91% konzumentů cítí vyšší zájem ke koupi produktu, pokud se s nimi prodejce snaží navázat personalizovanou komunikaci [19]. Legislativa však uživatele chrání před druhotným použitím otisku k jiným účelům [3]. Tyto praktiky jsou dnes natolik běžné, že cílenou reklamu při prohlížení vnímá každý běžný uživatel, který nevyužívá žádná bezpečnostní rozšíření. Dochází tak k naprostému porušení uživatelova soukromí, mnohdy i bezpečí.

2.2 Projekt JShelter

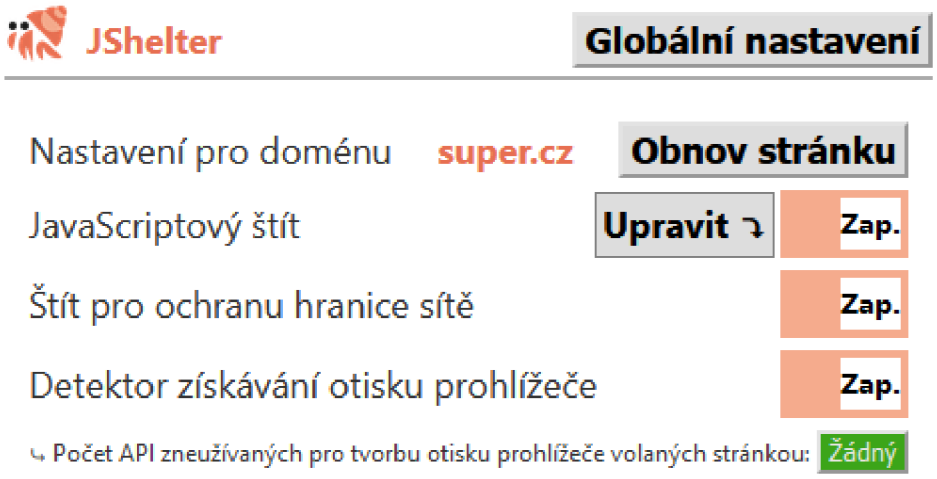
Tato sekce se soustředí na projekt *JShelter*, který je zásadní pro rámec celé práce. Popíše rozšíření a jeho nejzákladnější principy. Dále shrne princip a důležitost automatických testů pro projekt *JShelter* implementovaných v rámci diplomové práce pana Bednáře [8].

JShelter je bezpečnostní webové rozšíření, jehož cílem je poskytnout uživatelům schopnost bránit své vlastní soukromí a získat zpátky kontrolu nad svými osobními údaji [32]. Rozšíření chrání uživatele před tvorbou otisku jeho prohlížeče a dalšími praktikami, které ohrožují uživatelovo bezpečí a soukromí.

Rozšíření kontroluje a upravuje data, která získávají internetová *API* o uživateli během prohlížení webu. Představuje bezpečnostní vrstvu mezi uživatelem a navštívenou stránkou a zároveň dává uživateli na výběr, které akce ve funkčnosti omezí, nebo zakáže úplně [32]. Taková ochrana je účinná nejen proti tvorbě otisku prohlížeče, ale i proti útokům, které jsou cíleny proti specifickým zařízením a jejich prohlížečům, nebo operačním systémům. *JShelter* nabízí uživateli možnost vytvářet i úrovně vlastní pro specifické potřeby uživatele.

[1] Přístup k bankovnímu účtu z více nezávislých zeměpisných souřadnic může indikovat podvodnou činnost.

Na obrázku 2.1 lze vidět grafické uživatelské rozhraní rozšíření. Rozhraní umožňuje uživateli tlačítky jednoduše zapnout a vypnout ochranu nabízenou rozšířením a obnovit stránku v případě změny nastavení. Dále lze skrze tlačítko *Globální nastavení* přistoupit k nastavení rozšíření a upravit úroveň ochrany, kterou uživateli poskytuje.



Obrázek 2.1: Grafické uživatelské rozhraní rozšíření *JShelter* během návštěvy stránky *super.cz*.

Fingerprinting Detector

Jeden ze způsobů, kterým bojuje rozšíření *JShelter* s tvorbou otisku uživatelova prohlížeče, je mechanismus detekce tvorby otisku prohlížeče. Mechanismus monitoruje webová *API*, která jsou často využívána k tvorbě otisku prohlížeče a aplikuje heuristický přístup k rozeznání pokusu o tvorbu otisku v reálném čase. Pokud *JShelter* zaznamená pokus o tvorbu otisku, upozorní uživatele, který může následně blokovat budoucí *HTTP* požadavky inicializované stránkou obsahující sledovací prvky. Takové chování zamezí stránce získávat celý, nebo částečný otisk prohlížeče uživatele [43]. Rozšíření detekované akce zobrazuje ve svém uživatelském rozhraní.

JavaScriptový štít

Projekt *JShelter* implementuje mechanismus pro omezení a znesnadnění získání atributů voláním webových *API* použitím jazyka *JavaScript*. Vnitřní implementace realizuje omezení pomocí *wrapperů* – útržků kódu obalujících původní chování volání *JavaScriptových API* tak, aby zamezil tvorbě otisku prohlížeče s vysokou entropií. Chování *wrapperů* lze obecně rozdělit na tři typy [29]:

- Redukce přesnosti volání – hodnota získaná voláním *API* je ve většině případů příliš přesná a pro správný chod webové stránky není potřeba vracet hodnotu tak konkrétní.
- Falšování odpovědí – na *API* dotaz odpoví *JShelter* zfalšovaným atributem, aby zamezil tvorbě otisku prohlížeče. Rozšíření odpovídá hodnotami, které jsou odlišné pro stránky s různým *origin*. Dále odpovídá zfalšovanými hodnotami i po návštěvě

stejné stránky v rámci jiného sezení prohlížeče. Tato metoda *malých lží* značně komplikuje schopnost sledovacích prvků tvořit konzistentní otisk prohlížeče uživatele a tím jej identifikovat při pohybu skrze internetové stránky [43].

- Skrývání dat – pokud data nejsou obvykle potřebná k chodu stránky, může *JavaScriptový* štít zamezit jejich čtení vrácením prázdného atributu, chyby, nebo *API* zablokovat úplně.

Úrovně ochrany

JShelter podporuje čtyři základní úrovně ochrany [31]. Vlastní nastavení a přidání nové úrovně lze realizovat v uživatelském rozhraní přímo v prohlížeči. Ke každé metodě je přidán krátký komentář, který uživateli usnadní zvolení úrovně v závislosti na jeho preferencích. Přednastavené úrovně ochrany jsou:

- Úplné vypnutí *JavaScriptového* štítu – lze použít pro stránky, kterým uživatel důvěřuje a chce jejich funkcionality využívat naplno.
- Vypnutí obrany proti tvorbě otisku prohlížeče – ochrana bez omezení základní funkčnosti stránky bez ochrany proti tvorbě otisku prohlížeče, zakázání netypických *API*. Doporučeno v případě, že stránka je důvěryhodná a vyšší úrovně omezují její funkcionalitu, zvyšují čas načítání, nebo omezují samotný běh stránky. Vypnutím ochrany lze docílit rychlejšího načtení stránky a redukce výpočetní náročnosti prohlížení.
- Doporučená úroveň ochrany – mírně modifikuje výsledky volání *API* tak, aby nebyla možná identifikace uživatele napříč stránkami. Generované hodnoty volání se mění při každém restartu prohlížeče. Ihned po instalaci se jedná o výchozí úroveň ochrany.
- Striktní úroveň ochrany – využití všech metod ochrany proti tvorbě otisku prohlížeče. Volání *API* úplně blokuje, nebo vrací prázdné, nesmyslné nebo vzácné hodnoty. Cílem této úrovně je naprosto zamezit navštíveným stránkám přístup k atributům uživatele, které by mohly vést k odhalení detailů prohlížeče.

Stávající testy projektu

V rámci diplomové práce pana Bednáře [8] byly pro projekt *JShelter* vytvořeny komplexní automatické testy na třech různých úrovních, přičemž lze každou úroveň testovat nezávisle na sobě nebo na podpůrných nástrojích úrovní jiných.

Jednotkové testy

Testy tohoto typu zodpovídají za kontrolu definicí funkcí a očekávaných hodnot pro zadaný vstup. Testují základní funkcionality rozšíření. Samotné testy jsou implementovány v rámci *Jasmine* [1].

Každý vstup funkce při testování odpovídá očekávané výstupní hodnotě, nebo intervalu hodnot. Ověření funkčnosti funkcí je založeno na jejich opakovaném volání. Pro každou funkci vzniká sada jednotkových testů, která ověřuje, zda je funkce definovaná, vrací očekávaný datový typ, vrací výjimku, nebo očekávané hodnoty pro běžné a krajní vstupní

hodnoty. Ke každému zdrojovému souboru rozšíření *JShelter* je přiřazena komponenta odpovídající jeho jednotkovým testům. Hlavní komponentou v balíčku jednotkových testů je spouštěč provedení jednotkového testování.

Integrační testy

Cílem integračních testů je ověřit, zda rozšíření *JShelter* lze sestavit do balíčku k importu a nasadit jej do testovaného webového prohlížeče. Kontrolují, jestli *JShelter* při volání webových *API* vrací podvržené hodnoty tam, kde se očekávají, a nebrání poskytnout reálné hodnoty při volání konkrétních koncových bodů *API* prohlížeče.

Při spuštění integračních testů probíhá prvotní sestavení rozšíření do publikovatelného balíčku pro konkrétní testovaný prohlížeč ze zdrojového kódu repositáře. Po úspěšném sestavení probíhá kontrola na správnost vracených dat podle aktuálního nastavení úrovně ochrany rozšíření.

Integrační testy jsou automatizované nástrojem *Selenium WebDriver* [48] a na rozdíl od jednotkových testů implementované v jazyce *Python* s využitím rámce *pytest*. Stěžejní je pro ně soubor *config.py*, který udává detaily průběhu testů. Pro testy lze zvolit různé úrovně ochrany rozšíření, typ testovaného prohlížeče (momentálně podporují prohlížeče *Google Chrome* a *Mozilla Firefox*) a umístění souborů nutných pro spuštění automatického prohlížení, na základě kterého je vyhodnocena funkcionalita rozšíření. Při prohlížení jsou sbírány reálné návratové hodnoty, které jsou dále porovnávány s očekávanými návratovými hodnotami v závislosti na testovaném prohlížeči a úrovni ochrany. V mnoha případech se nejedná pouze o konstantní hodnoty, ale i o intervaly hodnot s určitou přesností (viz sekce 2.2).

Systémové testy

Systémové testy mají za úkol ověřit, zda rozšíření *JShelter* neomezuje chtěnou funkcionalitu navštěvovaných stránek. Ověřují, zda program funguje správně jako celek. V průběhu simulují potenciální prohlížení uživatelů webu při použití rozšíření pomocí automatizované návštěvy nejfrekventovanějších webových stránek z oficiálních žebříčků. Počet testovaných webových stránek tvoří vhodný reprezentativní vzorek, jehož velikost se odvíjí od časové náročnosti realizace testů. Testovací prostředí využívá nástroj *Selenium Grid* [47], který operuje na principu klient (testovací uzly) – server (řízení automatizovaného testování). Implementace testů je v jazyce *Python*.

Testy se soustředí na sbírání dvou typů dat – výpisy konzole prohlížeče (pro zkoumání varovných a chybových výpisů) a pořízené snímky obrazovky (pro zachycení vizuální informace o výsledku pokusu načíst testovanou stránku) při automatizovaném prohlížení. Při výskytu chybové, nebo varovné hlášky při použití rozšíření *JShelter* lze předpokládat, že rozšíření narušilo chtěnou funkcionalitu stránky. Stejně tak v případě snímků obrazovky, porovnáním vzhledu stránky bez použití a s použitím rozšíření získáním jejich rozdílu lze vyhodnotit dopad rozšíření na prohlížení stránek.

2.3 Další bezpečnostní rozšíření pro webové prohlížeče

Tato sekce představí dnes nejvíce používaná bezpečnostní webová rozšíření a další rozšíření zajímavá pro projekt *JShelter*. Soustředí se na jejich způsob zprostředkování ochrany a následně jej porovná s technikami rozšíření *JShelteru*.

Nejužívanější webové prohlížeče jako *Mozilla Firefox* a *Google Chrome* poskytují uživatelům možnost instalace webových rozšíření pomocí oficiálních stránek pro webové doplňky. V rámci této diplomové práce bylo vybráno devět bezpečnostních rozšíření v závislosti na jejich přístupu k boji proti tvorbě otisku prohlížeče. Všechna rozšíření jsou v době vzniku práce pravidelně aktualizována a jejich vývoj v současnosti pokračuje. Tabulka 2.1 znázorňuje vybraná rozšíření a jejich podporu v prohlížečích společně s celkovým počtem stažení skrze všechny prohlížeče, ve kterých jsou rozšíření dostupná. Počet stažení bere v potaz pouze desktopová zařízení.

Název rozšíření	Podporované prohlížeče	Celkový počet stažení
uBlock Origin	MF, GC, Edge, Opera	62 530 905
Privacy Badger	MF, GC, Edge, Opera	3 345 657
NoScript	MF, GC	412 736
DuckDuckGo Privacy Essentials	MF, GC, Edge, Opera, Safari	9 831 746
Ghostery	MF, GC, Edge, Opera, Safari	10 130 821
Netcraft	MF, GC, Edge, Opera	126 558
Decentraleyes	MF, GC, Edge, Opera	558 408
CanvasBlocker	MF	26,138
Canvas Blocker	GC	20,000

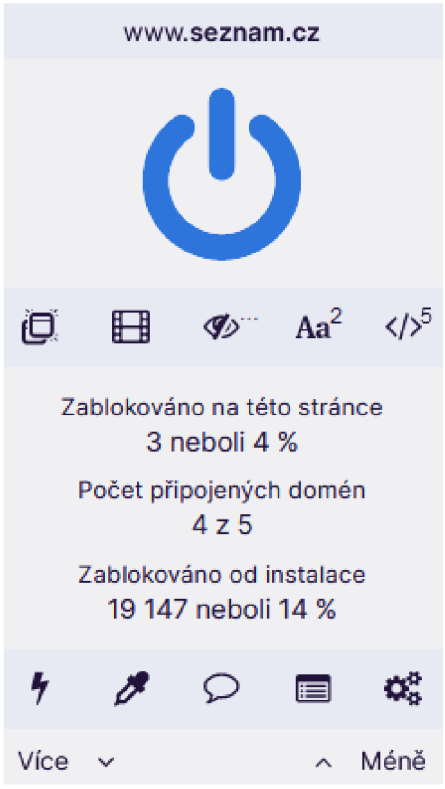
Tabulka 2.1: Tabulka vybraných rozšíření společně s podporovanými prohlížeči a počtem stažení. MF – *Mozilla Firefox*, GC – *Google Chrome*.

uBlock Origin

Rozšíření *uBlock Origin* [20] je v rámci práce nejpoužívanějším bezpečnostním rozšířením ze všech zmíněných. Ve vývoji je od roku 2014 a původně začalo jako projekt modifikace rozšíření *uMatrix* vývojáře *Raymonda Hilla*. Dnes se na jeho vývoji kromě hlavního vývojáře podílí mnohačlenný tým, včetně vývojáře *Nika Rollse*, díky kterému je dnes rozšíření dostupné i pro prohlížeč *Microsoft Edge*. Jeho uživatelské rozhraní na obrázku 2.2 dovoluje rozšíření jedním kliknutím zapnout a vypnout. Poskytuje uživateli informace relevantní k blokování obsahu stránky a nabízí využití nástrojů, jako je inspekce elementů stránky a tlačítko odkazující na nastavení rozšíření.

uBlock Origin pracuje na bázi seznamů sledovacích prvků, které fungují podobně jako černé listiny blokující nedůvěryhodný obsah od načtení na stránce, již uživatel momentálně prohlíží. Bližší princip jejich fungování je detailněji popsán v mé bakalářské práci [42].

Rozšíření se nezaměřuje na blokování metod pro tvorbu otisku prstu prohlížeče, pouze na blokování nežádoucího obsahu stránek. Jelikož se ale ve spoustě filtrovaného obsahu mohou takové nežádoucí metody nacházet, *uBlock Origin* může nezáměrně tvorbě otisku prohlížeče zabránit. Pro důkladnou ochranu je samotný *uBlock Origin* ale třeba doplnit dalšími rozšířeními, která implementují konkrétní metody pro ochranu proti tvorbě otisku prohlížeče. V tomto ohledu by bylo vhodné používat při prohlížení *uBlock Origin* společně s rozšířením *JShelter*.

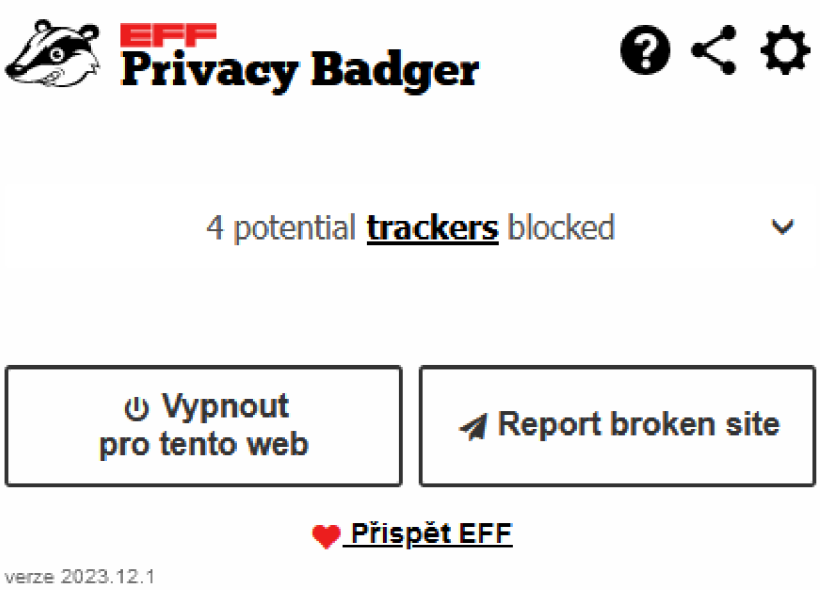


Obrázek 2.2: Grafické uživatelské rozhraní rozšíření *uBlock Origin* během návštěvy stránky *super.cz*.

Privacy Badger

Privacy Badger [17] je vyvíjeno neziskovou společností *Electronic Frontier Foundation* [16] zaměřující se na internetovou ochranu soukromí a legislativu. Rozšíření se nezaměřuje na blokování reklam, ale primárně se snaží zabránit sledovacím prvkům získat přístup k osobním údajům uživatelů. Reklamu zablokuje pouze tehdy, zjistí-li její snahu uživatele sledovat. Navíc nepoužívá k ochraně žádné seznamy blokovacích prvků a samo je schopno identifikovat sledující prvky třetích stran v průběhu sezení prohlížeče, které následně blokuje úplně. Díky periodické automatizované aktualizaci existence sledovacích prvků stránek pomocí vlastního principu *badger-sett* je rozšíření samoučící [15]. Tento mechanismus je však ve výchozím nastavení rozšíření deaktivován [7]. Obrázek 2.3 zobrazuje uživatelské rozhraní rozšíření umožňující tlačítkem rozšíření zapnout a vypnout. Rozhraní poskytuje informaci o počtu sledovacích prvků blokovaných rozšířením na navštívené stránce.

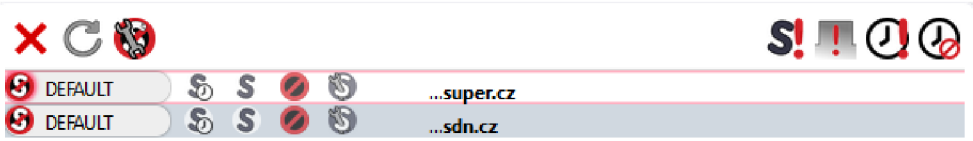
Princip rozšíření *Privacy Badger* spočívá v blokování obsahu třetích stran, jejichž záměrem je uživatele sledovat skrze internet. Blokuje i pokusy vytvořit otisk prohlížeče uživatele. Tento způsob ochrany může být kolizní s metodikou rozšíření *JShelter*, jelikož rozšíření *JShelter* v mnoha případech pro ochranu uživatele tato volání zcela neblokuje – místo toho vrací na dotaz fabrikované hodnoty. Při takovém chování by mohlo rozšíření *Privacy Badger* mít pocit, že vytvořený otisk se mění, tudíž není třeba dodávat uživateli ochranu. Používání obou rozšíření najednou tedy může mít negativní dopad na jejich funkčnost. Tento dopad však závisí na konkretní použité úrovni ochrany rozšíření *JShelter*.



Obrázek 2.3: Grafické uživatelské rozhraní rozšíření *Privacy Badger* během návštěvy stránky *super.cz*.

NoScript

Jako jedno z nejdéle vyvíjených rozšíření popsaných v této práci je *NoScript* [37] autora *Giorgio Maone* založeno na nejjednodušším poskytnutí ochrany před sledováním – preventivní blokování volání *JavaScriptových* funkcí. Rozšíření samotné je zakomponováno v *Tor browser* prohlížečích jako klíčový bezpečnostní komponent. Naprosté blokování *JavaScriptových* funkcí nedůvěryhodných domén zaručuje, že aktivní sledování uživatele doménou třetí strany pomocí *JavaScriptu* neprobíhá. Uživatelské rozhraní na obrázku 2.4 ukazuje domény, které chtějí na navštěvované stránce volat *JavaScriptové* funkce. Uživatel může u každé takové domény nastavit, jestli budou její volání povolena. *NoScript* ve výchozím nastavení blokuje všechna volání *JavaScriptových* funkcí všech domén na navštívené stránce.



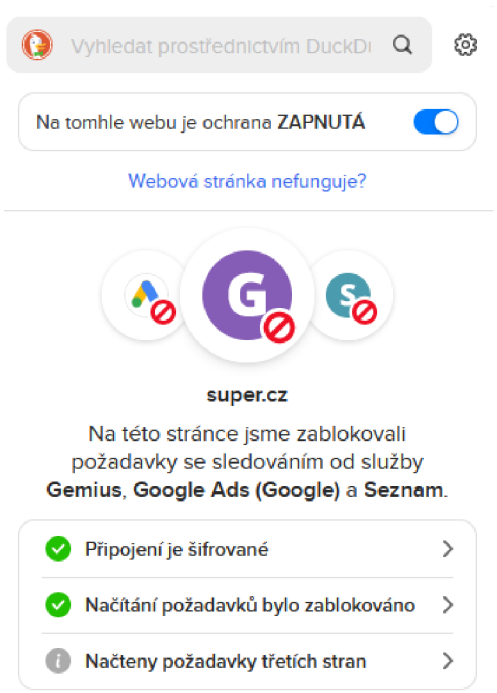
Obrázek 2.4: Grafické uživatelské rozhraní rozšíření *NoScript* během návštěvy stránky *super.cz*.

Rozšíření *NoScript* na rozdíl od rozšíření *Privacy Badger* nechává uživateli naprostou volnost co se týče obsahu, který se rozhodne zablokovat. To může být výhoda pro zkušenější uživatele internetu. Úplné zablokování ale může opět kolidovat se způsobem ochrany rozšíření *JShelter* a v mnoha případech být dokonce redundantní. *JShelter* uživateli nabízí ochranu bez jeho aktivního zapojení ihned po instalaci, rozšíření *NoScript* musí být při návštěvě nové stránky ručně konfigurováno. Rozhodnutí o blokování je tedy na uživateli a jeho schopnosti bezpečně procházet internet. Společné užívání těchto dvou rozšíření může být

pro uživatele přínosné, pokud ocení možnost blokovat konkrétní domény na stránce, kterou rozšíření *NoScript* nabízí.

DuckDuckGo Privacy Essentials

DuckDuckGo Privacy Essentials přímo navazuje na cíle a principy vyhledávače *DuckDuckGo* [44] od stejnojmenné společnosti. Rozšíření nastaví prohlížeči *DuckDuckGo* jako výchozí vyhledávač a implementuje zastavení většiny skrytých sledovacích prvků třetích stran od načtení na stránce. Blokuje tvorbu *cookies* třetích stran při návštěvě stránek a stránky nutí využívat šifrovanou verzi protokolu *HTTPS* místo protokolu *HTTP*. Zároveň nabízí ochranu před tvorbou otisku prohlížeče při prohlížení internetu a ochranu před sledovacími prvky vyskytujícími se v emailových zprávách uživatele. Uživatelské rozhraní rozšíření na obrázku 2.5 dovoluje uživateli rozšíření na stránce aktivovat, či deaktivovat. Podává stručnou informaci o doménách, které se pokusily uživateli doručit sledovací prvky s obsahem.

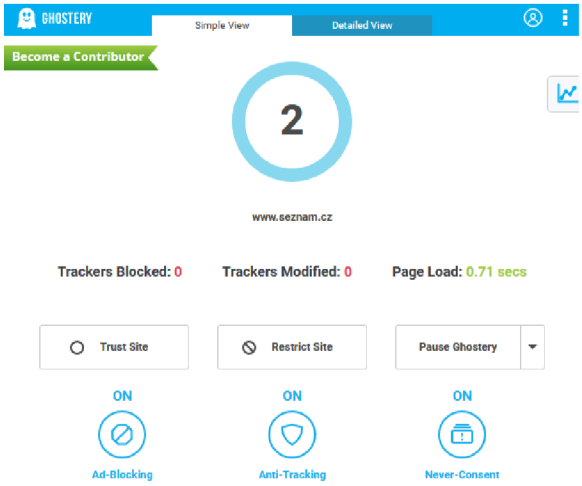


Obrázek 2.5: Grafické uživatelské rozhraní rozšíření *DuckDuckGo Privacy Essentials* během návštěvy stránky *super.cz*.

Dokumentace rozšíření neuvádí způsob, kterým *DDG PE* bojuje proti tvorbě otisku prohlížeče. Nelze tedy jednoznačně říct, zda koliduje se způsobem ochrany rozšíření *JShelter*. Na druhou stranu přináší rozšíření nový vyhledávač, který slibuje nesledovat své uživatele [14]. Díky dalším zmíněným možnostem jako vynucení protokolu *HTTPS* a ochrana proti sledovacím prvkům v elektronické poště by se mohla rozšíření *DDG PE* a *JShelter* doplňovat.

Ghostery

Ghostery od společnosti *Ghostery GmbH* se skládá z produktů se zaměřením na soukromí zvaných *Ghostery Privacy Suite* a jeho funkcionalita závisí na typu produktu, který uživatel využívá [21]. Práce se zaměřuje konkrétně na rozšíření prohlížeče s názvem *Ghostery – Privacy Ad Blocker*. Rozšíření nabízí tři hlavní druhy funkcionality – blokování nevyžádaného obsahu, blokování sledovacích prvků stránek a automatické vyjádření nesouhlasu s tvorbou *cookies* na navštěvovaných stránkách. Blokování reklam probíhá na principu černých listin (viz princip rozšíření *uBlock Origin*) z obsáhlé databáze rozšíření. Blokování sledovacích prvků však implementuje rozšíření pomocí heuristických metod v reálném čase, podobně jako rozšíření *JShelter*. Stejně tak rozšíření podvrhuje odchozí atributy uživatele, které si webová *API* pro tvorbu otisku prohlížeče vyžádají. Na obrázku 2.6 lze vidět jeho uživatelské rozhraní. Uživateli zobrazuje jednoduché statistiky, obdobně jako u předchozích rozšíření. Tlačítky může uživatel zapnout a vypnout dílčí části ochrany, kterou rozšíření poskytuje. Pod záložkou *Detailed view* nahoře zobrazuje rozšíření databázi sledovacích prvků domén připojených k navštívené stránce.



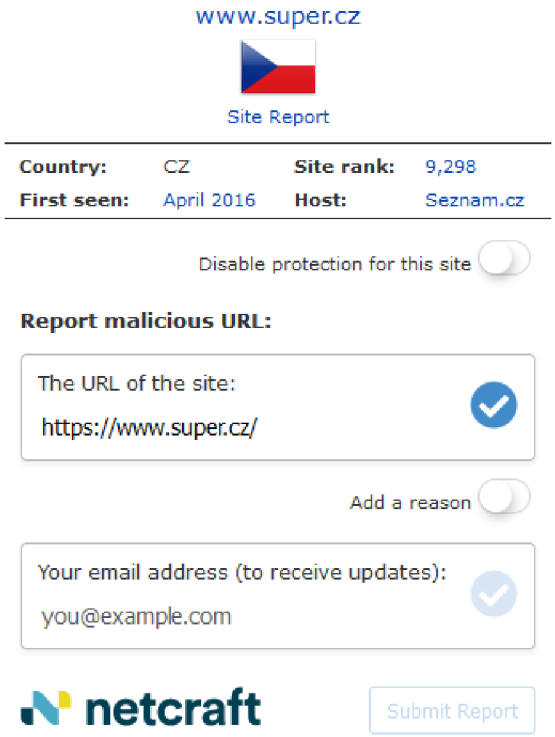
Obrázek 2.6: Grafické uživatelské rozhraní rozšíření *Ghostery* během návštěvy stránky *super.cz*.

Metoda blokování nevyžádaného obsahu může být kompatibilní s rozšířením *JShelter* a pro uživatele může být výhodné používat obě rozšíření zároveň. Metoda blokování sledovacích prvků se však podle dokumentace zdánlivě podobá metodám implementovaným v rozšíření *JShelter* a využití této funkcionality společně s rozšířením *JShelter* by mohlo nést nevyžádané výsledky pro uživatele během prohlížení za použití obou rozšíření.

Netcraft

Rozšíření *Netcraft* od stejnojmenné společnosti nabízí uživateli ochranu v reálném čase proti různým druhům kyberzločinu od *phishingu* po ochranu proti falešným internetovým obchodům a útokům typu *cross-site scripting* [39]. *Netcraft* stránky hodnotí podle vlastní interní databáze a vyhodnocuje jejich úroveň důvěryhodnosti. Pro stránky, které se v jeho databázi doposud nevyskytují, vyhodnocuje rizikový faktor na základě vlastností stránky, jako stáří registrované domény, nebo zda hostující síť byla v minulosti spojována se stránkami podvodného obsahu. Uživatel může rozšíření jednoduše ovládat skrze jeho grafické uživatelské

rozhraní znázorněné na obrázku 2.7. To poskytuje informace o navštívené stránce a nabízí uživateli možnost stránku nahlásit tvůrcům rozšíření, pokud na stránce zaznamená škodlivou aktivitu.

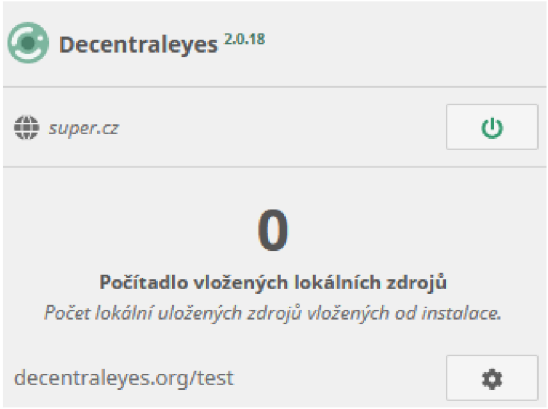


Obrázek 2.7: Grafické uživatelské rozhraní rozšíření *Netcraft* během návštěvy stránky *super.cz*.

Netcraft nebojuje při prohlížení s tvorbou otisku prohlížeče. Zabraňuje ale stránkám získávat cenné osobní údaje, jako jsou čísla kreditních karet, nebo hesla. Blokuje podezřelé *HTTP* požadavky třetích stran a zaznamenává je ve své databázi. Jelikož se rozšíření soustředí na boj proti kyberzločinu spíše než na ochranu před tvorbou otisků prohlížeče pomocí *API*, mohlo by fungovat jako dobrý doplněk pro používání společně s rozšířením *JShelter*.

Decentraleyes

Snaha *Decentraleyes* [45] spočívá v omezení obsahu zprostředkovaného třetími stranami při prohlížení internetu. Stále více obsahu je doručováno skrze požadavky na sítě pro doručování obsahu, přestože obsah může být uložen lokálně. Při získání obsahu stránek z lokálního úložiště se prohlížeč vyhne zbytečnému zasílání požadavků na sítě pro doručování obsahu a tím i potenciálnímu doručení sledovacích prvků společně s dotazovaným obsahem. Zároveň snižuje využitou šířku pásma zařízení. V dokumentaci autoři udávají jako vhodné doplnit funkcionalitu rozšíření o další populární rozšíření, jako například *uBlock Origin*. Uživatelské rozhraní rozšíření na obrázku 2.8 je velice jednoduché a poskytuje uživateli možnost rozšíření aktivovat a deaktivovat tlačítkem a přejít do globálního nastavení rozšíření. Také poskytuje uživateli informaci o počtu využití lokálních zdrojů místo dotázání se na sítě pro doručování obsahu od doby instalace.



Obrázek 2.8: Grafické uživatelské rozhraní rozšíření *Decentraleyes* během návštěvy stránky *super.cz*.

Rozšíření *Decentraleyes* se zaměřuje na velice specifickou problematiku soukromí na internetu – sítě pro doručování obsahu. Jejich požadavky filtruje a zamezuje tak přenášení sledovacích prvků při prohlížení. Tato funkcionalita by mohla rozšíření *JShelter* doplňovat v konkrétních oblastech bezpečnosti.

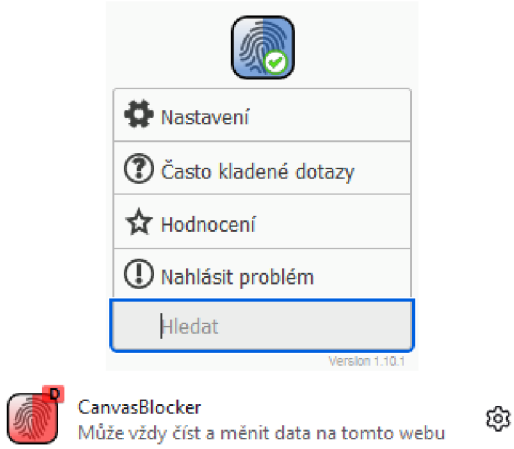
CanvasBlocker pro Mozilla Firefox

Méně známé rozšíření dostupné pouze pro prohlížeč *Mozilla Firefox* jménem *CanvasBlocker* [34] nabízí uživatelům celou řadu metod, jak zabránit navštíveným stránkám číst informace o zařízení uživatele. Modifikuje hodnoty získané voláním webových *API* a tím mění atributy otisku, jako jsou například informace o *canvas*, *WebGL* a zvukových datech zařízení. Podobně jako *JShelter* poskytuje ochranu v několika módech jako dodání falešných hodnot či úplnou blokaci výpočtu atributů. Nabízí uživateli možnost spravovat černou listinu nedůvěryhodných stránek, kterým při návštěvě informace o atributech neposkytne nikdy. Stejně tak umožňuje nastavení stránek důvěryhodných, kterým poskytne atributy při každé návštěvě. Uživatelské rozhraní zobrazené na obrázku 2.9 dovoluje uživateli přejít do nastavení rozšíření. Dále zobrazuje atributy, jejichž hodnoty byly při návštěvě stránky modifikovány. Tímto způsobem lze sledovat, jaká data uživatel stránce poskytuje.

Podle dokumentace rozšíření *CanvasBlocker* slibuje ochranu podobnou rozšíření *JShelter*, například v případě čtení hodnot *canvas*, *WebGL* a zvukového výstupu. Implementace praktické části této práce by mohla přinést zajímavé výsledky v podobě výsledných hodnot atributů, pro které se funkčnost obou rozšíření překrývá. Je nutno zmínit, že sám autor rozšíření *CanvasBlocker* nedoporučuje používat rozšíření společně s jakýmkoli dalším rozšířením, které modifikuje podobné atributy. Jinak dochází k drastickému poklesu výkonosti [34].

Canvas Blocker pro Google Chrome

Předchozí rozšíření soustředící se na úpravu grafických atributů lze využít pouze pro prohlížeč *Google Chrome*, nicméně existuje rozšíření podobného rázu od autora *joue.quroi*. Rozšíření v dokumentaci slibuje obranu proti metodám tvorby otisku, které využívají element *canvas*, konkrétně metody `toDataURL` a `dataURL` [27]. Rozšíření kromě tlačítka pro aktivaci a deaktivaci rozšíření nenabízí žádné uživatelské rozhraní.



Obrázek 2.9: Grafické uživatelské rozhraní rozšíření *CanvasBlocker* prohlížeče *Mozilla Firefox* během návštěvy stránky *super.cz*.

Podobně jako v případě předchozího rozšíření by mohlo testovací prostředí odhalit zajímavé interakce rozšíření s rozšířením *JShelter*, konkretně ve funkcionalitách, ve kterých se podle své dokumentace překrývají.

Bavíme-li se o konfliktech v rámci rozšíření, jedná se ve většině případů pouze o domněnku odvozenou z dokumentace rozšíření. Většina popsaných rozšíření je proprietární, analýza zdrojového kódu je tudíž komplikovaná a přesnější odhad možných konfliktů není možný. Je tedy žádoucí konflikty testovat přímo v praxi. Realizace běhového prostředí pro testování rozšíření ze zadání této práce má potenciál odpovědět na kladené otázky co se týče oblastí, ve kterých se rozšíření doplňují, překrývají, nebo naopak navzájem omezují svojí funkčnost.

2.4 Nástroj PETInspector

Jmenovaná rozšíření často poskytují uživateli jednoduché statistiky a výpisy, které shrnují jejich funkčnost. Díky nim lze odhadnout, jaké kroky rozšíření provádí k zajištění vyšší úrovně bezpečnosti a soukromí uživatele. Efektivita těchto kroků ale zůstává uživateli záhadou. Výpisy samotné totiž nereflektují, jakou úroveň soukromí uživateli rozšíření vlastně dodává v měřítku celého internetu.

V rámci článku [10] zkoumajícího efektivitu nejpopulárnějších bezpečnostních webových rozšíření byl vytvořen nástroj s názvem *PETInspector*. Nástroj cílí na otestování bezpečnostních rozšíření a následně na vyhodnocení úrovně jejich efektivity způsobem popsaným v článku [10]. Nástroj je implementován pomocí *Selenia* a se skládá ze tří částí [52]:

- *Fingerprinting server* – simulace reálného serveru, který díky sledovacím prvkům počítá otisk prohlížeče uživatele navštěvujícího stránku. Server implementuje techniky výpočtu otisku prohlížeče pomocí volání *JavaScriptových API* popsaných v dokumentaci nástroje [52].
- *Client simulator* – simulace uživatele navštěvujícího webové stránky, na kterých se objevují sledovací prvky. Simulaci lze provést za použití konkrétních bezpečnostních rozšíření nainstalovaných v uživatelově simulovaném prohlížeči.

- *Analysis engine* – analyzuje vygenerované otisky prohlížeče získané při simulaci uživatelova procházení a vyhodnocuje efektivnost nainstalovaných rozšíření na základě metrik z článku [10].

Pro tuto práci jsou zajímavé první dvě části nástroje. Simulace serveru, který počítá otisk prohlížeče uživatelů při použití jednotlivých rozšíření, poskytuje náhled na skutečné výsledky bezpečnostních rozšíření a dodává možnost otisky prohlížeče mezi sebou porovnat při návštěvách za použití bezpečnostních rozšíření. Generování takového otisku za použití rozšíření je i bez sofistikované analýzy poslední části nástroje dobrým indikátorem metodiky rozšíření. Samotná analýza efektivity však není součástí práce ani jejího zadání, cílem je uživateli dodat nástroj, který mu umožní vlastní analýzu na základě sesbíraných dat. Část simulující prohlížení uživatele poskytuje automatizované procházení stránek, na kterých se mohou objevovat sledující prvky. V rámci prostředí navrženého pro porovnání bezpečnostních rozšíření je automatizované procházení, žádoucí (viz sekce 3.1).

Kapitola 3

Požadavky na navrhovaný systém

Tato kapitola stanoví cíl praktické části práce a vytkne momentální nedostatky stávajících testů projektu *JShelter* popsaných v sekci 2.2, společně s doporučením, jak testy vylepšit v závislosti na dnešní dostupné technologii.

Dále uvede čtenáře do problematiky automatizovaného procházení webových stránek pomocí dostupných nástrojů a testování dostupných rozšíření prohlížeče se zaměřením na bezpečnost. Zmíní dostupné technologie a uvede jejich klady a zápory pro implementaci praktické části práce.

3.1 Cíl praktické části práce

Vycházíme-li z předchozího textu práce, můžeme s jistotou říci, že je pro dosažení určité úrovně soukromí a bezpečí při prohlížení internetu potřebné se chránit využitím dalších ochranných mechanismů, které široce dostupné prohlížeče nemusí nabízet. Mechanismy mohou existovat právě ve formě rozšíření prohlížečů. Tato práce zmiňuje pouze malý počet rozšíření, na trhu jich však existuje velké množství. Lze předpokládat, že mnoho bezpečnostních rozšíření vznikne i v budoucnu. Uživatel má tedy na výběr z mnoha rozšíření a jejich kombinací.

Rozšíření nejsou ve většině případů vyvíjena tak, aby se ve funkcionalitě různá rozšíření doplňovala. Kvůli implementaci ochranných mechanismů mohou ale u kombinací rozšíření nastat různé stavy – teoreticky lze najít kombinaci rozšíření, která zvýší úroveň ochrany uživatele, stejně tak lze najít kombinaci, která úroveň ochrany uživatele nezáměrně sníží. Stejně tak uživatel nechce zároveň využívat rozšíření, jejichž funkcionalita je redundantní a jejichž využití pouze zhorší výkonnost prohlížeče bez pozitivního efektu na uživatelovo soukromí.

Je tedy žádoucí, aby existoval nástroj, který dovolí testování rozšíření, nebo jejich kombinací, a poskytl uživateli zpětnou vazbu týkající se výsledku metodik ochrany uživatele. Nástroj by měl sbírat data, která po dalším přezkoumání dovolí uživateli odhadnout správnou kombinaci bezpečnostních rozšíření a odhalit kombinace nežádoucí.

Vytvoření nástroje pro porovnání rozšíření

Cílem této práce je vytvořit vhodné běhové prostředí, které přebírá metodiku prvních dvou částí nástroje *PETInspector* popsaného v sekci 2.4 a co nejvhodnějším způsobem ji kombinuje s metodikou stávajících testů projektu *JShelter* popsaných v sekci 2.2.

Ve své podstatě jsou integrační testy projektu *JShelter* nadstavbou nad výsledkem prvních dvou částí projektu *PETInspector* – testy totiž nejen vypisují reálné hodnoty vypočítané stránkou tvořící otisk prohlížeče uživatele, ale také je porovnávají s očekávanými hodnotami, které by rozšíření *JShelter* mělo na volání *API* vracet.

Jedním z výsledků této práce by tedy měl být nástroj, který poskytne hodnoty získané stránkou obsahující sledovací prvky pro výpočet otisku prohlížeče uživatele. Nástroj by měl kombinovat metody tvoření otisku prohlížeče obou implementací. Stejně jako v případě rozšíření *JShelter* by měl dodat i pro ostatní rozšíření a jejich kombinace informaci o tom, jestli se hodnoty shodují s hodnotami očekávanými. U ostatních rozšíření bohužel nelze detailně porovnat výsledné a očekávané hodnoty, protože zdrojový kód rozšíření není veřejně k dispozici. Nicméně by nástroj měl vývojářům daných rozšíření umožnit definovat očekávané hodnoty i pro jiná rozšíření, než je *JShelter*.

Nástroj by měl být konzistentní, deterministický a předvídatelný, co se týče výstupu simulace prohlížení stránky, která počítá otisk prohlížeče. To souvisí s pevnou kontrolou verzí rozšíření, ovladačů prohlížečů a operačních systémů. Takového chování lze dosáhnout vytvořením vlastního lokálního serveru, na kterém běží stránka navštěvovaná při simulaci procházení, který je zcela nezávislý na externích faktorech a výpočet otisku pomocí volání webových *API* je pod plnou kontrolou běhového prostředí, podobně jako v případě *fingerpringing server* projektu *PETInspector* ze sekce 2.4. Testy je tedy žádoucí spouštět v prostředí kontrolovaném programátorem, které je co nejméně závislé na externích faktorech (více v sekci 3.3).

Dalším cílem implementace je převést princip systémových testů projektu *JShleter* do nově vzniklého běhového prostředí tak, aby jím bylo možné testovat i ostatní rozšíření. V tomto případě nebude probíhat návštěva pouze jedné dedikované stránky, která počítá otisk prohlížeče. Stejně jako v případě systémových testů projektu bude v rámci testování navštíveno velké množství různých existujících stránek, které nemají žádné spojení s touto prací.

Běhové prostředí by mělo být schopné otestovat jak samotná rozšíření, tak i jejich libovolné kombinace. Dále by mělo být lehce udržovatelné. V průběhu času se kvůli problému různých verzí ovladačů prohlížečů a rozšíření očekává zásah programátora do vnitřní implementace, prostředí by tedy mělo být co nejlépe oddělené na samostatné funkční prvky.

Vylepšení stávajících testů projektu *JShelter*

Dalším cílem práce je vylepšit stávající testy projektu *JShelter* (viz sekce 2.2). Průběhem času se kvůli vývoji technologií, aktualizacím prohlížečů a *JavaScriptových* API mohou vyskytnout problémy týkající se vnitřní implementace dílčích částí testů. Takové problémy lze vyřešit pouze jejich pravidelným vývojem a údržbou, nikoliv jednou aktualizací odolnou vůči času. Tato práce se na ně tedy nesoustředí.

Další problém stávajících testů vzniká v jádru implementace integračních a systémových testů. Jejich implementace vznikla v rámci diplomové práce pana Bednáře [8] v roce 2020, kdy rámec *Selenium* ještě neobsahoval nástroj *Selenium Manager* pro spravování verzí ovladačů prohlížeče, které jsou pro spuštění testů stěžejní. V jejich dosavadní formě je tedy pro spuštění testů nutné po každé aktualizaci ovladačů prohlížeče související s aktualizacemi prohlížečů samotných ručně z oficiálních zdrojů získat nový aktuální ovladač a vložit jej do zdrojových adresářů testů. Nové verze ovladačů vývojáři dodávají v časovém rozmezí týdnů až měsíců, manuální aktualizace ovladačů je tedy pracná a náchylná k chybám. Díky nástroji *Selenium Manager* je nyní možné spravovat verze ovladačů prohlížečů a využívat

nejen jejich nejvíce aktuální verzi. Pokud by byly stávající testy upraveny na úrovni jejich zdrojového kódu tak, aby nástroj *Selenium Manager* využívaly, stalo by se spouštění testů a jejich údržba příjemnějším.

3.2 Automatizace procházení webových stránek

Pro zhodnocení funkčnosti rozšíření lze aplikovat dva základní postupy – zhodnocení neautomatizované, tedy ruční procházení webových stránek a posuzování chování prohlížeče za použití rozšíření a bez použití rozšíření, nebo prohlížení automatizované. Prohlížení automatizované spočívá ve využití stávajících technologií k simulaci uživatelova pohybu po internetu. Díky automatizaci lze pokrýt v případě systémových testů rozšíření několikanásobně větší počet stránek, než při prohlížení ručním. Dále stávající nástroje v mnoha případech poskytují možnost uchovávat informace a statistiky o běhu prohlížení. V případě zkoumání metodik rozšíření v ochraně proti tvorbě otisku prohlížeče dovoluje automatizace prohlížení jednoduše ukládat relevantní informace ve formě databází, které lze následně analyzovat pomocí dalších nástrojů.

Práce se soustředí na vytvoření běhového prostředí, které využívá automatizované procházení stránky tvořící otisk prohlížeče uživatele, nebo velkého množství webových stránek pomocí již existujícího rámce. Pro účely práce je mnohem efektivnější získávat velké objemy dat generované automatizovaným prohlížením, které lze zpětně analyzovat pro zhodnocení aplikace metod rozšíření a jejich kombinací. Navíc zhodnocení rozšíření jako takové není součástí zadání práce a nedává tedy smysl vytvářet běhové prostředí, které neposkytuje uživateli žádnou úroveň automatizace.

Dostupné nástroje pro automatizaci prohlížení

Pro účely práce lze využít již existující rámce pro automatizaci prohlížení stránek a sběr informací o průchodech za použití bezpečnostních rozšíření, i bez nich. Při výběru rámce je důležité se soustředit na podporovanou platformu, rozšíření, která je možné rámcem automatizovat, a kompatibilitu s dosavadními testy projektu *JShelter*.

Knihovna *Puppeteer* umožňuje kontrolu nad automatizovaným procházením webových stránek využitím *DevTools* protokolu [22]. Dovoluje vytvářet snímky obrazovky při procházení stránek, automatizaci uživatelské interakce, jako je například vyplňování formulářů, používání uživatelského rozhraní a vstup z klávesnice a vytvářet prostředí pro testování nejnovější verze *Chrome* a *JavaScriptu*. Pomocí knihovny lze testovat nainstalovaná rozšíření i v různých kombinacích.

I když je svojí funkcionalitou knihovna pro potřeby práce vyhovující, nepodporuje procházení pomocí jiných prohlížečů, než *Google Chrome* a *Chromium*. Tento nedostatek eliminuje velké množství testovaných rozšíření a částečně znemožňuje pro implementaci zahrnout stávající testy projektu *JShelter*. Její využití tedy v rámci práce nedává smysl.

Nástroj *OpenWPM* automatizací sbírá data o prohlížení webových stránek za pomocí využití rámce *Selenium* [4]. Jedná se o jeho nadstavbu za účelem uchování informací o průchodech, které se týkají přímo bezpečnosti a soukromí uživatele, jako jsou například zaslané *HTTP* dotazy, obdržené *HTTP* odpovědi, dokumentace přesměrování uživatele stránkou při prohlížení a volání *JavaScriptových* funkcí [5]. Vytváří instance prohlížečů a průchody (*crawls*) řídí skrze nastavitelný modul *Task Manager*. Pro průchod dovoluje vytvářet uživatelský profil na míru, včetně instalace konkrétních rozšíření prohlížeče. Výsledky průchodů ukládá do databází ve formátu *SQLite*, tudíž zjednodušuje jejich analýzu.

Principy nástroje a jeho výsledky jsou pro tuto práci velice vhodné. Problémem je fakt, že *OpenWPM* momentálně nepodporuje automatizaci jiného prohlížeče, než je *Mozilla Firefox*. Nástroj pro potřeby práce tedy z podobných důvodů, jako v případě knihovny *Puppeteer*, nelze využít. Jeho návrh a metodiky jsou ale silnou inspirací pro implementaci praktické části práce.

Jedním z nejznámějších a nejpoužívanějších nástrojů pro automatizaci prohlížení je nástroj *Selenium* [49]. *Selenium* dovoluje automatizovat velké množství úkolů spojených s prohlížeči, jako je automatizované procházení stránek, testování stránek pomocí skriptů a paralelní testování na více zařízeních. Pro účely práce je nejzajímavějším nástrojem *Selenium WebDriver*, který umožňuje automatizovat prohlížení, včetně úpravy uživatelského profilu a využití rozšíření prohlížeče. *WebDriver* podporuje velké množství prohlížečů, včetně prohlížečů *Google Chrome*, *Mozilla Firefox* a *Microsoft Edge*. Nástroj je do velké míry konfigurovatelný a vyhovuje požadavkům praktické části práce. Stěžejní je pro práci nově implementovaný *Selenium Manager* [50], který značně zjednodušuje problém instalace správné verze webových ovladačů (viz sekce 3.1). Díky němu je možné automaticky poskytnout nutný ovladač prohlížeče, včetně možnosti volby jeho verze. Nabízí rozmanité množství konfigurací, které lze využít při implementaci automatizace prohlížení běhového prostředí.

Díky možnosti automatizace více druhů prohlížečů je nástroj *Selenium* ideální pro potřeby implementační části práce. *Selenium Manager* má potenciál eliminovat nedostatky stávajících testů projektu *JShelter* a značně zjednodušit použití nástroje pro uživatele. Navíc jsou stávající testy projektu implementovány právě v tomto nástroji, což zjednoduší jejich úpravu.

3.3 Realizace běhového prostředí

Implementace běhového prostředí splňující cíle popsané v sekci 3.1 jistě bude obsahovat mnoho závislostí spojených s využitými nástroji, zároveň musí být možné v prostředí simulovat průchod uživatele internetem. Odpovědí na tyto požadavky může být virtualizace – prostředí bude abstraktní vrstvou spuštěnou nad hostitelským zařízením, která bude obsahovat vlastní závislosti potřebné pro realizaci testování [25]. Alternativně lze využít odlehčených kontejnerů, které nevytváří abstraktní vrstvu pomocí emulace fyzického hardware, nýbrž spojují knihovny, zdrojový kód, rámce a obecně všechny závislosti potřebné pro spuštění běhového prostředí sdílením jádra operačního systému [24].

Možnosti virtualizace

Princip virtualizace spočívá ve vytvoření abstraktní vrstvy nad hostitelským zařízením, která může existovat například ve formě sítě, hardwarových prvků, nebo aplikace. Virtuální stroje používají *hypervizor* k emulaci fyzického hardware a oddělují tak virtuální instance od operačního systému. Výsledné virtuální stroje jsou izolované a přenositelné.

Pro virtualizaci a tvorbu virtuálních strojů lze zvolit software *VirtualBox* společnosti *Oracle* [41]. Dostupný je pro operační systémy *Microsoft Windows*, *Linux* a *macOS*. Umožňuje simulaci operačních systémů typu *Windows*, *Linux*, *BSD*, *Solaris* a dalších. Nabízí příjemné uživatelské prostředí a vysokou konfiguraci virtuálních strojů, včetně přidělených prostředků hostitelských strojů.

Možnosti kontejnerizace

Kontejnery nabízí rychlejší a velikostně menší řešení pro nasazení běhového prostředí. Oproti virtualizaci pomocí virtuálních strojů využívají kontejnery místo fyzického hardware jádro operačního sytému a sdílené knihovny. Díky tomu kontejnerizace nabízí vysokou úroveň abstrakce a přenositelnosti na jiná zařízení nezávisle na jejich platformě. Zároveň vyžadují méně zdrojů pro spuštění.

Docker [12] je software poskytující jednotné prostředí pro odlehčenou virtualizaci. Dostupný je pro nejpopulárnější operační systémy *Microsoft Windows*, *Linux* i *macOS*. Umožňuje tvorbu kontejnerů pomocí *Dockerfile* [11], souboru obsahujícího jednoduché příkazy reprezentující přepis pro sestavení nového kontejneru. Kontejnery lze využít k rychlému nasazení aplikací díky možnosti předinstalovat veškeré potřebné závislosti v rámci *Dockerfile*. Takové kontejnery jsou vhodné pro realizaci testování prvků aplikací nezávisle na platformě. *Docker* dovoluje kontejnerům sdílet složky s hostitelským zařízením a zapisovat výstupní data testů do souborů, které přetrvají i po zániku kontejneru.

Kapitola 4

Návrh

Tato kapitola přednese návrh realizace běhového prostředí pro testování webových rozšíření se zaměřením na bezpečnost a soukromí uživatele. V závislosti na textu předchozí kapitoly 3 vybere konkrétní technologie vhodné pro implementaci praktické části práce. Dále uvede obecný návrh výstupní realizace včetně popisu architektury výsledného běhového prostředí.

4.1 Vybrané technologie

Podle sekce 3.2 je důležité pro realizaci testování vybrat vhodný nástroj pro automatické procházení webových stránek. V případě procházení za účelem provedení systémových testů je vhodné, aby bylo doplněno o sbírání relevantních dat o událostech, které mohou ovlivnit uživatelovo soukromí a bezpečí. Z textu sekce vyplývá, že nejlepším rámcem pro účely práce je díky podpoře mnoha prohlížečů rámec *Selenium*. Také se jedná o stejný rámec, který byl využit při implementaci projektu *PETInspector*. Díky tomu bude snazší převést části stávající implementace projektu do nové formy méně závislé na verzích ovladačů prohlížečů díky nástroji *Selenium Manager*. Výsledné běhové prostředí by mělo přebírat ideologii a metodiku projektu *PETInspector* a *JShelter* a vytvořit novější nástroj pro testování bezpečnostních webových rozšíření jednodušší na údržbu. Implementace ji bude přebírat zejména ze tří stávajících zdrojů:

- Stávající integrační testy projektu *JShelter* – integrační testy projektu *JShelter* provádí automatické navštěvování testovací stránky pro výpočet otisku uživatelova prohlížeče. Výsledné běhové prostředí bude kombinovat metody výpočtu otisku prohlížeče, integračních testů i projektu *PETInspector*. Pro rozšíření a jejich kombinace bude probíhat i následná analýza výsledků průchodu vůči očekávaným hodnotám.
- Stávající systémové testy projektu *JShelter* – systémové testy provádí automatizovaný průchod velkého počtu webových stránek za účelem vyhodnocení dopadu používání rozšíření na funkčnost stránek. Výsledné běhové prostředí přebere tento koncept a umožní systémové testy provádět i pro ostatní rozšíření a jejich kombinace.
- Nástroj *PETInspector* – běhové prostředí přebere metodiku tvorby otisku prohlížeče (viz výše) a implementaci serveru pro výpočet otisku prohlížeče, který běží lokálně a tudíž není závislý na externích doménách. Dále se inspiruje způsobem, kterým jsou před průchodem rozšíření instalována, a ukládáním dat v rámci části *client simulator*.

V otázce virtualizace sekce 3.3 bude běhové prostředí využívat kontejnery nástroje *Docker*. Kontejnery jsou oproti virtuálním strojům nástroje *VirtualBox* odlehčenější a díky

Dockerfile jednodušší na režii při automatickém testování výsledného běhového prostředí. Výsledný kontejner je menší a tedy přenositelnější, než virtuální stroj.

Vyjmenované technologie bude třeba spojit a vytvořit tak běhové prostředí pro testování bezpečnostních rozšíření, které vyhovuje zadání této práce. To zahrnuje vytvoření kontejneru pro testování, instalaci potřebných závislostí a vytvoření konfiguračních souborů pro uživatelský vstup. V rámci vnitřní implementace kontejneru je třeba ze zdrojového kódu projektu *PETInspector* vyjmout část simulující server pro tvorbu otisku prohlížeče a tu vhodně adaptovat pro běhové prostředí změnou a aktualizací zdrojového kódu. To samé je třeba udělat i pro automatický průchod pomocí simulace klienta. Metody tvorby otisku prohlížeče projektu *PETInspector* a *JShelter* je třeba vhodně zkombinovat.

4.2 Návrh implementace běhového prostředí

Prostředí bude implementováno a spouštěno v rámci kontejneru *Docker*. Použitím *Dockerfile* bude nejprve ustanoven přepis pro sestavení obrazu, to zahrnuje instalaci nutných závislostí dle již zmíněných vybraných technologií – *Selenium*, zejména nástroj *Selenium Manager*, podpůrné knihovny pro správný chod prohlížečů a další potřebné závislosti, které vzejdou postupně podle potřeb implementace. Uživateli bude umožněno vybrat si sestavením obrazu ze dvou módů:

- Mód testování rozšíření pomocí serveru pro výpočet otisku prohlížeče – podobně jako v případě projektu *PETInspector* a integračních testů projektu *JShelter* budou spuštěny testy, jejichž výsledek demonstruje efektivnost vybraných rozšíření nebo jejich kombinací v modifikaci získaných atributů často používaných při tvorbě unikátního otisku prohlížeče uživatele. Pomocí výstupních souborů serveru bude provedena analýza výsledných hodnot oproti očekávaným hodnotám při použití rozšíření nebo jejich kombinací.
- Mód testování průchodem známých domén – vybraná rozšíření nebo jejich kombinace bude otestována pomocí automatizovaného průchodu stránkami, podobně jako v případě systémových testů rozšíření *JShelter*. Seznam stránek bude uveden pomocí žebříčku *Tranco top sites*[1].

Architektura běhového prostředí

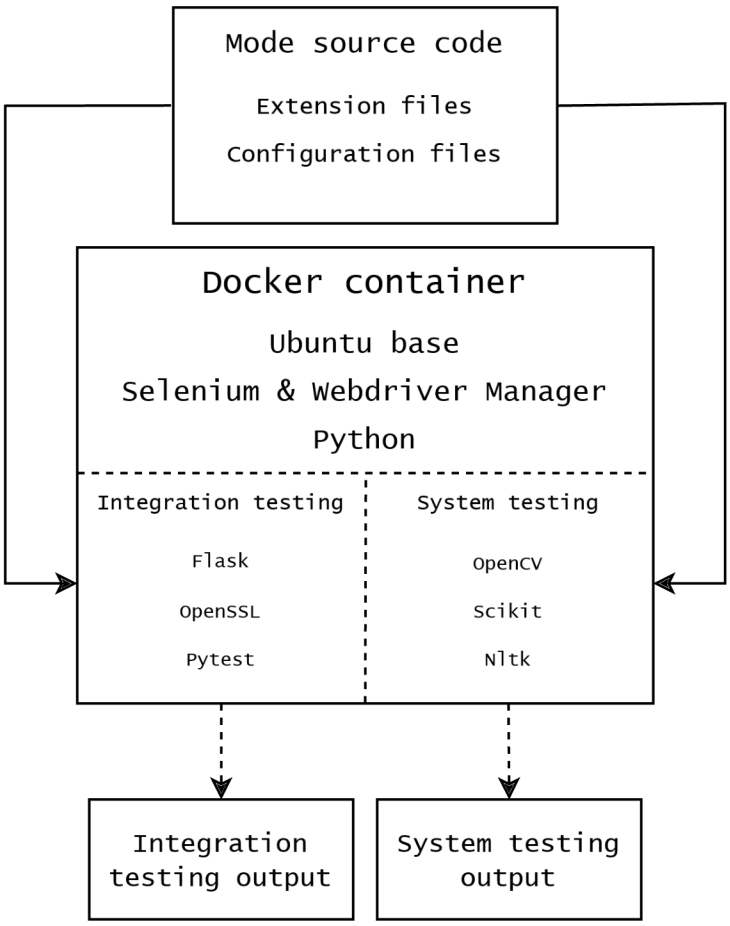
Běhové prostředí se bude skládat ze tří základních komponentů. Ústředním bodem bude *Docker* kontejner, jenž bude spuštěn ze sestaveného *Docker* obrazu podle použitého módu. Oba sestavené obrazy sdílí společné závislosti, ale v mnoha závislostech se liší. Pro jejich sestavení bude možné použít stejný soubor *Dockerfile*.

Podle vybraného módu vstoupí do kontejneru zdrojové soubory prostředí, které obsahují celkovou implementaci testovacího módu společně s konfiguračními soubory testů. Zdrojový kód také bude obsahovat soubory testovaných rozšíření ve formátu *XPI* v případě prohlížeče *Mozilla Firefox* a *CRX* pro prohlížeč *Google Chrome*.

Poslední částí budou výstupní soubory dílčích testovacích módů. Při návrhu je důležité zajistit persistenci výstupních souborů i po odstranění kontejneru a obrazu *Docker*.

Obrázek 4.1 zachycuje architekturu výsledného běhového prostředí. Cílem návrhu je docílit co nejmenšího výsledného obrazu, díky kterému bude možno běhová prostředí spustit.

[1] https://tranco-list.eu/



Obrázek 4.1: Architektura běhového prostředí.

Kapitola 5

Implementace běhového prostředí

Tato kapitola popisuje vnitřní implementaci běhového prostředí včetně všech jeho dílčích částí. Postupně se soustředí na komponenty v pořadí, ve kterém je prostředí sestaveno, konfigurováno a spuštěno.

Prostředí je rozděleno na dvě části – část integračního testování a systémového testování. Obě části jsou implementovány ve formě kontejneru programu *Docker* a jsou sestaveny společným souborem *Dockerfile*. Rozhodnutí poskytnout pro oba módy testování odlišné kontejnery vyplynulo z konzultace s vedoucím práce v návaznosti na typický způsob použití uživatelů. Překryv technologií není dostatečně velký, aby se vyplatilo sestavit jeden obraz zahrnující oba módy.

Sestavení využívá technologie poskytnuté programem *Docker* s názvem *multi-stage builds* [13]. Sestavení obrazu se skládá ze tří částí:

- Sestavení `base` vrstvy obrazu, která je společná pro oba výsledné kontejnery. Čerpá z *ubuntu:latest* a dodává nejzákladnější knihovny potřebné pro chod operačního systému. Dále dodává moduly *Selenium*, *Selenium Webdriver*, *numpy* a další.
- Vrstva `integration` poskytuje moduly potřebné pro provedení integračních testů. Její sestavení probíhá pomocí příkazu `-target` a přímo navazuje na vrstvu `base`.
- Podobně jako předchozí vrstva poskytuje vrstva `system` moduly pro systémové testování.

Vrstvy `integration` a `system` obě čerpají z vrstvy `base`, tedy pokud se vrstva `base` nachází v *cache* sestavení, ušetří se značné množství času a internetového pásma.

Kvůli rozdílnému přístupu operačních systémů *Linux* a *Windows* k proměnné prostředí, díky které lze provést testování pomocí grafického rozhraní prohlížečů, je nutné prostředí kontejneru spouštět různými příkazy. V případě operačního systému *Linux* bylo v době implementace nutné použít k sestavení obrazu a spuštění kontejneru *Docker Engine*. Pro *Windows* bylo nutností použít *Docker Desktop*, který *Docker Engine* zahrnuje. Kompatibilita prostředí pro zobrazení verze *Docker Desktop* pro operační systém *Linux* může být v budoucnu přidána a způsob implementace může vývoj reflektovat.

5.1 Integrační testování

Prostředí je realizováno ve formě kontejneru aplikace *Docker*. Pro sestavení kontejneru a instalaci potřebných závislostí lze využít souboru *Dockerfile*.

Prostředí je založeno na bázi operačního systému *Ubuntu*. V průběhu sestavení jsou staženy veškeré závislosti potřebné ke spuštění testovacího prostředí. Kromě mnoha standardně používaných *Linuxových* knihoven a knihoven potřebných pro využití prohlížečů *Google Chrome* a *Mozilla Firefox* lze zmínit například:

- *Flask* – Webová aplikace simulující server sbírající atributy otisku prohlížeče běží ve vývojovém prostředí *Flask*.
- *OpenSSL* – Knihovna potřebná k ustanovení *HTTPS* spojení mezi serverem a klientem. Atributy jako určování geografické pozice jsou v prohlížečích dostupné pouze pro návštěvy skrze bezpečnou komunikaci.
- *Selenium* a *Webdriver Manager* – Balíčky pro řízení a správu automatického procházení webových stránek.
- *Pytest* – Vykonává kontrolu samotných výsledků integračních testů.
- Knihovny *Regex* a *Numpy* sloužící k podpůrným operacím kontroly výsledků.

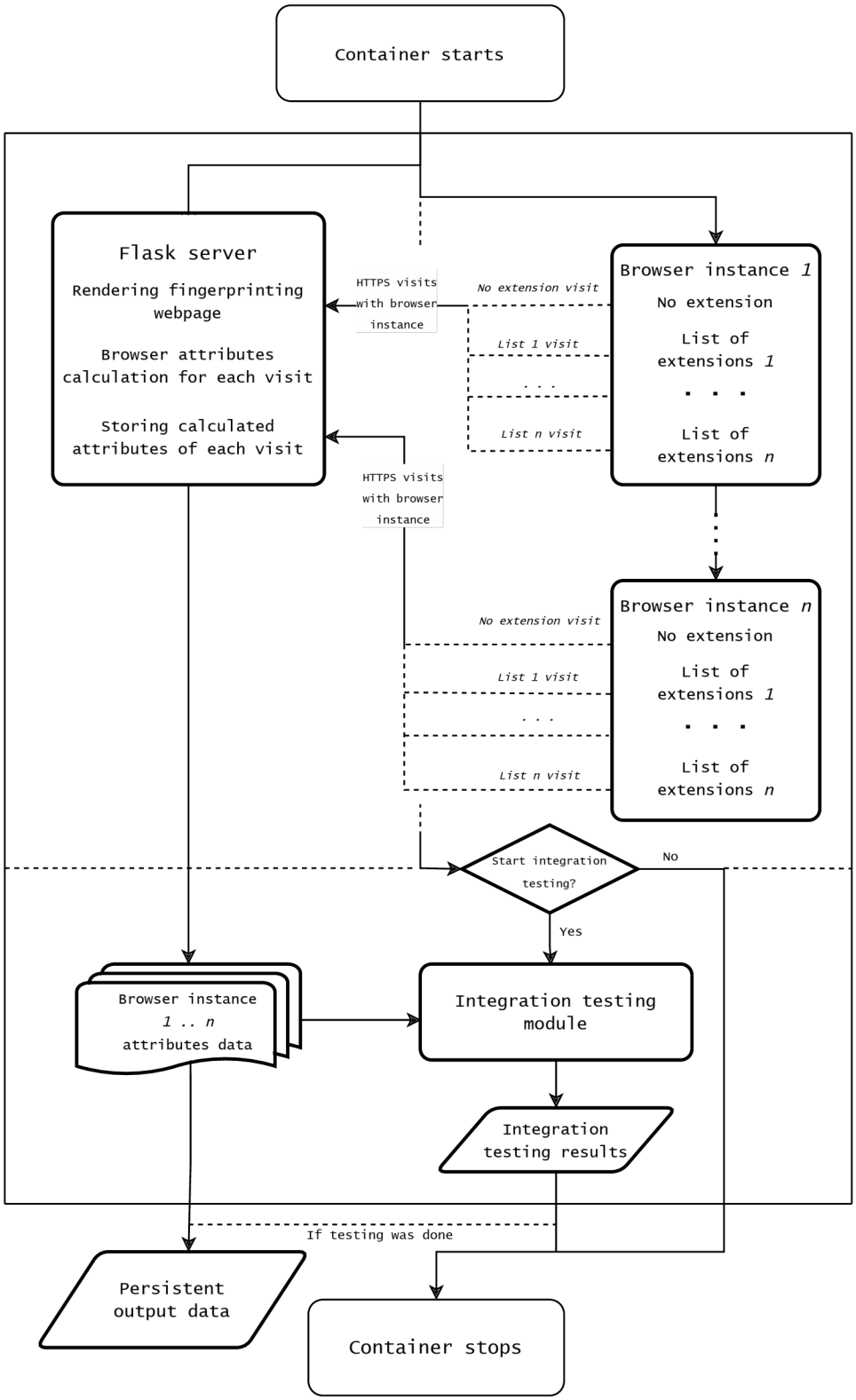
5.1.1 Úloha serveru, klienta a testovacího modulu

V rámci integračních testů běhového prostředí figurují tři hlavní dílčí části. Část serveru a klienta principiálně přebírá metodiku projektu *PETInspector* popsanou v 2.4. Testovací modul pracuje na obdobném principu, jako dosavadní integrační testy projektu *JShelter*. Komunikaci dále popsaných částí běhového prostředí reprezentuje diagram 5.1.

Server má na starosti zprostředkovat webovou stránku, kterou je možno navštívit. Na navštěvované stránce se objevuje značné množství technik pro tvorbu otisku prohlížeče jak statickými, tak dynamickými metodami. Dosavadní techniky je možno vyřadit, stejně tak je možné do seznamu měřených atributů přidat atributy nové. Dále se stará o správné ukládání dat o naměřených atributech, včetně jejich kategorizace podle dne návštěvy stránky, zvolených prohlížečů a jejich verzí. V rámci kategorií také označuje data podle rozšíření, která byla v průběhu návštěvy nainstalována. Výsledkem práce serveru jsou soubory ve formátů *JSON*, které lze dále zpracovat, či díky nim vyhodnotit chování rozšíření. Server může běžet zcela nezávisle na zbylých dvou částech. Server neslouží jako naprosto věrná reprezentace výpočtu otisku prohlížeče ve skutečném prostředí internetu. Proces výpočtu atributů otisku prohlížeče a jeho generování je velmi komplikovaná problematika. Výpočet atributů v rámci testovacího prostředí slouží k ověření funkčnosti rozšíření ve smyslu vypočtených hodnot.

Klient úzce spolupracuje se serverem a provádí návštěvu webové stránky pro tvorbu otisku prohlížeče. Rolí klienta je také poskytnout příslušné ovladače prohlížečů potřebných pro prohlížení, konfigurace těchto ovladačů a tím i prohlížečů samotných. Průchod je plně automatizován, včetně nastavení nainstalovaných rozšíření. To zahrnuje i možnost testovat uživatelské interakce s grafickým rozhraním rozšíření. Klient může fungovat samostatně, ale jeho akce negenerují žádný výsledek v podobě výstupních souborů. Tato zodpovědnost je pouze na serveru.

Testovací modul má za úkol vyhodnotit očekávané atributy otisku prohlížeče oproti naměřeným hodnotám v průběhu použití rozšíření. Modul nemá za úkol vyhodnotit efektivitu rozšíření co se týče boje za internetové soukromí a bezpečí. Integrační testování pouze dodává testujícím pohled na funkcionalitu testovaných rozšíření a jejich kombinací při použití různých verzí prohlížečů. Předpokládá určitou míru interaktivity ze strany testujícího.



Obrázek 5.1: Diagram znázorňující komunikaci dílčích částí běhového prostředí.

V principu testovací modul vždy očekává existenci alespoň dvou různých výstupů v rámci unikátní kombinace prohlížeče a jeho verze. První výstup obsahuje atributy otisku prohlížeče naměřené navštívenou stránkou pomocí prohlížeče, ve kterém nebyla nainstalována jakákoli rozšíření. Další výstup a následně libovolné množství výstupů poté obsahuje atributy otisku prohlížeče vypočtené navštívenou stránkou za použití rozšíření, nebo různých kombinací rozšíření. Tyto dva výstupy vnímá jako dvě různé instance dat, jež je potřeba proti sobě porovnat. Do toho vstupují hodnoty atributů, které testující rozšíření očekává při návštěvě stránky za použití rozšíření, nebo jejich kombinací. V mnoha případech očekávaná hodnota atributu za použití rozšíření přímo závisí na hodnotách naměřených při návštěvě bez nich. Očekávané hodnoty nelze univerzálně přesně určit pro každé rozšíření a jejich různé kombinace. Je tedy na testujícím rozhodnout, jaké hodnoty atribut vlastně očekává.

5.1.2 Konfigurační soubory a nastavení testování

Před spuštěním očekává prostředí konfiguraci průběhu testování. Jedná se o dva konfigurační soubory ve formátu *JSON*. Konfiguraci lze provést pro klienta a pro běh.

Konfigurace běhu

Konfigurace běhu udává, jaké stránky budou v průběhu testu navštíveny a zda implementovat nastavení *proxy*.

- `fp_sites` dovoluje při konfiguraci běhu ovlivnit, jaké stránky budou klientem navštíveny. V případě testovacího prostředí se předpokládá, že server běží na adrese a portu výchozí pro *Flask* server – `https://127.0.0.1:5000`. Server běží nad protokolem *HTTPS*, jelikož využití mnoha *API* prohlížečů je podmíněno šifrovaným provozem. V průběhu testování lze navštívit i jiné stránky přidáním jejich adres do seznamu `fp_sites`. To v případě, pokud je vhodné nasadit *Flask* server na jinou adresu nebo port. Lze provést návštěvu i internetových stránek, ty ale samozřejmě nebudou generovat žádné soubory s naměřenými atributy.
- `socks5_proxy` udává, zda při návštěvách využít *SOCKS5 proxy* [26]. Nastavení proxy probíhá uvnitř prohlížeče.
- `proxy_setting` potom specifikuje nastavení, které bude využito při konfiguraci.

Jako ukázku uvažujme konfiguraci běhu zobrazenou ve výpisu 5.1 ve formátu *JSON*:

```
{
    "fp_sites": ["https://127.0.0.1:5000"],
    "socks5_proxy": true,
    "proxy_setting": { "address": "127.0.0.1",
                       "port": 8000,
                       "bypass-list": "localhost, 127.0.0.1"
                        }
}
```

Výpis 5.1: Příklad konfiguračního souboru pro běh.

Při běhu bude navštívena stránka `https://127.0.0.1:5000`, na jejíž IP adrese při výchozím nastavení běží *Flask* server pro výpočet atributů prohlížeče.

Běh využije nastavení *SOCKS5 proxy*. Pro nastavení využívá možnosti prohlížečů *Google Chrome* a *Mozilla Firefox*.

Konfigurace klienta

Vybraný konfigurační soubor pro testování musí být ve formátu *JSON* a obsahovat tři základní klíče společně s jejich hodnotami. Tato klíčová slova jsou:

- `experiment_name` sloužící k odlišení spuštěných běhů za účelem budoucího zpracování dat. Jméno experimentu nemá – kromě odlišení na testování – vliv, nicméně by mohlo být přínosné pro testujícího v případě zájmu o další analýzu dat.

- `browsers` označuje seznam prohlížečů, které budou testovány. Prostředí dovoluje uživateli zvolit verzi prohlížeče, ve které chce rozšíření testovat. Vybranou verzi lze indikovat doplněním za symbol `=` v konfiguračním souboru. Prostředí podporuje testování ve dvou prohlížečích. Dostupné verze ovladačů pro *Google Chrome* podléhají dostupným verzím repositáře *Chrome For Testing* [9] a repositáře *Firefox Public Releases* [18] pro prohlížeč *Mozilla Firefox*. Verzi lze indikovat i klíčovými slovy jako `stable`, `dev`, `beta`, nebo `canary`. V případě prohlížeče *Mozilla Firefox* lze testovat i za použití verze *Extended Support Release*, tedy `esr`.

- `pets` očekává nastavení klíč – hodnota ve formátu prohlížeč – seznamu seznamů. Vnější seznam obsahuje libovolný počet seznamů kratších. Hodnoty oddělené čárkou ve vnitřních seznamech indikují, která rozšíření je třeba nainstalovat před návštěvou stránky tvořící otisk prohlížeče testujícího. Jména rozšíření v seznamech jsou reprezentována jejich zkratkami. Rozšíření, která lze v době implementace nainstalovat, odpovídají rozšířením popsaným v kapitole 2.3. Pro správnou činnost integračních testů předpokládá prostředí existenci jednoho běhu, při kterém nebyla nainstalována žádná rozšíření prohlížeče. Seznam reprezentující takový běh bude obsahovat pouze jednu hodnotu a to `None`. Takový seznam musí být ve vnějším seznamu vždy na první pozici. Důvodem je, že nasazený server pro tvorbu otisku prohlížeče běží neustále od bodu spuštění kontejneru testovacího prostředí. Běh bez nainstalovaných rozšíření serveru napomáhá smysluplně organizovat výstupní soubory obsahující vypočtené atributy. Společně se zkratkou reprezentující rozšíření lze udat i jeho nastavení. V případě rozšíření *JShelter* lze vybrat úroveň ochrany využitou při návštěvě stránky (viz 2.2). Speciální nastavení rozšíření indikuje existence symbolu `_` v hodnotě seznamu. Hodnoty mohou symbolizovat i speciální případy pro testování. Například hodnoty `JS_NBS` a `JS_DLS` provedou testy interakce uživatele s nastavením rozšíření skrze jeho grafické uživatelské rozhraní. Hodnoty `JS_0`, `JS_1`, `JS_2` a `JS_3` říkají, jakou úroveň rozšíření testovat. Seznam zkratek reprezentující jednotlivá rozšíření lze nalézt v dokumentaci implementace ve formátu `README`. Instrukce jak přidat do prostředí nová rozšíření lze naleznout v repositáři projektu.

- Hodnota `delay` dovoluje testujícímu nastavit dobu návštěvy stránky pro výpočet atributů prohlížeče.

- `mode` stanovuje způsob návštěvy stránky. Pokud je běh spuštěn s hodnotou `threaded`, probíhají dílčí návštěvy pro prohlížeče paralelně. Pomocí argumentu `interactive` může testující sledovat vypočtené hodnoty a průběh návštěvy každé skupiny rozšíření. V tomto módu lze nahlédnout do vývojářské konzole prohlížeče.

- `integration_testing` reprezentuje, zda se má testovací modul integračních testů spustit ihned po dokončení návštěv stránky pomocí všech vyjmenovaných prohlížečů, jejich verzí a testovaných kombinací rozšíření.

Pro názornou ukázku uvažujme výpis 5.2 reprezentující konfigurační soubor klienta ve formátu *JSON*:

```
{
    "experiment_name": "exp01",
    "browsers": [ "chrome=stable", "firefox=esr"],
    "pets": {
        "chrome=stable": [ ["None"], ["JS_NBS"], ["JS_3"], ["JS_2"], ["UO"],
            ["JS_2", "UO"]],
        "firefox=esr": [ ["None"], ["JS_3"] ]
        },
    "delay": 35,
    "mode": "interactive",
    "integration_testing": true
}
```

Výpis 5.2: Příklad konfiguračního souboru pro klienta.

Hodnotu `experiment_name` lze nastavit podle uvážení a potřeb uživatele. Pro následující návštěvy v rámci testů budou využity dva různé prohlížeče – *Google Chrome* v aktuální verzi pro uživatele, tedy *stable* a *Mozilla Firefox* ve verzi *esr*.

Návštěva proběhne nejdříve skrze prohlížeč *Google Chrome* ve verzi *stable*. Jako první bude stránka pro výpočet atributů uživatele navštívena bez jakéhokoli nainstalovaného rozšíření. Tato návštěva je povinná a je vnímána jako referenční, tudíž musí vždy proběhnout první.

Druhý test proběhne nikoli ve formě návštěvy stránky, ale jako test interakce uživatele s uživatelským rozhraním rozšíření *JShelter*. Test kontroluje správné nastavení funkčnosti *Network Boundary Shield* [30]. Tento typ testu nevytváří výstupní soubor, výsledky testů vypisuje na standardní výstup.

Jako další proběhne návštěva stránky pro tvorbu otisku za využití rozšíření *JShelter* nastaveného do úrovně 3, tedy *přísná* [29]. Následně proběhnou dvě další návštěvy, při první bude nainstalováno rozšíření *JShelter* v úrovni 2 a při druhé rozšíření *uBlock Origin*. Jako poslední proběhne návštěva stránky, při které byla nainstalovaná obě rozšíření – *JShelter* v úrovni 2 i *uBlock Origin*. Při každé takové návštěvě generuje server soubor s výstupními výsledky.

Po provedení testů začne testování pomocí prohlížeče *Mozilla Firefox* ve verzi *esr*. Obdobně jako u předchozího prohlížeče proběhne první návštěva bez instalace jakéhokoli rozšíření. Následně proběhne návštěva s rozšířením *JShelter* v úrovní 3.

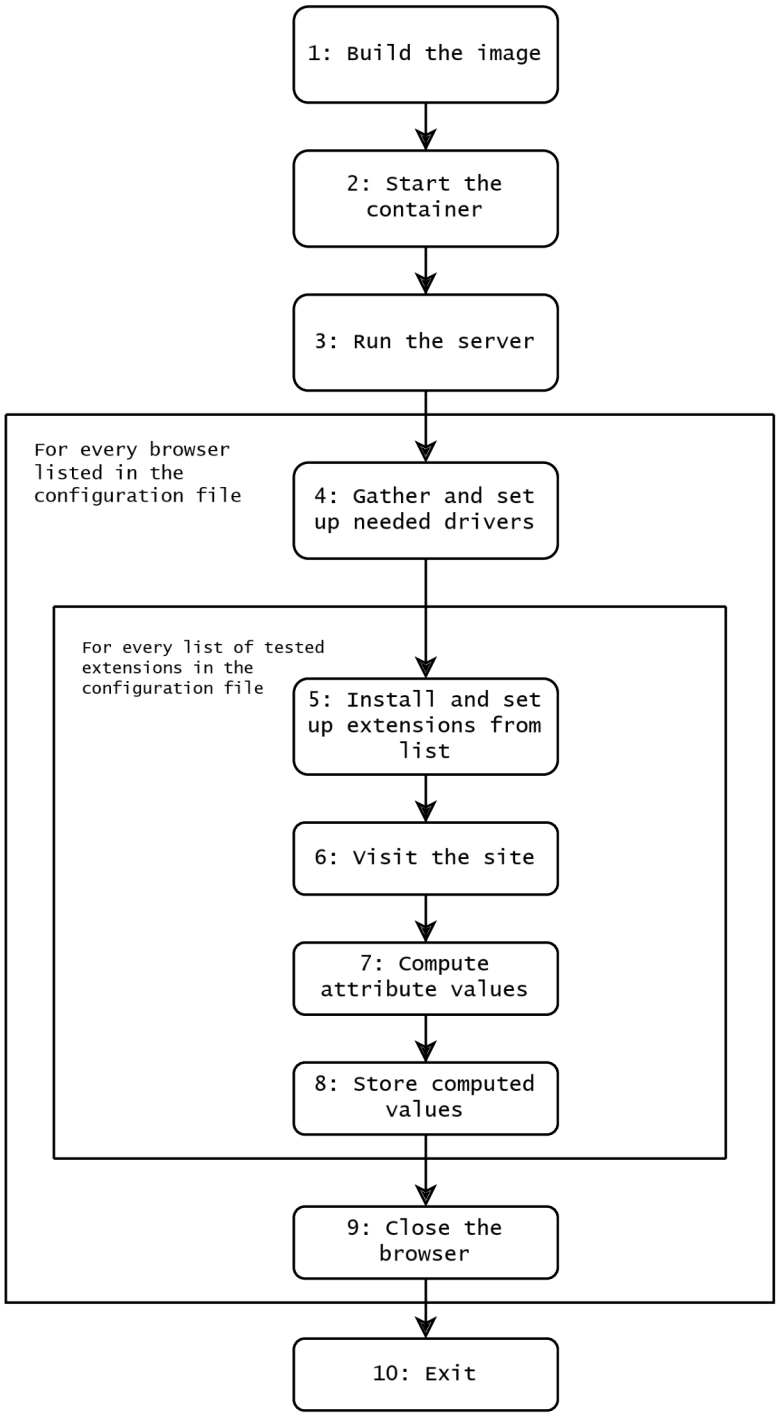
Návštěva testovací stránky bude pro každou kombinaci rozšíření trvat třicet pět sekund. Návštěvy proběhnou jedna po druhé.

Jelikož je hodnota proměnné `integration_testing` nastavena na `true`, proběhne ihned po dokončení všech návštěv test sesbíraných dat.

V případě konfiguračního souboru výše by návštěva stránky proběhla celkem sedmkrát, což odpovídá počtu seznamů bez testů konfigurace uživatelského nastavení. Ovladač prohlížeče by byl nastaven osmkrát, to odpovídá celkovému počtu vnitřních seznamů. Výsledkem by bylo sedm výstupních souborů a výsledky testu nastavení na standardním výstupu.

5.1.3 Průběh testování

Celý proces integračního testování se skládá ze tří stěžejních částí: výpočet dat, instanciace hodnot a test na očekávané výsledky. Diagram 5.2 zobrazuje proces získávání výstupních hodnot.



Obrázek 5.2: Diagram znázorňující průběh procesu sbírání výstupních dat.

Poté co je sestaven obraz nástrojem *Docker* je ihned po spuštění kontejneru nasazen webový server, který slouží pro výpočet atributů prohlížeče při návštěvách.

Nastavení ovladače prohlížeče

Po nasazení čeká server na návštěvu klientem. Ještě před návštěvou je nutno upravit nastavení ovladače prohlížeče. Mimo nastavení *SOCKS5 proxy* je nutno provést několik kroků.

Prohlížeč je v rámci programu vnímán jako instance objektu `Browser`. Pro oba testované prohlížeče je třeba zvolit správnou verzi jeho ovladače. O získání správné verze ovladače se stará *Selenium Manger*. Verze prohlížeče odpovídá verzi zvolené v konfiguračním souboru výše. Proměnná `browser` odpovídá hodnotám prohlížečů v konfiguračním souboru.

V případě prohlížeče *Mozilla Firefox* je nutné upravit využitý uživatelský profil tak, aby bylo možné testovat čtení geolokace uživatele. Výpis 5.3 zachycuje nejdůležitější prvky nastavení, které je nutné provést pro správné určení polohy v prohlížeči v rámci automatizované návštěvy. V prostředí kontejneru chybí ve výchozím profilu odkaz na *API* zprostředkující data o poloze uživatele a musí být nakonfigurován pomocí *Selenia*. Dále je nutné dovolit prohlížeči přijímat nepodepsané certifikáty, jelikož k lokálnímu webovému serveru je přistupováno pomocí protokolu *HTTPS* bez existence platných certifikátů. Pro momentální potřeby běhového prostředí a správného průběhu testů je takové řešení dostačující.

```
fp.set_preference("geo.enabled", True)
fp.set_preference('geo.prompt.testing', True)
fp.set_preference('geo.prompt.testing.allow', True)
fp.set_preference("geo.provider.testing", True)
fp.set_preference("geo.provider.network.url",
    "https://location.services.mozilla.com/v1/geolocate?key=test")
fp.set_preference("accept_insecure_certs", True)
firefox_options.profile = fp
```

Výpis 5.3: Konfigurace *Mozilla Firefox* pro geolokaci.

Prohlížeč *Google Chrome* vyžaduje pro běh v kontejneru v testovacím režimu další nastavení. Jedná se o specifika vyžadována přímo ovladačem *Chrome for Testing* a prostředím kontejneru zobrazená ve výpisu 5.4. Stejně tak je nutností dovolit prohlížeči přijímat neplatné certifikáty. Krok získání ovladače a jeho nastavení je v diagramu 5.2 znázorněn jako krok číslo 4.

```
chrome_options.add_argument("--disable-gpu")
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument('--disable-dev-shm-usage')
chrome_options.add_argument('--ignore-certificate-errors')
chrome_options.add_argument("--dns-prefetch-disable")
chrome_options.add_argument("enable-automation")
```

Výpis 5.4: Konfigurace *Google Chrome* pro správný průběh testování v rámci kontejneru.

Následně jsou pomocí ovladače nainstalována rozšíření vybraná v konfiguračním souboru. Pro stažení ovladače pomocí *Selenium Manager* je vyžadováno internetové připojení. Rychlost nastavení se odvíjí od rychlosti připojení. Instalaci a nastavení rozšíření reprezentuje v diagramu 5.2 krok číslo 5.

Speciální pozornost je věnována rozšíření *JShelter* a úrovni, ve které je očekáváno jeho testování. Nastavení musí proběhnout po instanciaci ovladače prohlížeče. Při volbě testovací

úrovně je ještě před návštěvou stránky pro výpočet atributů otisku prohlížeče nutné změnit úroveň ochrany popsané v sekci 2.2. Způsob, jakým *Google Chrome* a *Mozilla Firefox* přiřazují identifikační řetězce je odlišný, tudíž je nutné úrovně nastavit odděleně.

Google Chrome přiřadí rozšíření unikátní ID v rámci *Chrome Web Store*[1], tedy po každé instalaci je rozšíření přiřazeno stejné konstantní ID. V případě vlastního sestavení je ale rozšíření přiřazeno ID odlišné. Je tedy nutné ID získat přímo z prohlížeče návštěvou stránky `chrome://extensions/`. Poté nastavení probíhá pomocí přesměrování na stránku globálního nastavení *JShelter* a přepnutí úrovně pomocí vyhledávací funkce ovladače.

Mozilla Firefox přiřadí rozšíření náhodné ID generované po jeho instalaci. Pro získání ID rozšíření *JShelter* je potřeba nejprve navštívit jednu z konfiguračních stránek prohlížeče *Mozilla Firefox*, která nainstalovaná rozšíření zobrazuje. V implementaci bylo využito stránky `about:debugging#/runtime/this-firefox` a ID bylo nalezeno pomocí existence řetězce *JShelter* na stránce.

V případě `JS_NBS` a `JS_DLS` jsou provedeny testy uživatelského rozhraní po vytvoření instance ovladače. V rámci diplomové práce bylo popsáno rozšíření *Ghostery*, které ihned po jeho instalaci vyžaduje dodatečná oprávnění. Pokud bylo rozšíření *Ghostery* nainstalováno, jsou mu oprávnění ovladačem dodána. V diagramu 5.2 symbolizuje proces nastavení rozšíření krok číslo 5.

Návštěva vybraných stránek

Po nastavení příslušného ovladače přechází program k návštěvám stránek definovaných v `fp_sites` v konfiguračním souboru běhu. Na každé stránce zůstává po dobu nastavitelnou testujícím. Při volbě doby návštěvy je nutno brát v potaz, že komunikace se serverem probíhá skrze protokol *HTTPS*, pro generování správného výstupu je tedy důležité, aby měl server dostatečné množství času na výpočet atributu prohlížeče, uložení těchto atributů do souboru a zároveň dostatek času na ustanovení spojení a jeho řádného ukončení. Na straně serveru probíhá generování hodnot s odstupem až devíti sekund, je tedy vhodné předpokládat alespoň deset sekund pouze na výpočet atributů. Potřebný čas se výrazně odvíjí od výpočetní kapacity hostujícího zařízení. V případě zařízení s nízkou výpočetní kapacitou se testujícímu doporučuje prodloužit dobu návštěvy. Ovladač dále klikne na tlačítko nutné ke čtení atributů spojených se zvukovým výstupem zařízení uživatele. Tento krok odpovídá bodu číslo 6 v diagramu 5.2.

Výpočet atributů otisku prohlížeče

Role výpočtu získaných atributů se zcela přesouvá na server. Server je nasazen ihned po spuštění kontejneru a běží po celou dobu jeho existence.

Čtení hodnot atributů probíhá v jazyce *JavaScript*. Před nasazením server získá do povědomí všechny zkoumané atributy pomocí načtení souborů ve složce `attributes`. Díky získaným souborům může vykreslit úvodní stránku, kterou klient navštíví. Na domovské stránce jsou zobrazeny získané atributy společně s podpůrným obsahem důležitým pro jejich správný výpočet. Proměnná `request` je serveru předána skrze *URL* při návštěvě uživatele. Obsahuje hodnoty odpovídající konfiguračnímu souboru klienta.

Při návštěvě stránky server volá metodu `store`. Metoda uloží vygenerované atributy ve formátu *JSON*. Vygenerované hodnoty mohou být před uložením upraveny, například může být z hodnot vytvořen *hash*. Výstupní soubory jsou kategorizovány podle použitého

[1] https://chromewebstore.google.com

prohlížeče a času zahájení testu prohlížeče. Soubory jsou pojmenovány podle využitých rozšíření.

Metody pro výpočet atributů je možno kdykoliv přidat. Stačí vytvořit dva soubory ve složce `/attributes/standard/js`. Soubor ve formátu *JSON*, který udává parametry potřebné k uložení a zobrazení atributu na domovské stránce a *JavaScriptový* soubor obsahující metodu získání parametru. Detailní popis vytvoření metody pro získání atributu otisku prohlížeče lze nalézt v dokumentaci běhového prostředí v jeho repositáři. Atributy samotné a jejich význam při výpočtu otisku prohlížeče jsou popsány v jejich příslušném *JSON* souboru.

Při testování rozšíření se mohou vyskytnout případy, kdy uživatel či vývojář nemá zájem získat všechny hodnoty atributů ve výchozí verzi běhového prostředí. Například v situaci, kdy je uživateli známo chování rozšíření a výpočet atributu, který zůstane rozšířením nepozměněn, je zbytečný.

Pokud uživatel nemá zájem vypočíst a testovat hodnotu dostupných atributů, může změnit obsah proměnné `tested_attributes` v souboru `env_config.py` ve zdrojové složce serveru. Ve výchozím stavu jsou testovány všechny atributy, pro které byly implementovány výpočetní metody. Nechce-li uživatel, aby k výpočtu hodnoty atributu docházelo, stačí jej ze seznamu v proměnné `tested_attributes` odstranit. Stejně tak vytvoří-li uživatel novou metodu pro výpočet atributu, musí ji do seznamu přidat. Integrační testování reflektuje tuto možnost a při instanciaci objektů vnímá výchozí hodnotu všech atributů jako `None`. Pokud se hodnota atributu nenachází ve výstupním souboru, dá se předpokládat, že nedošlo k jejímu výpočtu záměrně a modul *pytest* test oproti očekávaným hodnotám atributu přeskočí.

Výstupní soubory jsou serverem uloženy v adresářové struktuře odpovídající datu zahájení testů, která dále obsahuje výstupní soubory roztříděné podle použitého prohlížeče. Proces výpočtu a ukládání atributů otisku prohlížeče je v diagramu 5.2 znázorněn čísly 7 a 8.

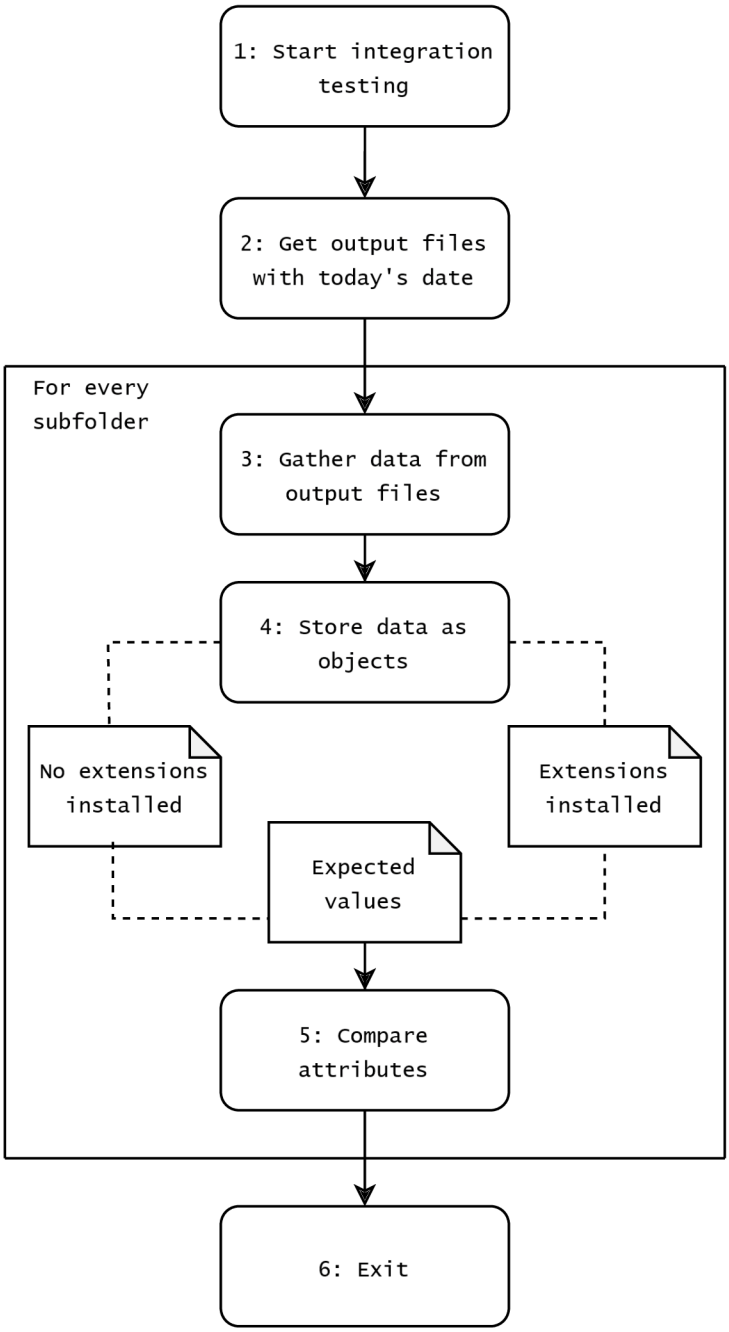
Provedení integračních testů

Pokud byl klient konfigurován v možnosti `integration_testing` hodnotou `true`, začne ihned po ukončení návštěv stránky integrační testování. Diagram znázorňuje průběh integračního testování.

Logika integračního testování je zcela převzata z diplomové práce pana Bednáře [8]. Rozdíl spočívá v její implementaci. V původních integračních testech přebíral roli výpočtu atributů i testování klient, tedy ovladač prohlížeče. Nyní modul integračního testování nevypočítává žádné hodnoty. Místo toho zpracovává výstupní soubory, které vytvořil server.

Role integračního testování spočívá v instanciaci objektů reprezentujícího tři základní stavy vypočtených hodnot. Tyto hodnoty odpovídají definici objektu `TestedValues` v souboru `values_tested.py` a značí:

- Objekt reprezentující hodnoty získané serverem při použití prohlížeče, ve kterém nebyla v průběhu návštěvy nainstalována žádná rozšíření.
- Objekt reprezentující hodnoty získané serverem při použití prohlížeče, ve kterém byla v průběhu návštěvy nainstalována rozšíření, nebo jejich kombinace.
- Objekt reprezentující hodnoty, které jsou testujícím udány jako očekávané při použití rozšíření nebo jejich kombinací.



Obrázek 5.3: Diagram znázorňující průběh integračního testování.

Testovací modul prochází adresáře odpovídající použitým prohlížečům. Pro každý adresář vytvoří na základě odpovídajícího výstupního souboru objekt, který obsahuje atributy vypočtené bez použití rozšíření. Stejně tak vytvoří další objekty odpovídající atributům za použití rozšíření. Na základě těchto objektů a použitých rozšíření zvolí ze souboru `values_expected.py` odpovídající instanci objektu reprezentující hodnoty, které jsou pro danou kombinaci rozšíření očekávány. Tento proces je v diagramu 5.3 znázorněn krokem číslo 4. Definice objektů byla pro účely práce převzata z původní implementace integračních testů [8] a obohacena o další atributy.

Objekt `noaddon` značící hodnoty vypočtené při návštěvě bez použití rozšíření se v rámci testovaných prohlížečů neliší. Objekt `addonRun` je nahrazen aktuální reprezentací hodnot za použití rozšíření.

Definice objektu očekávaných hodnot je zvolena na začátku testu modulu *pytest* pomocí jeho konfiguračního souboru. Hodnoty jsou vybrány podle řetězce reprezentujícího rozšíření nainstalovaná v průběhu návštěvy. Řetězec obsahuje zkratky rozšíření dané konfiguračním souborem klienta oddělené symbolem _. Na pořadí zkratek v řetězci záleží, zkratky rozšíření je nutné při instanciaci objektu očekávaných hodnot udávat v abecedním pořadí.

Po vytvoření objektů přechází *pytest* na provedení dílčích integračních testů podle jejich definice ve složce `tests_definition`. Příklad testu na očekávané hodnoty zachycuje výpis 5.5. Výsledky testů jsou vypsány na standardní výstup. V diagramu 5.3 se jedná o krok číslo 5.

```
def test_to_data_URL(noaddon, addonRun, expected):
    if get_shared_addonRun().canvasDataURL is None:
        pytest.skip("Canvas attributes not tested.")
    if addonRun.canvasDataURL == "ERROR":
        print("\n toDataURL error.")
        assert False
    else:
        if expected.canvasDataURL == 'SPOOF VALUE':
            assert addonRun.canvasDataURL != noaddon.canvasDataURL
        else:
            assert addonRun.canvasDataURL == noaddon.canvasDataURL
```

Výpis 5.5: Příklad provedeného testu.

5.1.4 Modifikace běhového prostředí

Co se týče modifikace integračního modulu běhového prostředí testujícím, nebude technická zpráva práce zacházet do detailů. Všechny potřebné informace včetně popisu důležitých souborů jsou dodány v repositáři projektu ve formě souborů `README`. Pokud je při modifikaci potřeba věnovat speciální pozornost konkrétním funkcím či příkazům, nachází se tato upozornění přímo ve zdrojovém kódu implementace.

V otázce modifikace je ale nutno podotknout, že nelze vytvořit universální běhové prostředí, které vyhovuje testování všech existujících rozšíření ve všech jejich možných konfiguracích. Je tedy na testujícím aby pro případné nastavení rozšíření v prohlížeči vhodně využil možnosti nástroje *Selenium*. Jedná se o nastavení obdobné úrovním ochrany rozšíření *JShelter* [31].

Kromě nastavení konkrétních rozšíření je ale prostředí navrženo tak, aby bylo co nejvíce přívětivé případným modifikacím. Jedná se hlavně o přidání testovaných rozšíření, přidání metod pro výpočet atribut otisku prohlížeče a úpravu objektů reprezentujících vypočtené a očekávané atributy při použití konkrétního rozšíření, nebo jejich kombinací. Nejen že by měly být případné modifikace co nejjednodušší, rozhodně jsou vřele vítány. Postupem času se při vývoji internetu mohou objevit další *API*, které mohou být zneužity pro tvorbu otisku prohlížeče. Implementované běhové prostředí tedy v budoucnu očekává rozšíření testovaných atributů a testů na jejich hodnoty.

5.2 Systémové testování

Závislosti systémového testování podléhají stejným podmínkám jako v původní práci pana Bednáře [8]. Mimo původních závislostí systémových testů dochází ke stažení *Webdriver Manager*, který je zodpovědný za správu verzí ovladačů. Průběh je mimo tento fakt téměř identický, pro detailnější popis včetně příslušných diagramů se lze odkázat na práci samotnou.

5.2.1 Změny oproti původní implementaci

Sestavení obrazu probíhá podobně jako u integračních testů skrze soubor *Dockerfile*. Při sestavení vrstvy systémových testů dochází k poskytnutí modulů potřebných pro spuštění původní verze systémových testů (viz [8]).

Nová implementace systémových testů využívá konceptu kontejnerizace a automatickou správu ovladačů pomocí *Selenium Manager*. Uživatel již nemusí poskytnout ovladač ve zdrojových souborech. Ovladač je v kontejnerovém prostředí nutno nastavit podobným způsobem, jako v případě integračního testování.

V průběhu procházení *Tranco Top Sites* je nyní možné použít více rozšíření a jejich kombinací najednou. Soubory rozšíření je nutné poskytnout ve složce `addons`.

V konfiguračním souboru systémových testů lze poté doplnit rozšíření nainstalovaná v průběhu procházení. V případě přidání nového rozšíření je třeba vytvořit ve slovníku reprezentujícím rozšíření a jejich soubory novou dvojici klíč – hodnota ve formátu zkratka rozšíření – soubor rozšíření. V následující ukázce by byla při prohlížení nainstalovaná rozšíření *JShelter* a *uBlock Origin*.

```
_extensions_dict_chrome = { "gh": "Ghostery.crx",
                            "uo": "uBlockOrigin.crx",
                            "PB": "PrivacyBadger.crx",
                            "NS": "NoScript.crx",
                            "DDGPE": "DuckDuckGoPE.crx",
                            "Gh": "Ghostery_c",
                            "NC": "Netcraft.crx",
                            "DE": "Decentraleyes.crx",
                            "JS": "JShelter.crx"}

_extensions_to_test = ["JS", "uo"]
```

Výpis 5.6: Konfigurace rozšíření běhu

V původní formě testů získaných z repositáře projektu *JShelter*[2] docházelo při návštěvách stránek k chybám a program nikdy neprovedl úspěšnou analýzu výpisů, ani snímků obrazovky. Důvodem může být například značně odlišná verze ovladače v době původní implementace. Bylo tedy nutné systémové testy opravit do funkční podoby. Mimo opravu proběhlo přejmenování proměnných a výpisů tak, aby lépe reflektovaly průběh testování. Pro hlubší nahlédnutí do implementace systémových testů včetně odpovídajících diagramů lze nahlédnout do práce pana Bednáře [8], který je autorem původní implementace.

V době psaní práce stále neexistoval žádný způsob, jak získat záznamy o procházení v prohlížeči *Mozilla Firefox* a zároveň zachovat logiku získání dat původní implementace systémových testů. Prohlížeč totiž neimplementuje rozhraní `get_logs` využité při sbírání konzolových výpisů prohlížeče. Nicméně úprava běhového prostředí za účelem přizpůsobení

[2] https://pagure.io/JShelter/webextension/blob/main/f/tests/system_tests

sběru informací v průběhu návštěv za použití prohlížeče *Mozilla Firefox* by mohla být v budoucnu zajímavým cílem.

Kvůli povaze *Selenium Grid* nelze v tuto chvíli zvolit pro průchod jinou verzi prohlížeče *Google Chrome*, než *stable*. Nicméně autoři nástroje *Selenium* jsou si této skutečnosti vědomi a v budoucnu můžeme očekávat vývoj tímto směrem[3]. V případě vývoje je implementace připravena poskytnout různé verze prohlížeče pro testování, podobně jako v případě integračních testů.

[3] https://github.com/SeleniumHQ/selenium/issues/12530

Kapitola 6

Testování prostředí a kompatibilita rozšíření

Následující kapitola popisuje proces testování implementace běhového prostředí a jejich výsledky. Testování celého projektu bylo rozděleno na dvě hlavní části. První část ověřuje, že výsledné prostředí splňuje celkovou chtěnou funkcionalitu a jedná se o plnohodnotné vylepšení současných integračních a systémových testů projektu *JShelter*. Druhá část testů zkoumá kompatibilitu rozšíření *JShelter* s ostatními rozšířeními dle zadání.

Všechny dále popsané testy byly provedeny na dvou strojích, přičemž první stroj obsahoval operační systém *Windows* a druhý operační systém *Linux*.

- Operační systém *Windows*
  - *Microsoft Windows 11 Pro*
  - *Docker Desktop* verze *4.27.1*
  - *X410 server* verze *4.0.0*
  - 16GB RAM
- Operační systém *Linux*
  - *Linux Mint 21.3 "Virginia", Cinnamon Edition*
  - *Docker Engine* verze *25.0.5*
  - 4GB RAM

6.1 Chování běhového prostředí

Podmínkou úspěšné implementace bylo vytvořit prostředí, které přejímá základní funkcionalitu komponentů zmíněných v sekci 4.1 a kombinuje je způsobem vhodným pro testování rozšíření a jejich kombinací. Jelikož cíle integrační a systémové části testování jsou rozdílné, testování bylo provedeno odděleně.

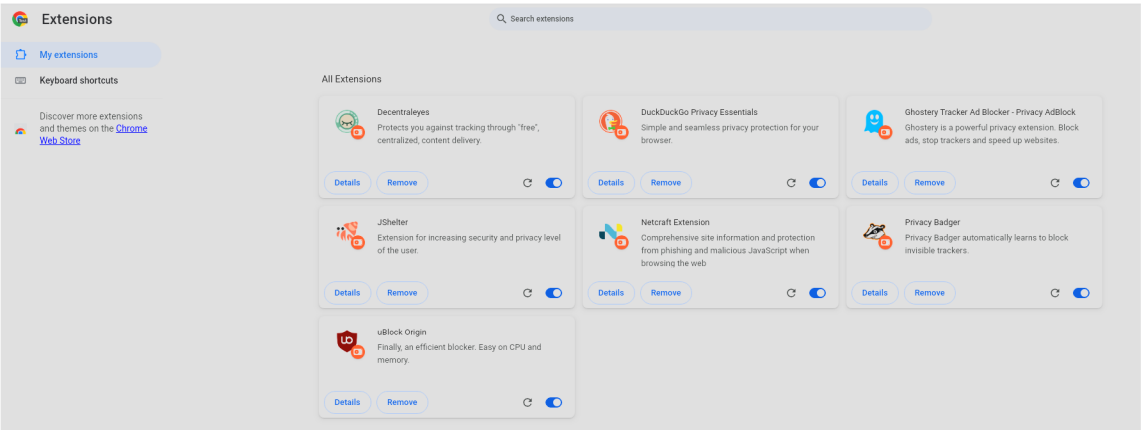
Chování integračních testů

Cílem testů bylo zjistit, zda prostředí v případě integračních testů splňuje následující náležitosti:

- Návštěva stránky pro výpočet atributů otisku prohlížeče proběhne správně alespoň v případě rozšíření popsaných v sekci 2.3 této práce[1]. Stejně tak správně proběhne instalace vybraných rozšíření.
- Prostředí zcela pokrývá množinu testovatelných atributů zahrnutých jak v projektu *PETInspector*, tak ve stávajících integračních testech projektu *JShelter*. Atributy lze vypočítat pro různé kombinace testovaných rozšíření.
- Při návštěvě bude správně využito verzí prohlížečů definovaných v konfiguračním souboru klienta.
- Bude možné otestovat vypočtené hodnoty atributů proti očekávaným hodnotám atributů za použití jakéhokoli rozšíření nebo kombinace rozšíření podobně jako v případě stávajících integračních testů projektu *JShelter*.
- Na straně serveru bude možno zvolit, jaké atributy ze seznamu dostupných atributů budou vypočteny.
- V případě chyby úmyslně zavedené do objektu reprezentujícího očekávané hodnoty a do definic rozšíření samotných zvládne integrační testování chybu rozpoznat.

Návštěva stránky pro výpočet atributů

Pro posouzení splnění první podmínky byl spuštěn běh za použití *Google Chrome* a *Mozilla Firefox* v aktuální verzi *stable*. Nejprve došlo k návštěvě stránky, při které bylo použito každé rozšíření ze sekce 2.3 individuálně. Následně došlo k návštěvě, při které byla nainstalována všechna rozšíření zmíněná v práci najednou.



Obrázek 6.1: Snímek obrazovky potvrzující správnou instalaci rozšíření.

V případě všech testovaných prohlížečů proběhla návštěva bez komplikací a na konci návštěvy byl úspěšně vygenerován výstupní soubor. Server v průběhu návštěv jeho stránky nehlásil žádnou chybovou hlášku. Přímo v prohlížeči bylo poté možné ověřit, zda jsou rozšíření správně nainstalována (viz obrázek 6.1).

[1]Správnost lze předpokládat i u jiných rozšíření, ale nelze otestovat každé existující rozšíření.

Dostupnost výpočtu atributů

V rámci výpočtu atributů otisku prohlížeče se na straně serveru podařilo implementovat naprostou většinu technik původních integračních testů. Na straně serveru se bohužel nepodařilo ověřit implementaci ochrany atributů při použití `web worker` [55] jazyku *JavaScript* původních integračních testů. V době psaní diplomové práce totiž testy v původní implementaci neprobíhaly[2]. Atributy získané původním projektem *PETInspector* byly ponechány, redundantní atributy získané v rámci integračních testů i projektu *PETInspector* byly odstraněny. Při posouzení splnění druhé podmínky implementace nejsou tedy zmíněné atributy brány v potaz.

Druhá podmínka byla testována referenčním během, při kterém bylo použito prohlížečů *Google Chrome* a *Mozilla Firefox* v aktuální verzi *stable*. Pro každý prohlížeč proběhla jedna referenční návštěva testovací stránky bez nainstalovaného rozšíření a dalších dvanáct návštěv odpovídajících prohlížení za použití individuálních rozšíření, včetně dostupných testovacích úrovní. Následně proběhlo několik návštěv za použití různých kombinací rozšíření. Pro každý měřený atribut byla při každé takové návštěvě v odpovídajícím výstupním souboru vytvořena vypočtená hodnota. V tomto případě jsou validní i hodnoty značící chybu při výpočtu, pokud o nich existuje ve výstupním souboru záznam. Chybová hláška může být výsledkem chování konkrétních rozšíření, nebo jejich kombinací. Důležitá je existence takového záznamu pro další testování.

Jedinou výjimkou bylo rozšíření *NoScript*, pro které nebyl vygenerován žádný výstupní soubor. Takové chování je standardní, jelikož rozšíření blokuje ve výchozí konfiguraci všechna volání *JavaScriptových* funkcí. To zahrnuje i funkce obsažené v souboru `clientAPI.js`, pomocí kterých server generuje a ukládá data vypočtená navštívenou stránkou.

Správné verze prohlížečů

Pro ověření funkčnosti nástroje *Selenium Manager* a správné využití verzí prohlížečů byl proveden běh, při kterém bylo použito několik verzí prohlížečů *Google Chrome* a *Mozilla Firefox*. Pro oba prohlížeče proběhly návštěvy za použití verzí *stable*, *beta*, *dev* a *canary*. Pro prohlížeč *Mozilla Firefox* byla navíc otestována funkčnost verze *esr*. Mimo verze popsané jejich aktuálností proběhly také návštěvy za použiti konkrétní verze prohlížeče. Všechny testované verze a výsledky pokusů o získání jejich ovladače jsou shrnuty v tabulce 6.1. Pro testování lze zvolit všechny verze, které momentálně poskytuje webové *API* odpovídajícího prohlížeče.

Nástroji *Selenium Manager* se nepodařilo získat všechny verze ovladačů pro prohlížeč *Mozilla Firefox*. Existuje tedy limit ovladačů, které lze v rámci prostředí použít. Podle dotázaných verzí se ale zdá, že dostupnost je ovlivněna především stářím verze ovladače a novější verze – *100.0* a výš – lze využít bez problému. Verze označené jako *beta* verze v repositáři bylo možno testovat, ale při použití verze označené jako *esr* došlo k chybě. V prohlížeči *Mozilla Firefox* lze tedy běhové prostředí v případě *Extended Support Release* momentálně testovat pouze v aktuální verzi pomocí hodnoty `firefox=esr`.

V případě *Google Chrome* nástroj úspěšně získal každou verzi prohlížeče od verze *113* a výš. Verze *112* a nižší *API* dotazované nástrojem *Selenium* neposkytuje.

Pokud se nástroji nepodařilo verzi získat, použil k návštěvě stránky výchozí verzi, tedy verzi obsáhlou v *cache Selenium Manager*. To samé chování demonstroval v případě nevalid-

[2] https://pagure.io/JShelter/webextension/blob/main/f/tests/integration_tests/testing/tests_definition/test_07_ECMA_arrays.py

Google Chrome	*Mozilla Firefox*
`stable` – OK	
`beta` – OK	
`dev` – OK	
`canary` – OK	
	`esr` – OK
`113` – OK	`105.0b3` – OK
`117.0.5917.0` – OK	`115.0b8` – OK
`122.0.6190.0` – OK	`115.5.0esr` – *Unable to discover*
`124.0.6335.0` – OK	`118.0.1` – OK
`125.0.6377.0` – OK	`120.0b7` – OK
`112` – *Unable to obtain*	`75.0b6` – Chyba 64
`126.0.6423.0` – OK	`38.0` – Chyba 255
`126.0.6424.0` – OK	`90.0` – Chyba 0
`121` – OK	`115.5.0` – OK

Tabulka 6.1: Testované verze prohlížeče.

ního řetězce verze, jako například `abc`. Pokud byla *cache* prázdná, nebylo možné návštěvu provést vůbec.

Správnou verzi prohlížeče bylo následně možné ověřit po zobrazení okna prohlížeče v sekci dedikované informacím o prohlížeči. Ve všech případech, kdy byl ovladač úspěšně stažen, byla návštěva provedena v očekávané verzi. Správnost použitých verzí byla ve všech případech zkontrolována ručně.

Po konzultaci s vedoucím této práce se dostupnost verzí v čase implementace ukázala jako dostatečná, jelikož všechny očekávané verze prohlížečů byly dostupné. Jedná se převážně o nejnovější verze prohlížečů společně s několika staršími verzemi. Nedostupné verze by nemuseli být pro testujícího zajímavé.

Testování vypočtených hodnot

Díky existenci výstupních souborů ze sekce 6.1 lze ověřit korektní chování v otázce porovnání vypočtených hodnot proti očekávaným hodnotám. Ověření nejprve proběhlo pro návštěvy, při kterých bylo nainstalováno rozšíření *JShelter* ve všech dostupných konfiguracích. Ověřit funkčnost testování bylo možné skrze referenční implementaci původních integračních testů projektu.

V případě všech návštěv za použití různých úrovní ochrany *JShelter* byly testy úspěšné. Lze tedy říci, že v době implementace stávající verze rozšíření *JShelter* (0.18) podávala očekávané výsledky.

Stejnou logikou byla testována i další rozšíření. Ve všech případech rozšíření i jejich kombinací byla data úspěšně načtena a porovnána s očekávanými hodnotami. V mnoha případech testy neprošly, ale takové chování bylo očekáváno.

V otázce výsledků testů dalších rozšíření nelze přesně určit, jaké výsledné hodnoty atributů očekávat. Pro nezasvěceného testujícího by pochopení muselo přijít po studii zdrojového kódu rozšíření, nebo dokumentů dodaných vývojářem. Je na testujícím, aby sám udal běhovému prostředí hodnoty, oproti kterým testovat.

V rámci testování hodnot proběhla návštěva pomocí individuálních rozšíření. Očekávané hodnoty naměřených atributů byly pro integrační testování zadány podobně, jako pro úroveň 0 rozšíření *JShelter*, tedy návštěva bez využití rozšíření. Návštěvy probíhaly ve snaze odhalit vnitřní chování nainstalovaných rozšíření:

- Po spuštění proběhla referenční návštěva stránky pro výpočet atributů prohlížeče bez nainstalovaných rozšíření.
- Pro každé rozšíření následně proběhla návštěva stránky po instalaci samostatného rozšíření bez dalších kombinací rozšíření.
- Do souboru `values_expected.py` byly přidány objekty reprezentující očekávané hodnoty za použití individuálních rozšíření. Očekávané hodnoty naměřených atributů těchto objektů byly ekvivalentní očekávaným hodnotám naměřeným za použití rozšíření *JShelter* testovaného v úrovni 0. Testy vlastně očekávaly hodnoty, které nebyly rozšířením nijak pozměněny.
- Nakonec byly spuštěny integrační testy, díky kterým lze posoudit jak správnost výpočtu atributů prohlížeče, tak i efekt rozšíření na úpravu těchto vypočtených atributů. Výsledky integračního testování bylo nutné porovnat s dokumentací konkrétního rozšíření.

Při návštěvě stránky po instalaci rozšíření *CanvasBlocker* v prohlížeči *Mozilla Firefox* došlo ke změně hodnot atributů popsaných v tabulce 6.2. Změna atributů reflektuje chování popsané v dokumentaci rozšíření [34]. Vypočtené hodnoty se správně lišily od hodnot očekávaných ve všech případech zmíněných v dokumentaci. Nahlédnutím do naměřených hodnot lze usoudit, že rozšíření *CanvasBlocker* implementuje podobné techniky, jako úroveň 2 rozšíření *JShelter*, tedy metodu „malých lží“ [31]. Analýzou konkrétních hodnot atributů výstupních souborů pro rozšíření *CanvasBlocker* je možné vypozorovat snahu rozšíření měnit malé množství hodnot v případě zvukových dat a falsifikovat reálné hodnoty elementu *canvas* a *WebGL*, nejspíše za stejným účelem jako v případě rozšíření *JShelter* v úrovni 2.

Při návštěvě za použití rozšíření *Canvas Blocker* pro prohlížeč *Google Chrome* došlo ke změně vypočtených atributů pro element *canvas* a změně řetězce generovaného voláním metody `canvas.getContext`. Změna parametrů elementu *canvas* je podle dokumentace rozšíření korektní a očekávaná. Změny oproti očekávaným hodnotám zobrazuje tabulka 6.3.

Při návštěvě došlo v případě rozšíření *Privacy Badger* ke změně hodnoty `doNotTrack` objektu `Navigator`. Takové chování je podle dokumentace rozšíření korektní [17].

Mimo tyto hodnoty neprokázalo integrační testování rozšíření žádnou změnu. Tento fakt ale neznačí neefektivnost rozšíření, pouze prokazuje, že rozšíření nijak neupravují atributy, které lze vypočíst při návštěvě na straně serveru. Zároveň se jedná – v naprosté většině rozšíření zmíněných v této práci – o jejich standardní chování popsané dokumentací. Nicméně například u rozšíření *Ghostery* je absence změny hodnot atributů překvapivá, jelikož popis rozšíření na stránce `addons.mozilla.org` slibuje „zaměnění reálných dat za náhodné hodnoty“[3].

[3] https://addons.mozilla.org/en-US/firefox/addon/ghostery/

	API s odchylkou od očekávaných hodnot
Hardwarové specifikace	✓

Hodnoty elementu *canvas*	`getImageData`
	`to_data_URL`
	`to_blob`

Hodnoty elementu *navigator*	✓
Hodnoty *performance*	✓
Metody *ECMA polí*	✓
Generované hodnoty *time*	✓
Výsledky metod *toString*	✓

Hodnoty *webaudio*	`channel_data`
	`byte_time_domain`
	`float_time_domain`
	`byte_time_frequency`
	`float_time_frequency`

Hodnoty elementu *WebGL*	`other_parameters`
	`webgl_read_pixels`
	`webgl_to_data_URL`

Data o zeměpisném *umístění uživatele*	✓

Tabulka 6.2: Tabulka znázorňující výsledek testů na očekávané hodnoty při použití rozšíření *CanvasBlocker* pro *Mozilla Firefox*.

Volba testovaných atributů

Od běhového prostředí se očekává, že bude možné zvolit, jaké atributy otisku prohlížeče budou vypočteny a tudíž i které budou testovány. Pro ověření podmínky bylo spuštěno několik běhů.

V prvním běhu byl proveden výpočet všech dostupných atributů. Následné integrační testování proběhlo standardně. V rámci dalšího běhu server dle konfiguračního souboru neprováděl výpočet atributů `canvas_blob`, `device`, `IOdevices`, `navigator`, `screen` a `worker`. Na straně serveru i klienta proběhly návštěvy bez chyb. Výstupní soubor reflektoval volby v konfiguračním souboru a neobsahoval hodnoty ignorovaných atributů. Stejně tak byly v průběhu integračního otestování korespondující testy přeskočeny. V případě návštěv,

	API s odchylkou od očekávaných hodnot
Hardwarové specifikace	✓

Hodnoty elementu *canvas*	`getImageData`
	`to_data_URL`
	`to_blob`

Hodnoty elementu *navigator*	✓
Hodnoty *performance*	✓
Metody *ECMA polí*	`canvas.getContext`
Generované hodnoty *time*	✓
Výsledky metod *toString*	✓
Hodnoty *webaudio*	✓
Hodnoty elementu *WebGL*	✓
Data o zeměpisném *umístění uživatele*	✓

Tabulka 6.3: Tabulka znázorňující výsledek testů na očekávané hodnoty při použití rozšíření *Canvas Blocker* pro *Google Chrome*.

při kterých nebyly měřeny žádné atributy (proměnná `tested_attributes` byla prázdná), vygeneroval server výstupní soubor obsahující pouze informace získané statickou metodou tvoření otisku prohlížeče, tedy čtením *HTTP* hlaviček. Modul integračního testování stejně jako v předchozím případě reflektoval konfiguraci a všechny dílčí testy přeskočil.

Detekce úmyslně zavedené chyby

Kontrolu na tuto podmínku je možné provést pouze prostřednictvím rozšíření *JShelter*, jelikož předem známe správné očekávané hodnoty testů pro jednotlivé úrovně.

Pro ověření správné detekce chyby zavedené do objektu očekávaných hodnot byly provedeny jednoduché testy, ve kterých došlo k přejmenování objektů reprezentujících úrovně rozšíření *JShelter*.

- Očekávané hodnoty v souboru `values_expected.py` pro rozšíření *JShelter* v úrovni 2 byly zaměněny za očekávané hodnoty *JShelter* v úrovni 1.
- Očekávané hodnoty pro rozšíření *JShelter* v úrovni 2 byly zaměněny za očekávané hodnoty *JShelter* v úrovni 3.
- Očekávané hodnoty pro rozšíření *JShelter* v úrovni 3 byly zaměněny za očekávané hodnoty *JShelter* v úrovni 1.
- Očekávané hodnoty pro rozšíření *JShelter* v úrovni 3 byly zaměněny za očekávané hodnoty *JShelter* v úrovni 2.
- Záměna byla testována v prohlížeči *Google Chrome* i *Mozilla Firefox*.

Ve všech zmíněných případech došlo při integračním testování k selhání testů na hodnoty, ve kterých se zaměněné úrovně lišily. Integrační testování správně detekovalo chybu v atributech týkajících se výstupních zařízení, elementu *canvas*, *WebGL*, zvukového výstupu i informací o pozici uživatele pro relevantní úrovně *JShelter*.

Následně proběhlo úmyslné zavedení chyb do samotného rozšíření *JShelter*. Ve zdrojovém kódu došlo ke změně definice některých atributů úrovní v souboru `levels.js`[4]. Konkrétní změny zachycuje tabulka 6.4.

Změna hodnot úrovně 2	**Změna hodnot úrovně 3**
`"webgl"`: 1 →2	`"webgl"`: 2 →1
`"enumerateDevices"`: 2 →3	`"enumerateDevices"`: 3 →2
`"hardware"`: 1 →3	`"hardware"`: 3 →1
`"geolocation"`: 3 →6	`"geolocation"`: 6 →3

Tabulka 6.4: Přehled změn jako injekce chyby do zdrojového kódu rozšíření *JShelter*.

Číselné hodnoty v souboru `levels.js` reprezentují metodu ochrany v případě konkrétního atributu. Tabulka 6.5 znázorňuje hodnoty, ve kterých modul integračního testování zaznamenal odlišnost očekávané od vypočtené hodnoty atributu po záměrné injekci chyb do rozšíření *JShelter*.

Výsledky reflektuji chyby úmyslně zavedené do zdrojového kódu rozšířeni. Pro úroveň 2 nebyla při integračním testování detekována chyba, protože rozšíření mění v nastavení `"hardware": 3` hodnoty `deviceMemory` a `hardwareConcurrency` na hodnoty, které stále považuje úroveň za správné, v tomto případě se jednalo o hodnotu `4`.

[4] https://pagure.io/JShelter/webextension/blob/main/f/common/levels.js

	Odchylka hodnot úrovně 2	Odchylka hodnot úrovně 3
Hodnoty *WebGL*	`webgl_precisions`	`other_parameters` `webgl_precisions`

Hodnoty *geolokace*	`test_accuracy`
	`test_latitude`
	`test_longitude`
	`test_timestamp`
		`test_accuracy`
		`test_latitude`
		`test_longitude`
		`test_timestamp`

Hardwarové specifikace		`device_memory` `hardware_concurrency`

Tabulka 6.5: Chyba testu na očekávané hodnoty po injekci chyby do zdrojového kódu rozšíření *JShelter*.

Chování systémových testů

V případě systémových testů měla implementace za cíl dovolit uživateli testovat i jiná rozšíření než pouze *JShelter*. Mimoto také umožnit testovat různé kombinace rozšíření dalších. Dalším cílem prostředí bylo využít nově dostupný nástroj *Selenium Manger* k usnadnění každodenního používání díky automatickému získání prohlížečů a jejich ovladačů.

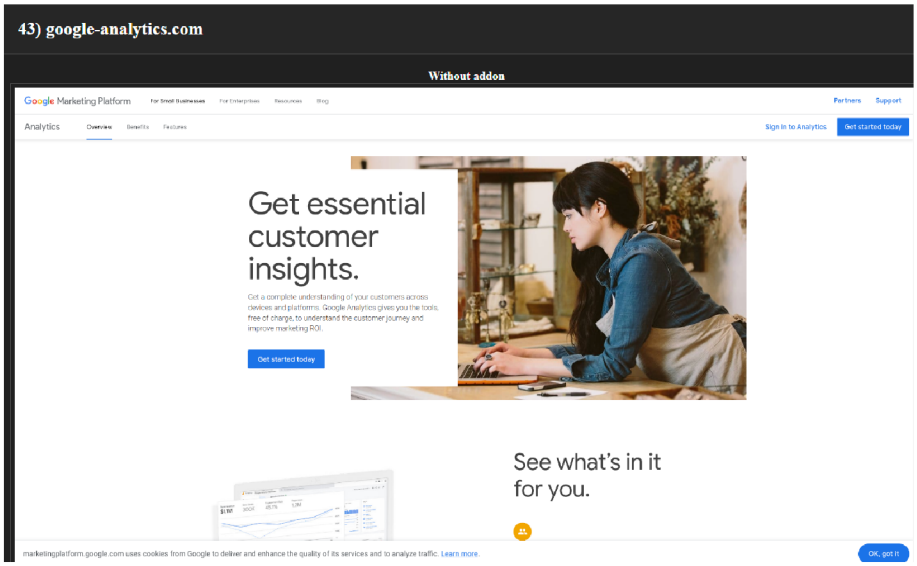
Pro každé rozšíření popsané v sekci 2.3 bylo provedeno několik běhů pro ověření funkčnosti implementace v případě použití jiných rozšíření a jejich kombinací. Všechny běhy skončily úspěšně a jejich výstupem byly odpovídající soubory ve formátu `.log` a `.png`, stejně jako v případě původní implementace systémových testů. Modul posuzující výsledky běhů podle konzolových výpisů prohlížeče a snímků obrazovky pořízených v průběhu návštěvy, implementovaný panem Bednářem, po úpravě stejně tak zdárně vyprodukoval očekávané výstupní soubory ve formátu `.html`. Výpisy reflektovaly použitá rozšíření.

Použití rozšíření mělo očekávaný dopad jak na analýzu konzolových výpisů prohlížeče tak na analýzu snímků obrazovky. Při návštěvách stránek za použití *uBlockOrigin* analýza výpisů prokázala velký výskyt chybových hlášek o blokování obsahu ze strany klienta, tedy rozšířením. Příklad jednoho z výpisů lze vidět na obrázku 6.2.

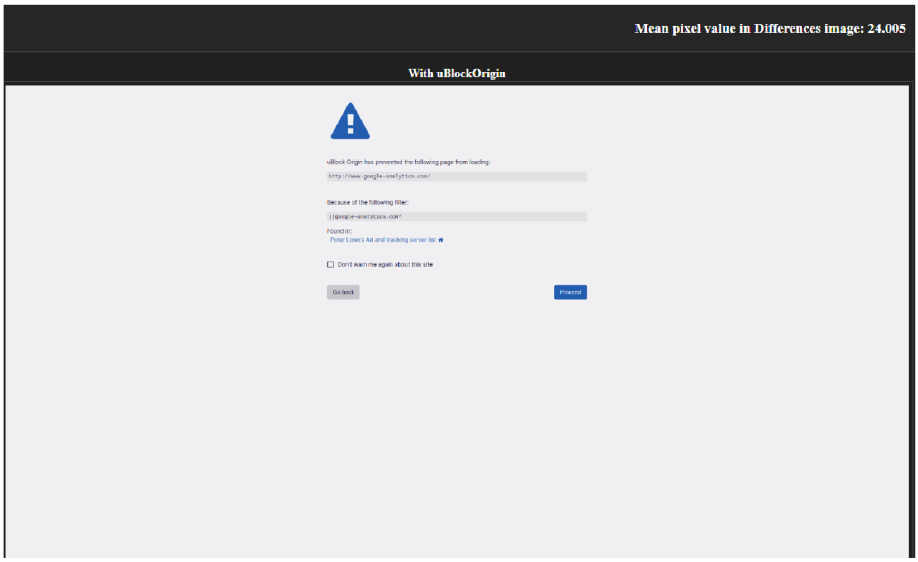
Obrázek 6.2: Příklad výpisu informujícího o blokování obsahu ze strany klienta při použití rozšíření *uBlockOrigin* na stránce *office.com*.

Stejně tak v případě testů snímků obrazovky došlo k projevu blokace obsahu rozšířením. V případě výpisů i snímků obrazovky se dá předpokládat, že použité seznamy blokovacích prvků [38] rozšíření *uBlockOrigin* vyhodnotily zdroj dat jako nevhodný a zakázaly tak získání dotazovaného obsahu. To lze vidět i na výpisu rozšíření na obrázcích 6.3, 6.4 a 6.5.

Systémové testování prokázalo změnu konzolových výpisů i snímků obrazovky pro každé rozšíření popsané v této práci. Dá se tedy říci, že testovací prostředí úspěšně integro-



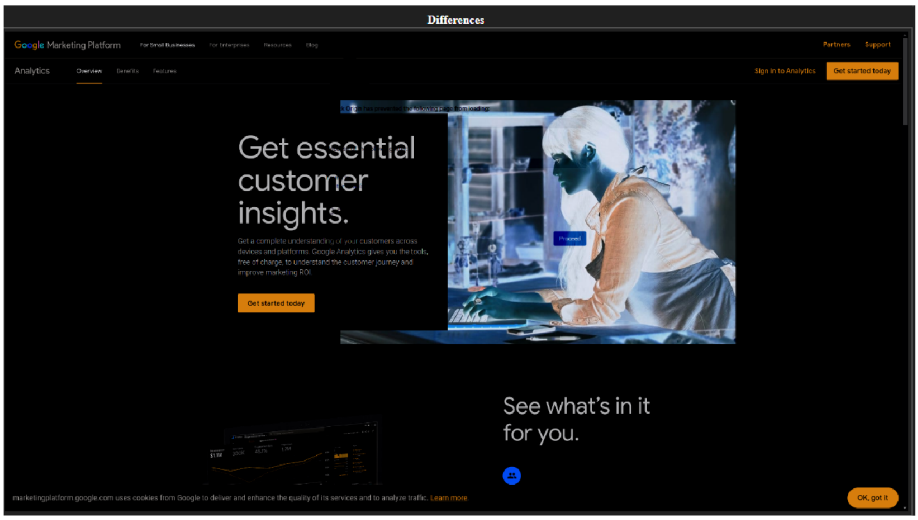
Obrázek 6.3: Snímek obrazovky pořízený bez rozšíření na stránce *google-analytics.com*.



Obrázek 6.4: Snímek obrazovky pořízený při použití rozšíření *uBlockOrigin* na stránce *google-analytics.com*.

valo i další rozšíření než původně zamýšlený *JShelter*. Tato práce se ale nutně nesoustředí na funkčnost rozšíření a příklady pro další rozšíření zde nebudou popsány. V případě zájmu může čtenář provést testy vlastní.

Sestavený obraz programu *Docker* neobsahoval žádnou verzi prohlížeče *Google Chrome*. Nicméně došlo při všech bězích ke správnému stažení jak prohlížeče, tak ovladače. Následné návštěvy stránek probíhaly standardním způsobem za použiti prohlížeče *Google Chrome* ve verzi *stable*.



Obrázek 6.5: Rozdíl předchozích snímků obrazovky na stránce *google-analytics.com*.

6.2 Kompatibilita rozšíření JShelter

Díky úspěšnému ověření očekávaného chování běhového prostředí v předchozí sekci můžeme prostředí použít k vyhodnocení kompatibility různých rozšíření a jejich kombinací s rozšířením *JShelter*. V otázce kompatibility rozšíření je nutné opět posoudit výsledky integračního a systémového testovaní zvlášť. Pro oba režimy lze totiž kompatibilitu v rámci práce definovat rozdílně. V následující sekci budou shrnuty výsledky testu kompatibility pro jmenované dílčí části a dále budou tyto výsledky interpretovány jako celek.

Integrační testování

Pro vyhodnocení kompatibility rozšíření je nutné nejdříve poukázat na povahu integračního testování. Integrační testování nemá za cíl vyhodnotit efektivnost jakéhokoli rozšíření v boji proti tvorbě otisku prohlížeče, nýbrž zkoumá hodnoty atributů, které lze na straně serveru při návštěvě získat. Jednoduše odpovídá na otázku, zda se rozšíření chová očekávaným způsobem. Běhové prostředí v módu integračního testování není navrženo tak, aby posoudilo efektivnost rozšíření. Efektivnost rozšíření se skládá z mnoha důležitých faktorů, jako je například chování rozšíření v průběhu dlouhodobého prohlížení webu. Integrační testování nemůže tento faktor posoudit. Mimo jiné testovaná rozšíření nemusí podle dokumentace čtené atributy měnit vůbec. Místo toho nabízí jiné techniky pro boj za internetové soukromí. To se prokázalo i v průběhu testování dílčích rozšíření (viz sekce 6.1). V návaznosti na tento fakt je třeba kompatibilitu rozšíření vnímat následovně:

- Návštěva testovací stránky pomocí rozšíření *JShelter* v kombinaci s jakýmkoli jiným rozšířením proběhne standardně a server úspěšně uloží výsledky výpočtů měřených hodnot.
- Při testování *JShelter* v kombinaci s libovolným rozšířením budou vypočteny stejné hodnoty, jaké jsou předpokládány v případě návštěvy za použití samotného rozšíření *JShelter*.
- Předchozí podmínky musí být splněny pro libovolnou konfiguraci rozšíření *JShelter*, tedy pro libovolnou testovanou úroveň definovanou v konfiguračním souboru.

V otázce kompatibility tedy nehraje roli, zda ostatní rozšíření zvyšují či snižují efektivnost rozšíření *JShelter*, jelikož testovací prostředí není navrženo tak, aby samotnou efektivnost posoudilo. Jedná se o posouzení očekávaných hodnot oproti hodnotám vypočteným.

Pro vyhodnocení kompatibility *JShelter* s ostatními rozšířeními bylo spuštěno celkem 168 běhů. Číslo reflektuje všechny možné varianty kombinace rozšíření *JShelter* a dílčích rozšíření popsaných v této práci v sekci 2.3, všechny dostupné úrovně nastavení *JShelter* popsané v sekci 2.2 a oba prohlížeče – *Google Chrome* a *Mozilla Firefox* ve verzích *stable*, *dev* a *beta* – pomocí kterých lze rozšíření v rámci běhového prostředí testovat. Jedinou výjimkou bylo rozšíření *NoScript*, a to kvůli poznatku popsanému v sekci 6.1.

Výsledky všech běhů byly vyhodnoceny modulem integračního testování. Pro každou kombinaci rozšíření *JShelter* s jiným rozšířením byl v souboru `values_expected.py` vytvořen objekt reprezentující očekávané hodnoty. To platí pro všechny testované úrovně rozšíření *JShelter*. Očekávané hodnoty atributů byly definovány stejným způsobem, jako hodnoty očekávané v případě použití pouze rozšíření *JShelter* pro konkrétní testovanou úroveň. Například objekt reprezentující očekávané hodnoty atributů vypočtené serverem při návštěvě za použití rozšíření *JShelter* v úrovni 2 byl stejný, jako objekt reprezentující očekávané hodnoty za použití *JShelter* v úrovni 2 a rozšíření *uBlock Origin*. Jedná se o logiku podobnou testovaní popsanému v sekci 6.1, ve které jsou dále zahrnuty i všechny možné testovací konfigurace rozšíření *JShelter*. Cílem bylo zjistit, zda některé rozšíření neočekávaně mění naměřené hodnoty atributů jiným způsobem než *JShelter* a v tomto ohledu znehodnocuje jeho funkčnost.

Integrační testování prokázalo odchylku očekávaných hodnot atributů od naměřených hodnot a to v případě použití rozšíření *JShelter* v úrovni 2 společně s rozšířením *CanvasBlocker* pro *Mozilla Firefox*. V případě použití obou rozšíření najednou se nepodařilo očekávaně změnit hodnoty `unmaskedVendor` a `unmaskedRenderer` parametru *WebGL*. Tabulka 6.6 ukazuje odchylky očekávaných a naměřených hodnot v případě použití kombinace těchto rozšíření.

	Původní hodnota	**Očekávaná hodnota**	**Získaná hodnota**
`vendor`	`'Mesa'`	`SPOOF VALUE`	`'Mesa'`
`renderer`	`'llvmpipe, or similar'`	`SPOOF VALUE`	`'llvmpipe, or similar'`

Tabulka 6.6: Tabulka původních, očekávaných a získaných hodnot *WebGL* při použití kombinace rozšíření *CanvasBlocker* a *JShelter*.

Při použití kombinace *JShelter* v úrovni 2 a *CanvasBlocker* se rozšíření *JShelter* nepodařilo korektně změnit výše popsané hodnoty a tím byla omezena jeho funkčnost.

Podle dokumentace rozšíření *CanvasBlocker* [34] je vhodné se dále zaměřit na výpočetní čas potřebný k modifikaci atributů. To hlavně v případech, kdy rozšíření *CanvasBlocker* slibuje úpravu stejných hodnot, jako rozšíření *JShelter*. V návaznosti na výsledky popsané v sekci 6.1 se lze soustředit u rozšíření *CanvasBlocker* konkrétně na dobu výpočtů atributů otisku, které jsou spojeny s elementy *canvas* a *WebGL*. Tabulka znázorňuje čas potřebný k získání těchto hodnot pro různé běhy. Výpočetní čas byl stanoven jako průměrný roz-

díl hodnot `performance.now()` na začátku a na konci volané funkce pro výpočet atributu pro padesát nezávislých běhů. Poměrně velký počet běhů byl proveden kvůli metodě *wrapperu* [28] rozšíření *JShelter*. Metoda počítá čas v milisekundách.

	Doba výpočtu *canvas*	**Doba výpočtu *WebGL***
Bez rozšíření	11,8	106,0
CanvasBlocker	94,2	150,2
JShelter úroveň 2	189,8	190,6
JShelter úroveň 3	205,6	128,6
JShelter úroveň 2 *CanvasBlocker*	271,2	442,2
JShelter úroveň 3 *CanvasBlocker*	224,6	135,125

Tabulka 6.7: Tabulka znázorňující výsledek testů na očekávané hodnoty při použití rozšíření *CanvasBlocker* pro *Mozilla Firefox*.

Tabulka 6.7 vypovídá o poměrně velkém poklesu výkonnosti výpočtu atributů v případě použití kombinace rozšíření *CanvasBlocker* a *JShelter* v úrovni 2. Rozdíl rychlosti se projevil i u kombinace *CanvasBlocker* a *JShelter* v úrovni 3, ale velikostně je zanedbatelný. Zvýšený výpočetní čas nemusí nutně značit nekompatibilitu rozšíření, ale jedná se o zajímavý indikátor, který by měl uživatel při výběru instalace rozšíření brát v potaz.

Pokles efektivity výpočtu v tabulce 6.7 společně s chybou na očekávané hodnoty z tabulky 6.6 značí, že rozšíření *CanvasBlocker* není kompatibilní s rozšířením *JShelter*.

V případě prohlížeče *Google Chrome* a rozšíření *Canvas Blocker* bylo podle sekce 6.1 v rámci kompatibility vhodné se zaměřit na atributy spojené s elementem *canvas*. V jakékoli kombinaci úrovně rozšíření *JShelter* u elementu však neexistoval rozdíl mezi naměřenou a očekávanou hodnotou. Jedinou odchylkou byla hodnota `toString` pro úrovně 2 i 3. Odchylka je zobrazena v tabulce 6.8. Tato odchylka se projevila již v testování rozšíření *Canvas Blocker* v sekci 6.1.

	Očekávaná hodnota	**Získaná hodnota**
`toString` `canvas.getContext`	`function getContext()` `[native code]`	`function ()` `[native code]`

Tabulka 6.8: Tabulka očekávaných a získaných hodnot při použití kombinace rozšíření *Canvas Blocker* pro *Google Chrome* a *JShelter*.

Podobně jako u rozšíření pro *Mozilla Firefox* má smysl se i u prohlížeče *Google Chrome* soustředit na výpočetní čas při použití rozšíření. Tabulka 6.9 zobrazuje průměrný čas výpočtu metod pro získání atributů otisku při použití rozšíření a jejich kombinací. Na rozdíl od rozšíření pro *Mozilla Firefox* nemá kombinace *Canvas Blocker* pro *Google Chrome* a *JShelter* výrazně negativní dopad na výpočetní čas.

	Doba výpočtu *canvas*	**Doba výpočtu *WebGL***
Bez rozšíření	22,59	55,15
Canvas Blocker	31,89	53,416
JShelter úroveň 2	171,45	179,378
JShelter úroveň 3	252,59	254,99
JShelter úroveň 2 *Canvas Blocker*	178,49	182,65
JShelter úroveň 3 *Canvas Blocker*	255,31	254,018

Tabulka 6.9: Tabulka znázorňující výsledek testů na očekávané hodnoty při použití rozšíření *Canvas Blocker* pro *Google Chrome*.

U běhů, při kterých bylo nainstalováno rozšíření *PrivacyBadger*, se změnila hodnota atributu popsaná v sekci 6.1.

Kromě výše popsaných situací se dá říci, že použití ostatních rozšíření popsaných v této práci společně s rozšířením *JShelter* nemá v rámci integračního testování negativní vliv na jeho funkčnost a neočekávaně nemění atributy vypočtené v průběhu návštěvy stránky pro výpočet otisku prohlížeče.

Systémové testování

Povaha systémového testování nedovoluje test na očekávané a získané hodnoty. V průběhu testování jsou navštíveny internetové stránky, jejichž chování není pro testovací prostředí deterministické a předvídatelné. Místo toho je třeba zkoumat výsledky analýzy konzolových výstupů prohlížeče a snímků obrazovky. Rozšíření lze považovat za kompatibilní s rozšířením *JShelter*, pokud splňuje tyto podmínky:

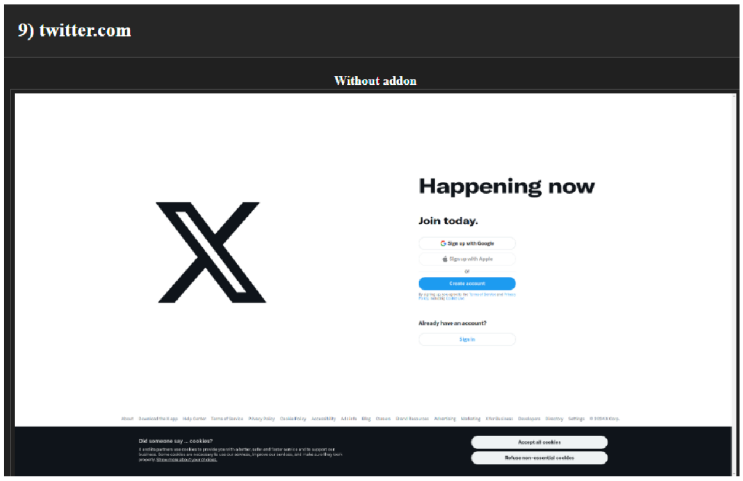
- Návštěva všech stránek pomocí rozšíření *JShelter* v kombinaci s jakýmkoli jiným rozšířením proběhne standardně a vygeneruje výsledky, které bude možné použít pro analýzu konzolových výstupů a snímků obrazovky.
- Předchozí podmínka musí být splněna pro libovolnou konfiguraci rozšíření *JShelter*, tedy pro libovolnou testovanou úroveň definovanou v konfiguračním souboru.

Obě podmínky je možné ověřit pomocí běhového prostředí. Co se týče dopadu použití kombinací rozšíření na konkrétní procházenou stránku, nelze kvůli nedeterministické a dynamické povaze navštívených stránek přesně říci, zda jsou rozšíření dokonale kompatibilní. Místo toho je třeba použít běhové prostředí k provedení několika běhů a vyhodnocení analýzy systémových testů pro každou kombinaci rozšíření *JShelter* v různých úrovních s dalšími rozšířeními popsanými v této práci.

V seznamu nejnavštěvovanějších stránek *TRANCO*[5] se stejně jako v původní implementaci systémových testů poměrně často objevují stránky, ke kterým nejde přistoupit vůbec. Prohlížeč v takových případech hlásí chybu `DNS_PROBE_FINISHED_NXDOMAIN` a to nejen v kontejnerovém prostředí, ale i mimo něj na hostujícím stroji. Takové stránky jsou patřičně označeny a jejich analýza neprobíhá.

Pro každou úroveň *JShelter* byly provedeny běhy v kombinaci s každým rozšířením popsaným v sekci 2.3, tedy třicet dva běhů, při kterých bylo navštíveno 100 stránek ze žebříčku *TRANCO*. K tomu byly provedeny čtyři referenční běhy po instalaci pouze rozšíření *JShelter* ve všech možných úrovních. Referenční běhy byly porovnány s běhy v ekvivalentní úrovni pro všechny různé kombinace *JShelter* s ostatními rozšířeními.

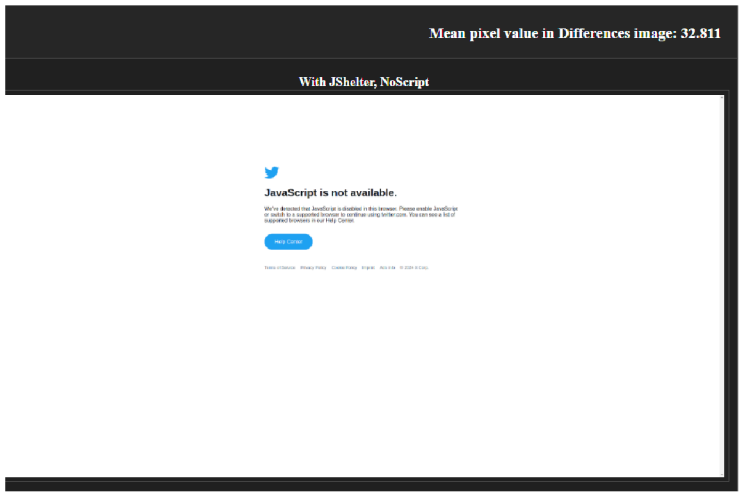
Zdaleka největší dopad na výsledek jak analýzy konzolových výpisů tak analýzy snímků obrazovky mělo s rozšířením *JShelter* ve všech úrovních rozšíření *NoScript*. Spousta navštívených stránek bez volání *JavaScriptových* funkcí ztratila dynamičnost a prakticky nefungovala vůbec. To se projevilo i poměrně velkým množstvím chybových výpisů konzole při analýze konzolových výpisů. Takové chování je ale korektní podle dokumentace rozšíření [37]. Příklad analýzy snímku obrazovky na stránce *twitter.com* lze vidět na obrázcích 6.6 a 6.7.



Obrázek 6.6: Snímek obrazovky pořízený bez rozšíření na stránce *twitter.com*.

Podle výsledků systémových testů lze usoudit, že ve výchozím nastavení nejsou rozšíření *JShelter* a *NoScript* z pohledu systémového testování kompatibilní, jelikož *NoScript* předem eliminuje výpočet atributů otisku prohlížeče a to ještě dřív, než by jej mohl *JShelter* jakkoli pozměnit. Tato kombinace tedy jistě nebude vhodná pro průměrného uživatele, ale pro pokročilé uživatele by po konfiguraci vhodná být mohla.

[5] https://tranco-list.eu



Obrázek 6.7: Snímek obrazovky pořízený při použití rozšíření *JShelter* a *NoScript* na stránce *twitter.com*. Při návštěvě nebyla volání *JavaScriptových* funkcí vůbec dostupná.

Mimo rozšíření *NoScript* prokázaly systémové testy různé úrovně odchylky referenčních běhů a běhu v kombinaci rozšíření. Při testech byl úspěšně vytvořen stejný objem výstupních informací ve formě konzolových záznamů a snímků obrazovky.

Největší rozdíl se promítnul do analýzy konzolových výstupů, ve kterých se objevovalo větší množství chybových hlášek. Zcela jistě se jedná o projev blokování obsahu rozšířeními, jak bylo popsáno v práci [38].

V případě analýzy snímků obrazovky nelze z výsledků testů tvrdit, že by byl rozdíl referenčních běhů a běhů v kombinaci zásadně pozměněn. Ve většině případů, kdy byl zaznamenán rozdíl snímků, se jednalo o nevyhnutelnou změnu působení dynamičnosti webu (dále popsáno v práci [8]).

Výsledek testu kompatibility

Testování pomocí integračních testů jasně prokázalo nekompatibilitu rozšíření *JShelter* s rozšířením *CanvasBlocker* pro prohlížeč *Mozilla Firefox* kvůli přímému konfliktu očekávaných a vypočtených hodnot při použití kombinace rozšíření *CanvasBlocker* a *JShelter* v úrovních 2 a 3. Navíc došlo k prodloužení času výpočtu atributů. Podobně v případě prohlížeče *Google Chrome* a rozšíření *Canvas Blocker* došlo ke konfliktu metody `canvas.getContext`, tudíž se u této kombinace nedá mluvit o naprosté kompatibilitě, i když použití rozšíření nemělo výrazně negativní dopad na výpočetní čas.

Systémové testování odhalilo nekompatibilitu rozšíření *NoScript* ve výchozím nastavení s rozšířením *JShelter*. V návaznosti na dokumentaci rozšíření [37] se jedná o očekávaný výsledek. Při použití jiných kombinací rozšíření popsaných v této práci nebyla systémovým testováním objevena žádná zjevná nekompatibilita ve smyslu omezení funkčnosti rozšíření *JShelter*.

Kapitola 7

Možná navázání projektu

V budoucnu může být vhodné rozšířit integrační testování o výpočet a porovnání dalších možných hodnot atributů otisku prohlížeče. Běhové prostředí je připraveno na možné rozšíření doplněnou dokumentací ve formátů `README` a komentáři zdrojového kódu implementace. Rozšiřitelná místa ve zdrojovém kódu jsou příslušně označena. Při rozšíření se lze inspirovat například technikami implementovanými v repositáři [6]. Stejně tak může dojít k rozšíření testování na další populární prohlížeče, jako jsou *Microsoft Edge*, nebo *Opera*.

Systémové testování je implementačně připraveno na podporu různých verzí prohlížeče *Google Chrome* v prostředí *Selenium Grid*. Vývojáři nástroje v roce 2023 vyjádřili zájem o implementaci mechanismu, který bude využívat potenciál nástroje *Selenium Manager* pro dynamický výběr verzí prohlížeče bez nutnosti verzi předem definovat při vytváření uzlu nástroje *Selenium Grid*[1]. Možným vylepšením systémových testů je rozhodně přidání podpory analýzy výpisů konzole pro prohlížeč *Mozilla Firefox*. V případě instalace rozšíření do prohlížeče *Mozilla Firefox* byla funkcionalita *Selenium Grid* od roku 2020, kdy původní implementace systémových testů vznikala (popsáno v práci [8]), značně vylepšena a rozšíření lze instalovat bez problému. Nicméně v otázce analýzy výpisů prohlížeče *Mozilla Firefox* je potřeba implementovat úplně novou metodu sbírání konzolových výpisů, jelikož se nelze spolehnout na funkci `get_logs` implementovanou nástrojem *Selenium* jako u prohlížeče *Google Chrome*.

V rámci běhového prostředí by bylo přínosné implementovat další lokální server, skrze který by bylo možné otestovat jiné metody ochrany při návštěvě stránky. Například omezení schopnosti třetích stran číst informace o lokálních aplikacích způsobem popsaným v článku [33]. V případě rozšíření *JShelter* by bylo v této návaznosti vhodné implementovat způsob, jak dovolit prostředí testovat funkcionalitu *Network Boundary Shield* [30]. Možným řešením může být buďto spuštění více lokálních serverů v rámci jednoho *Docker* kontejneru, nebo spouštění více různých kontejnerů, na kterých poběží vlastní lokální server.

Kontejner ve výchozí formě nepropojuje zařízení hostujícího stroje do vlastního prostředí. Při zkoumání úpravy atributů jako jsou vstupní a výstupní zařízení je tedy referenční hodnota metody `enumerateDevices` bez použití rozšíření vždy prázdná. To je dostatečné pro test schopnosti rozšíření měnit původní hodnoty atributů, ale mohlo by být zajímavé implementovat způsob spojení vstupních a výstupních zařízení hostujícího stroje a kontejneru, například pomocí přepínače `-device`[2]. Případně by mohlo být zajímavé kontejneru simulovat úplně nové periferie.

[1] https://github.com/SeleniumHQ/selenium/issues/12530

[2] https://docs.docker.com/reference/cli/docker/container/run/#device

V případě testu kompatibility v sekci 6.2 byl implementován mechanismus pro posouzení dopadu na výpočetní čas při použití různých kombinací rozšíření pomocí *API* volání `performance.now()`. V rámci práce se jednalo konkrétně o výpočet hodnot atributů elementů *canvas* a *WebGL* z důvodu překryvu metod obrany rozšíření *JShelter* a *CanvasBlocker*. V budoucnu by ale mohl být pro testujícího přínosný modul, který výpočetní náročnost zaznamená pro všechny výpočty provedené serverem a jejich hodnotu uvede jako testovací kritérium.

Nakonec by bylo vhodné adresovat jeden z problémů systémových testů. Jedná se o problém dynamičnosti webu při pořizování snímků obrazovky v rámci prohlížení webových stránek. Při návštěvě jedné stránky se může testujícímu pokaždé zobrazit jiný obsah, čímž je znehodnocena analýza dopadu použití rozšíření na navštěvovanou stránku pomocí snímků obrazovky. Rozdíl je totiž kalkulován pouze pomocí jednoho snímku bez rozšíření a jednoho snímku s rozšířením. V budoucnu by mohly systémové testy využívat mechanismus tvorby „očekávané masky“ stránky, díky které bude možné detekovat neočekávané změny při použití rozšíření. Mechanismus by se mohl inspirovat článkem [40].

Kapitola 8

Závěr

Závěrečná práce nastínila problém sledování uživatele na internetu a technik, které tvůrci sledovacích prvků stránek implementují.

Představila projekt *JShelter* společně s metodami, kterými chrání uživatele proti tvorbě otisku jeho prohlížeče při procházení internetových stránek. Popsala současnou implementaci jednotkových, integračních a systémových testů projektu a jejich nedostatky.

Soustředila se na další používaná rozšíření pro prohlížeč, která bojují za ochranu bezpečnosti a soukromí uživatele. Přednesla úvahu, jak by se mohla s rozšířením *JShelter* doplňovat a v čem by se mohla překrývat.

Udala požadavky na praktickou část diplomové práce, společně s popisem dostupných technologií, pomocí kterých lze implementaci realizovat. Shrnula cíle, kterých by měla budoucí implementace běhového prostředí pro testování rozšíření dosáhnout.

Představila návrh výsledného běhového prostředí dle zadání této práce. Při návrhu se soustředila na výběr konkrétních popsaných technologií. Navrhla možnou architekturu implementace běhového prostředí, společně s popisem významu a funkce jejích dílčích částí.

Dle návrhu bylo běhové prostředí implementováno. Struktura implementace a její zajímavé aspekty byly v práci popsány, stejně jako rozdíly mezi původní implementací integračních a systémových testů vytvořených panem Bednářem v rámci práce [8]. Popis implementace se soustředil hlavně na způsob výpočtu atributů otisku prohlížeče v případě integračních testů, doplnění původní implementace systémových testů, průběh testování a generování výstupních souborů.

Pro výslednou implementaci integračních i systémových testů běhového prostředí byly definovány podmínky, jejichž splnění je zásadní pro splnění zadání této práce. Splnění podmínek bylo v práci postupně testováno a popsáno. Při průběhu testování byly zvýrazněny zajímavé výsledky testů. Testování běhového prostředí se dále věnovalo otázce kompatibility rozšíření *JShelter* s ostatními rozšířeními popsanými v této práci. Práce definovala kompatibilitu pro oba módy testování a zhodnotila výsledky testů kompatibility jak pro dílčí části, tak pro použití kombinací rozšíření jako celku.

Nakonec práce navrhla možná pokračování a vylepšení stávající implementace společně se zdroji popisujícími momentální překážky.

Literatura

[1] *Jasmine: Simple JavaScript testing* [online]. [cit. 2023-11-4]. Dostupné z: https://jasmine.github.io/.

[2] *Smlouva u fungování Evropské Unie, Článek 16* [online]. 2012 [cit. 2023-11-20]. Dostupné z: https://eur-lex.europa.eu/legal-content/CS/TXT/PDF/?uri=CELEX:12012E/TXT.

[3] *Opinion 9/2014 on the application of Directive 2002/58/EC to device fingerprinting* [online]. 2014 [cit. 2023-11-20]. Dostupné z: https://www.dataprotection.ro/servlet/ViewDocument?id=1089.

[4] *OpenWPM* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://github.com/openwpm/OpenWPM.

[5] *Schema Documentation* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://github.com/openwpm/OpenWPM/blob/master/docs/Schema-Documentation.md.

[6] Finesse. *FingerprintJS* [online]. 2023 [cit. 2024-05-02]. Dostupné z: https://github.com/fingerprintjs/fingerprintjs.

[7] ARRIETA, A. *Privacy Badger Is Changing to Protect You Better* [online]. 2020 [cit. 2023-11-25]. Dostupné z: https://www.eff.org/deeplinks/2020/10/privacy-badger-changing-protect-you-better.

[8] Bednář, M. *Automatické testování projektu JavaScript Restrictor*. Brno, CZ, 2020. [cit. 2023-11-11]. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22376/.

[9] Chrome. *Chrome for Testing availability* [online]. 2024 [cit. 2024-01-23]. Dostupné z: https://googlechromelabs.github.io/chrome-for-testing/.

[10] Datta, A., Lu, J. a Tschantz, M. C. Evaluating Anti-Fingerprinting Privacy Enhancing Technologies. In: *The World Wide Web Conference*. New York, NY, USA: Association for Computing Machinery, 2019, s. 351–362. WWW '19. DOI: 10.1145/3308558.3313703. ISBN 9781450366748. Dostupné z: https://doi.org/10.1145/3308558.3313703.

[11] Docker. *Dockerfile reference* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://docs.docker.com/engine/reference/builder/.

[12] Docker. *Make better, secure software from the start* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://www.docker.com/.

[13] Docker Inc.. *Multi-stage* [online]. 2023 [cit. 2024-02-20]. Dostupné z: https://docs.docker.com/build/guide/multi-stage/.

[14] DuckDuckGo. *DuckDuckGo Web Tracking Protections* [online]. 2023 [cit. 2024-02-25]. Dostupné z: https://duckduckgo.com/duckduckgo-help-pages/privacy/web-tracking-protections/.

[15] Electronic Frontier Foundation. *Badger Sett* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://github.com/EFForg/badger-sett.

[16] Electronic Frontier Foundation. *EFF: The leading nonprofit defending digital privacy, free speech, and innovation.* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://www.eff.org/.

[17] Electronic Frontier Foundation. *Privacy Badger* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://privacybadger.org/.

[18] Firefox, M. *Mozilla Firefox Public Releases* [online]. 2024 [cit. 2024-01-23]. Dostupné z: https://ftp.mozilla.org/pub/firefox/releases/.

[19] Froehlich, N. *The Truth In User Privacy And Targeted Ads* [online]. 2022 [cit. 2023-11-4]. Dostupné z: https://www.forbes.com/sites/forbestechcouncil/2022/02/24/the-truth-in-user-privacy-and-targeted-ads/.

[20] Gazvoda, U. *UBlock Origin - Free, open-source ad content blocker.* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://ublockorigin.com/.

[21] Ghostery. *Ghostery: Privacy you can see* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://www.ghostery.com/.

[22] Google. *Overview of Puppeteer* [online]. 2018 [cit. 2023-12-10]. Dostupné z: https://developer.chrome.com/docs/puppeteer.

[23] Hauk, C. *What is Browser Fingerprinting? How it Works And How To Stop It* [online]. 2023 [cit. 2023-11-4]. Dostupné z: https://pixelprivacy.com/resources/browser-fingerprinting/.

[24] IBM. *What are containers?* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://www.ibm.com/topics/containers.

[25] IBM. *What is virtualization?* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://www.ibm.com/topics/virtualization.

[26] IBS. *SOCKS5 Configuration* [online]. 2021 [cit. 2024-02-03]. Dostupné z: https://www.ibm.com/docs/en/secure-proxy/6.0.2?topic=scenarios-socks5-configuration.

[27] joue.quroi. *Canvas Blocker* [online]. 2023 [cit. 2024-04-17]. Dostupné z: https://webextension.org/listing/canvas-fingerprint-blocker.html.

[28] JShelter Project. *HTML Performance* [online]. 2021 [cit. 2024-01-4]. Dostupné z: https://jshelter.org/hrt/.

[29] JShelter Project. *JavaScript Shield* [online]. 2021 [cit. 2023-11-4]. Dostupné z: https://jshelter.org/levels/.

[30] JShelter Project. *Network Boundary Shield (NBS)* [online]. 2021 [cit. 2023-11-4]. Dostupné z: https://jshelter.org/faq/#network-boundary-shield-nbs.

[31] JShelter Project. *Protection levels* [online]. 2021 [cit. 2023-11-4]. Dostupné z: https://jshelter.org/levels/.

[32] JShelter Project. *About JShelter* [online]. 2023 [cit. 2023-11-4]. Dostupné z: https://jshelter.org/.

[33] JShelter Project. *How JShelter prevents other parties from sniffing on your local applications?* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://jshelter.org/localportscanning/.

[34] kkapsner. *CanvasBlocker* [online]. 2023 [cit. 2024-04-17]. Dostupné z: https://github.com/kkapsner/CanvasBlocker.

[35] Koops, B.-J. The trouble with European data protection law. *International Data Privacy Law*. Říjen 2014, sv. 4, č. 4, s. 250–261. DOI: 10.1093/idpl/ipu023. ISSN 2044-3994. Dostupné z: https://doi.org/10.1093/idpl/ipu023.

[36] Laperdrix, P., Bielova, N., Baudry, B. a Avoine, G. Browser Fingerprinting: A Survey. *ACM Trans. Web*. New York, NY, USA: Association for Computing Machinery. apr 2020, sv. 14, č. 2. DOI: 10.1145/3386040. ISSN 1559-1131. Dostupné z: https://doi.org/10.1145/3386040.

[37] Maone, G. *NoScript: Own Your browser!* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://noscript.net/.

[38] Matoušek, P., Burgetová, I., Ryšavý, O. a Victor, M. On Reliability of JA3 Hashes for Fingerprinting Mobile Applications. In: *Digital Forensics and Cyber Crime. ICDF2C 2020*. Springer International Publishing, 2021, sv. 351, s. 1–22. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. DOI: 10.1007/978-3-030-68734-2_1. ISBN 978-3-030-68733-5. Dostupné z: https://www.fit.vut.cz/research/publication/12307.

[39] Netcraft. *Netcraft: Browser Extension* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://www.netcraft.com/apps-extensions/browser-extension/.

[40] Nikiforakis, N., Joosen, W. a Livshits, B. PriVaricator: Deceiving Fingerprinters with Little White Lies. In: *Proceedings of the 24th International Conference on World Wide Web*. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2015, s. 820–830. WWW '15. DOI: 10.1145/2736277.2741090. ISBN 9781450334693. Dostupné z: https://doi.org/10.1145/2736277.2741090.

[41] Oracle. *VirtualBox* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://www.virtualbox.org/.

[42] Petráňová, J. *Porovnání webových rozšíření zaměřených na bezpečnost a soukromí*. Brno, CZ, 2021. [cit. 2023-11-12]. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/23315/.

[43] Polčák, L., Saloň, M., Maone, G., Hranický, R. a McMahon, M. JShelter: Give Me My Browser Back. In: *Proceedings of the 20th International Conference on Security and Cryptography*. SciTePress - Science and Technology Publications, 2023, s. 287–294. DOI: 10.5220/0011965600003555. ISBN 978-989-758-666-8. Dostupné z: https://www.fit.vut.cz/research/publication/12716.

[44] Project DuckDuckGo. *Tired of being tracked online? We can help.* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://duckduckgo.com/.

[45] Rientjes, T. *Decentraleyes* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://git.synz.io/Synzvato/decentraleyes/activity.

[46] Samat, S., Acquisti, A. a Babcock, L. Raise the Curtains: The Effect of Awareness About Targeting on Consumer Attitudes and Purchase Intentions. In: *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. Santa Clara, CA: USENIX Association, červenec 2017, s. 299–319. ISBN 978-1-931971-39-3. Dostupné z: https://www.usenix.org/conference/soups2017/technical-sessions/presentation/samat-awareness.

[47] Selenium Project. *Grid* [online]. 2021 [cit. 2023-11-4]. Dostupné z: https://www.selenium.dev/documentation/grid/.

[48] Selenium Project. *WebDriver* [online]. 2021 [cit. 2023-11-4]. Dostupné z: https://www.selenium.dev/documentation/webdriver/.

[49] Selenium Project. *Selenium automates browsers. That's it!* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://www.selenium.dev.

[50] Selenium Project. *Selenium Manager (Beta)* [online]. 2023 [cit. 2023-12-10]. Dostupné z: https://www.selenium.dev/documentation/selenium_manager/.

[51] Stouffer, C. *What are data brokers? Tips to keep your data safe* [online]. 2023 [cit. 2023-11-20]. Dostupné z: https://us.norton.com/blog/privacy/data-brokers.

[52] tadatitam. *PETInspector* [online]. 2019 [cit. 2023-11-20]. Dostupné z: https://github.com/tadatitam/pet-inspector.

[53] Team Brave. *Brave, Fingerprinting, and Privacy Budgets* [online]. 2019 [cit. 2023-10-24]. Dostupné z: https://brave.com/web-standards-at-brave/2-privacy-budgets/.

[54] W3C. *Mitigating Browser Fingerprinting in Web Specifications* [online]. 2021 [cit. 2023-10-24]. Dostupné z: https://w3c.github.io/fingerprinting-guidance/.

[55] WHATWG community. *Web workers* [online]. 2023 [cit. 2024-03-14]. Dostupné z: https://html.spec.whatwg.org/multipage/workers.html.