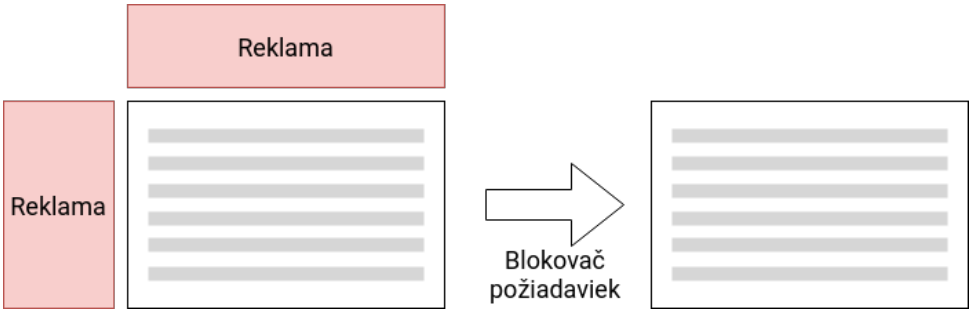
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

Blokovanie sledovacích prvkov pre prehliadače založené na WebKitGTK

Samuel Dudík*



Abstrakt

Cieľom práce je vytvoriť rozšírenie pre prehliadače založené na technológii WebKitGTK, ktoré užívateľom umožní pohodlné blokovanie reklám, sledovacích prvkov a rôznych nežiadúcich prvkov. Samotné rozšírenie je implementované v jazyku C. Na pozadí komunikuje so serverom napísanom v jazyku Rust, ktorý rozhoduje, či konkrétnu požiadavku zablokovať alebo povoliť. Server využíva knižnicu adblock-rust, ktorá bola pôvodne vytvorená pre potreby prehliadača Brave. Komunikácia medzi serverom a klientom prebieha pomocou mechanizmu Unix domain socket. Výsledkom práce je BlocKit – plnohodnotné rozšírenie určené na filtrovanie obsahu podporujúce okrem sieťového aj kozmetické filtrovanie. Súčasťou je i minimalistické GUI na jednoduchú konfiguráciu a interakciu s rozšírením.

Kľúčové slová: WebKitGTK — adblock — rozšírenie — prehliadač

Priložené materiály: Demonštračné video — Github repozitár

*xdudik01@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

Prehliadanie internetu bez blokovača požiadaviek a reklám rapídne znižuje jeho kvalitu. Užívateľ je nielen bombardovaný veľkým množstvom otravných a rušivých reklám, ale taktiež postupne prichádza o svoje súkromie, vďaka sledovacím prvkom, ktoré zaznamenávajú jeho identitu a činnosť. Z toho dôvodu je dôležité, aby mal každý užívateľ prístup k plnohodnotnému riešeniu na blokovanie požiadaviek.

Užívateľská základňa prehliadačov založených na technológii WebKitGTK je síce malá, no určite nie zanedbateľná. Medzi najpoužívanejšie prehliadače tohto typu patrí napr.:

- **Nyxt** – Prehliadač zameraný na ovládanie pomocou klávesnice implementovaný v jazyku Lisp.
- **Vimb** – Modálny prehliadač inšpirovaný textovým editorom Vim.
- **Surf** – Prehliadač vyvíjaný komunitou suckless zameraný na minimalizmus a jednoduchosť použitia.

Informácia o počte užívateľov týchto prehliadačov nie je dostupná. Približnú predstavu ale poskytuje spoločný počet hviezd na platforme Github, ktorý sa pohybuje na úrovni viac ako 9 000. WebKitGTK však trpí na nedostatok rozšírení určených na blokovanie

požiadaviek. Dostupné je len rozšírenie wyebadblock[1], ktoré si svoju úlohu plní iba čiastočne.

Práca sa zaoberá najmä problémom sieťového a kozmetického filtrovania. Podľa dokumentácie rozšírenia Adblock Plus [1] sieťové filtrovanie predstavuje blokovanie nežiadúcich požiadaviek, zatiaľ čo kozmetické sa týka blokovania samotných prvkov dokumentu. Rozšírenie by malo byť schopné vyhodnocovať oba typy filtrovania na základe dodaných filtrovacích zoznamov, ktoré podliehajú špecifickej syntaxi.

Táto práca prináša kompletné riešenie na blokovanie požiadaviek. Oproti konkurenčnému riešeniu podporuje dynamické kozmetické filtrovanie, GUI na konfiguráciu blokovača, manuálne odstránenie nežiadúcich prvkov stránky a podľa testovania v sekcii 6 je filtrovanie približne o 25 % rýchlejšie a výsledky filtrovania sa úplne zhodujú s referenčným riešením blokovacieho jadra prehliadača Brave. Keďže moje riešenie využíva knižnicu adblock-rust[2], ktorú vyvíja tím prehliadača Brave, je od neho možné očakávať rovnakú úroveň blokovania ako od najpoužívanejších rozšírení. Výsledkom je prenesenie kvalít najlepších dostupných rozšírení na platformu WebKitGTK, ktorej možnosti v tejto oblasti boli doposiaľ obmedzené.

V druhej sekcii je zhrnutá teória potrebná k pochopeniu zvyšku práce. Nasleduje sekcia o existujúcich riešeniach a ich nedostatkoch oproti môjmu riešeniu. Návrh a implementácia rozoberá vývoj rozšírenia a popis problémov, na ktoré som počas neho narazil. Posledná sekcia pred záverom sa venuje testovaniu rozšírenia s konkurenčným riešením.

2. Teória

Filtrovacie pravidlo určuje, či má byť konkrétna HTTP požiadavka alebo prvok stránky zablokovaný, resp. nezablokovaný. Dokumentácia Adblock Plus [1] ich rozdeľuje na:

- **Blokovacie filtre** – Blokujú HTTP požiadavky na sieťovej úrovni.
- **Obsahové filtre** – Umožňujú skryť HTML prvky stránky. Zaužívaný je taktiež výraz kozmetické filtre.
- **Výnimkové filtre** – Slúžia na negáciu iných filtrov, teda na povolenie požiadaviek zablokovaných blokovacím filtrom alebo na opätovné zobrazenie prvkov stránky, ktoré boli skryté obsahovým filtrom. Za výnimkové filtre sa označujú aj blokovacie a obsahové filtre, ktoré negujú efekt iných filtrov.

[1] https://github.com/jun7/wyebadblock
[2] https://github.com/brave/adblock-rust

Pravidlá sú vždy jednoriadkové. Môžu obsahovať skupinu možností, ktoré pravidlo napr. obmedzia na špecifické domény, typ požiadavky atď. Na obrázku 1 možno vidieť úryvok filtrovacieho zoznamu zobrazeného v prehliadači zoznamov rozšírenia uBlock Origin.

```
/googleadmanager-
/googleadpush.
/googleadright.
/googleads-$domain=~github.com
/googleads.$~script,domain=~googleads.github.io
/googleads/*$domain=~github.com
##adswidget4-quick-adsense-reloaded-3
##adswizzBanner
##adsxpls2
##adszed-728x90
##adtab
```

Obrázok 1. Časť jedného z najpoužívanejších filtrovacích zoznamov – EasyList. 2 mriežky na začiatku pravidla označujú obsahové filtre. Pravidlá bez mriežok sú naopak blokovacie filtre. Symbol doláru označuje začiatok možností pravidla.

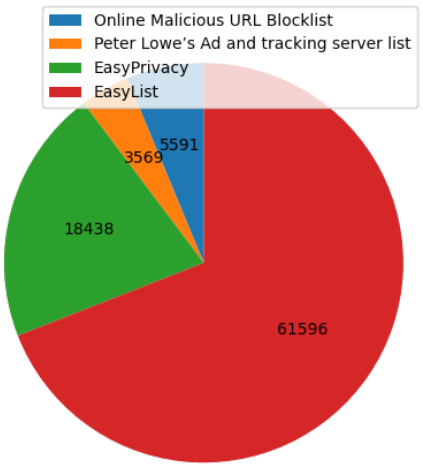
Filtrovací zoznam je zoznam filtrovacích pravidiel. Filtrovacie zoznamy nepodliehajú žiadnemu oficiálnemu deleniu. Väčšinou sú zamerané na určitú oblasť, napr. geografický región či konkrétna stránka/spoločnosť. Existujú aj také zoznamy, ktoré sa snažia byť univerzálne a pokryť čo najväčšie množstvo generických reklám či sledovacích prvkov.

3. Existujúce riešenia

Užívatelia sú aktuálne odkázaní na 2 riešenia: Content Blocker API alebo rozšírenie wyebadblock.

Content Blocker API je rozhranie umožňujúce aplikáciu filtrovacích pravidiel uložených v špeciálnom deklaratívnom formáte v snahe filtrovanie urýchliť [2]. Nevýhod je v tomto prípade hneď niekoľko. Počet aktívnych filtrov je limitovaný na 50 000, čo sa na prvý pohľad môže javiť ako dostatočný počet, no na základe grafu 2 je možné dedukovať, že pri bežnom použití je počet aktívnych filtrov vyšší o viac ako polovicu. Druhým problémom je nekompatibilita spomínanej špeciálnej syntaxe so zaužívanou Adblock Plus filter syntaxou, ktorá je de facto štandardom. Na využitie rozhrania Content Blocker je potrebný prístup k objektu WebView, ktorý ale z rozšírenia dostupný nie je. Z toho dôvodu je použitie tohto rozhrania limitované na implementáciu priamo v samotnom prehliadači, čím prichádzame o všetky benefity univerzálneho rozšírenia.

Jediné dostupné rozšírenie na blokovanie požiadaviek wyebadblock je na tom s použiteľnosťou o niečo lepšie, no stále nedosahuje požadovaných kvalít. Podporuje zaužívanú syntax, no pri testovaní dosahuje len



Obrázok 2. Graf reprezentujúci počet filtrovacích pravidiel v jednotlivých zoznamoch. Vybrané zoznamy verne reprezentujú aktívne zoznamy bežného užívateľa, keďže sú automaticky aktivované po novej inštalácii jedného z najpoužívanejších rozšírení na blokovanie požiadaviek – uBlock Origin.

62 % zhodu s referenčným blokovačom od spoločnosti Brave. Okrem toho vo svojom README [3] uvádza, že nepodporuje dynamické kozmetické filtrovanie a umožňuje len vygenerovanie statického zoznamu pravidiel CSS, ktorý je následne nutné manuálne nastaviť, čo podporujú len niektoré prehliadače. Statické blokovanie nie je schopné zohľadniť prvky nachádzajúce sa na aktuálne navštívenej stránke, a preto sú mnohé nežiadúce prvky prehliadnuté.

Je nutné taktiež spomenúť prehliadač Brave, ktorého knižnicu na filtrovanie požiadaviek využíva riešenie tejto práce. Ide o prehliadač založený na prehliadači Chromium[3]. Od konkurencie sa ale odlišuje tým, že jeho predvolené nastavenie automaticky blokujú reklamy a sledovacie prvky, ako uvádza na svojej oficiálnej stránke [4]. Knižnica, ktorá túto funkcionalitu umožňuje, je voľne dostupná pod názvom adblock-rust[4]. Ako z názvu vyplýva, je napísaná v programovacom jazyku Rust, čo zaručuje rýchlosť a bezpečnosť kódu. Vďaka práci tímu stojaceho za prehliadačom Brave, možno od rozšírenia očakávať väčšiu robustnosť, ako keby za blokovacím jadrom stála jedna osoba. Samotné rozšírenie sa môže následne zamerať na prívetivosť užívateľského rozhrania a na uľahčenie bežných úkonov pri interakcii s blokovačom. Existencia serveru obchádza obmedzenia WebKitGTK Webextension API, vďaka čomu šetrí prostriedky operačného systému. Server taktiež umožňuje iným vývojárom pristupovať ku knižnici adblock-rust v ľubovoľnom jazyku, čo znova zvyšuje potenciálnu adopciu alternatívnych prehliadačov, ktoré ešte nedisponujú rozšírením na filtrovanie obsahu.

[3] https://www.chromium.org/
[4] https://github.com/brave/adblock-rust

4. Návrh

Pôvodným zámerom práce bolo implementovať samotnú blokovaciu logiku. Inšpiráciu som čerpal z prístupu zaužívaného rozšírenia uBlock Origin na základe príspevkov od spoločností Cliqz [5] a Brave Software [6]. Každé pravidlo z dodaných filtrovacích zoznamov je najprv rozdelené na tzv. tokeny. Ide o alfanumerické časti pravidla oddelené separátormi (napr. bodka, lomka atď.). Pre každý token je následne určený jeho výskyt vo všetkých dodaných zoznamoch. Na základe tohto histogramu je každé pravidlo indexované tokenom, ktorý sa v ostatných pravidlách nachádza čo najmenej. Výsledkom je hašovacia tabuľka obsahujúca polia pravidiel indexované najviac diskriminatívnymi tokenmi.

Rozhodovanie o zablokovaní, resp. nezablokovaní požiadavky spočíva v nájdení pravidla, ktoré sa zhoduje s adresou a typom požiadavky v hašovacej tabuľke. Pri nájdení zhody požiadavka vôbec nie je odoslaná na server, čo napomáha rýchlosti prehliadania. URL adresa požiadavky je rovnako ako pri spracovaní zoznamov rozdelená na tokeny. Pre každý token je v hašovacej tabuľke vyhľadané pole pravidiel. Pravidlá sú postupne testované na zhodu. Ak je nájdená zhoda, celé vyhodnocovanie je ukončené. Pred vrátením výsledku ešte prebehne kontrola na zhodu požiadavky s nejakým výnimkovým pravidlom. To prebieha takmer totožne ako hľadanie zhody s blokovacími pravidlami, rozdiel je v opačnom efekte pri zhode.

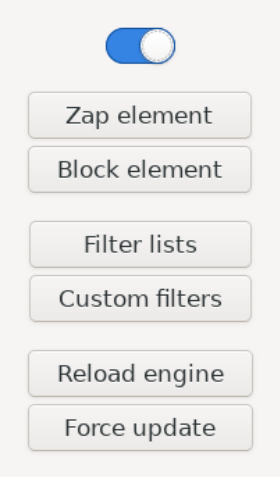
Pri finalizovaní filtrovacej logiky som ale narazil na problém s tým, ako WebKitGTK využíva dostupné prostriedky operačného systému. Pri otvorení nového okna prehliadača sa vytvorí nový proces WebView, ktorý dynamicky načíta dodané rozšírenia. Každé okno teda pracuje s izolovaným procesom rozšírenia, ktorý pri spustení musí opakovane spracovať filtrovacie zoznamy a následne ich uložiť do RAM na neskoršie použitie. Pri väčšom počte okien prehliadača môže ísť o pomerne veľkú stratu pamäte.

5. Implementácia

Na riešenie problému s neefektívnym využitím pamäte som podobne ako konkurenčné riešenie wyebadblock, pristúpil k architektúre klient-server. Klienti (jednotlivé okná prehliadača) sa serveru, ktorý má v pamäti uložené spracované filtrovacie zoznamy, dopytujú, či je konkrétna požiadavka nežiadúca. Na samotnú komunikáciu bol použitý mechanizmus Unix domain socket,

ktorý podporuje plný duplex prenos, ktorý je potrebný na správne fungovanie blokovača. Takáto architektúra, resp. akékoľvek použitie medziprocesovej komunikácie, odstraňuje nutnosť blokovacie jadro implementovať v jazyku C. Novonadobudnutá voľnosť upriamila moju pozornosť na jedinú dostupnú knižnicu na blokovanie požiadaviek adblock-rust. Bez nutnosti implementovať blokovaciu logiku, som sa mohol zamerať na užívateľskú prívetivosť a jednoduchosť použitia.

5.1 Užívateľské rozhranie



Obrázok 3. Užívateľské rozhranie pozostáva len z jedného okna.

Užívateľské rozhranie rozšírenia sa snaží byť čo najjednoduchšie na použitie. Pokročilé funkcie sú dostupné z prehľadného grafického užívateľského rozhrania, no hlavnú funkcionalitu je možné nakonfigurovať úpravou textových súborov. To ocenia najmä užívatelia alternatívnych prehliadačov, ktorí uprednostňujú jednoduchý software umožňujúci prepojenie s inými programami, resp. skriptami. Napríklad, za prehliadačom surf stojí zoskupenie suckless, ktoré vo svojom manifeste [7] vyzýva vývojárov k tvorbe minimálneho softvéru. Samotný prehliadač sa počtom riadkov kódu pohybuje na hranici do 2 200.

GUI rozšírenia, ktoré je možné vidieť na obrázku 3, pozostáva len zo 6 tlačidiel a jedného spínača:

- **Spínač** – Zapnutie resp. vypnutie blokovania požiadaviek a prvkov.
- **„Zap element“** – Dočasné manuálne odstránenie prvku stránky. Termín „zap“ je zaužívaný u populárneho rozšírenia uBlock Origin.
- **„Block element“** – Permanentné odstránenie prvku stránky (vytvorí filtrovacie pravidlo).
- **„Filter lists“** – Otvorenie súboru obsahujúceho zoznam aktívnych filtrovacích zoznamov v predvolenom textovom editore.
- **„Custom filters“** – Otvorenie súboru obsahujúceho vlastné filtrovacie pravidlá v predvolenom textovom editore.
- **„Reload engine“** – Reštartovanie blokovacieho jadra za účelom aplikovania zmien vykonaných v súbore so zoznamom aktívnych filtrovacích pravidiel alebo v súbore s vlastnými filtrovacími pravidlami.
- **„Force update“** – Vynútené aktualizovanie všetkých aktívnych filtrovacích zoznamov na najnovšiu verziu a následné rovnaké reštartovanie ako pri aktivovaní tlačidla „Reload engine“.

Tlačidlá na odstránenie prvkov do webovej stránky aktivujú kód v jazyku Javascript, ktorý vizuálne označuje práve vybraný prvok (možno vidieť na obrázku 4). Po kliknutí je prvok odstránený a prípadne je vygenerované a uložené pravidlo na permanentné zablokovanie prvku.

Obrázok 4. Ukážka módu na vyberanie prvku stránky určeného na odstránenie/zablokovanie.

Po viacerých pokusoch implementovať vhodné rozhranie na úpravu filtrovacích zoznamov a vlastných filtrov, som sa nakoniec rozhodol pre jednoduché tlačidlá, ktoré po kliknutí otvoria príslušný súbor v predvolenom textovom editore. Bolo by zbytočné znovu navrhovať rozhranie, ktoré sa vo výsledku snaží replikovať funkcionalitu obyčajného textového editora. Tento prístup je taktiež viac v súlade s filozofiou typického užívateľa alternatívnych prehliadačov a umožňuje vytvorenie vlastného pracovného postupu nad týmito súbormi pomocou skriptov.

6. Testovanie

Testovanie rozšírenia sa zaoberá porovnaním rozsahu a správnosti filtrovania oproti konkurenčnému riešeniu wyebadblock. Ako referenčné riešenie bola použitá knižnica adblock-rust, ktorá narozdiel od iných rozšírení umožňuje jednoduchý prístup k blokovaciemu jadru. Keďže novovytvorené rozšírenie využíva na filtrovanie práve túto knižnicu, očakáva sa od neho 100 % zhoda vo výsledkoch testovania.

Tabuľka 1. Výsledky testovanie pre rozšírenie wyebadblock.

	Zhoda	Nezhoda	False positive	False negative
EasyList a EasyPrivacy	159 322	83 623	4 947	78 676
Všetky zoznamy	155 190	87 755	5 121	82 634

Tabuľka 2. Výsledky testovanie pre rozšírenie BlocKit.

	Zhoda	Nezhoda	False positive	False negative
EasyList a EasyPrivacy	242 945	0	0	0
Všetky zoznamy	242 945	0	0	0

Ako dataset simulujúci reálne použitie prehliadača bol využitý zoznam 242 944 požiadaviek vytvorený spoločnosťou Cliqz za účelom vlastného testovania [8]. Vznikol navštívením 500 najnavštevovanejších domén (podľa ich vlastného vyhľadávača) a ich 3 náhodne vybraných podstránok.

Dataset bolo pred použitím nutné upraviť do požadovaného textového formátu a ku každej požiadavke bola pripojená informácia o tom, či ju adblock-rust považuje za nežiadúcu.

Testovanie rozšírení na blokovanie požiadaviek, najmä na rôznych prehliadačoch, je problematické. Dokumentácia aplikačného rámca Selenium [9], o ktorého použití som uvažoval, uvádza, že ho ovplyvňujú viaceré faktory ako napr. rýchlosť spustenia prehliadača, rýchlosť odozvy zo serverov, na ktoré sa prehliadač pripája atď. V snahe vytvoriť testovacie podmienky s najmenším množstvom nežiadúcich vplyvov, bolo nutné pracovať priamo s blokovacími jadrami rozšírení.

Testovací program prejde celým súborom a pre každú požiadavku vyhodnotí reakciu môjho rozšírenia BlocKit a wyebadblock. Výsledok je porovnaný s reakciou knižnice adblock-rust a na konci programu je vypísaná prehľadná štatistika.

Testovanie bolo rozdelené na 2 časti, pričom v prvej z nich sú rozšírenia testované len na filtrovacích zoznamoch EasyList a EasyPrivacy. Druhý test už prebieha s využitím všetkých bežných filtrovacích zoznamov. Dôvodom je snaha poukázať na rozdiely v testovaní s menším počtom filtrovacích zoznamov. Zoznamy EasyList a EastPrivacy spoločne predstavujú určitý zlatý štandard filtrovacích zoznamov, keďže sami o sebe pokrývajú pomerne veľký počet bežných reklám a sledovacích prvkov.

Z výsledkov testovania v tabuľke 1 je možne dedukovať, že moje rozšírenie BlocKit dosahuje očakávanú 100 % zhodu s referenčným riešením adblock-rust. Konkurenčné riešenie wyebadblock podľa tabuľky 2 pri testovaní s oboma zoznamami správne odhadlo len 65.5 % požiadaviek. Nesprávne odhady sú z 94 % tvorené označením nežiadúcej požiadavky ako neutrálnej. Záverom tejto časti testovania je fakt, že rozšírenie BlocKit vracia 1,52 krát viac správnych odhadov ako konkurenčné riešenie wyebadblock, pričom dosahuje 100 % zhodu s referenčným riešením adblock-rust.

6.1 Komunikácia s užívateľmi

Súčasťou testovania je i odozva od užívateľov. Ešte pred započatím práce na rozšírení som sa formou platformy Github, zapojil do diskusie o možnosti pokročilého blokovania požiadaviek v jednom z najpoužívanejších prehliadačov založených na WebKitGTK – Nyxt[5]. Užívateľov som po vydaní prvej verzie rozšírenia opäť kontaktoval s prosbou o spätnú väzbu a o nahlásenie akýchkoľvek chýb. Podobným spôsobom som pokračoval aj na sociálnej sieti Reddit, kde som rozšírenie publikoval do skupín zameraných na Unixové systémy a prehliadače založené na technológii WebKitGTK. Výsledkom je viac ako 130 stiahnutí serveru, 3 opravené chyby/pridané funkcie a niekoľko desiatok hviezd na platforme Github. V tejto komunikácii plánujem pokračovať, aby sa pod vplyvom spätnej väzby mohlo rozšírenie stať niečím, čo užívatelia skutočne chcú používať.

7. Záver

Cieľom práce bolo vytvoriť rozšírenie pre prehliadače založené na technológii WebKitGTK, ktoré predstavuje kompletné riešenie oproti konkurenčnému rozšíreniu wyebadblock. Súčasťou rozšírenia je podpora sieťového a dynamického kozmetického filtrovania, automatické aktualizovanie filtrovacích zoznamov a prehľadné GUI na úpravu nastavení a prípadné manuálne blokovanie prvkov stránky.

Rozšírenie v testovaní vykazuje o 38 % väčší počet správnych odhadov oproti rozšíreniu wyebadblock. Taktiež dosahuje priemerne o štvrtinu rýchlejší čas vyhodnotenia zhluku požiadaviek popísaných v sekcii 6.

Rozšírenie je dostupné na platforme Github[6] spolu s návodom ako ho nainštalovať a používať. Už na

[5] https://github.com/atlas-engineer/nyxt
[6] https://github.com/dudik/blockit

počiatku tvorby rozšírenia bol udržiavaný kontakt s užívateľmi prehliadačov WebKitGTK.

Na rozšírení pracujem i naďalej. Plánujem sa sústrediť na požiadavky a návrhy užívateľov. Okrem toho je taktiež nutné mierne refaktorovanie kódu za účelom zjednodušiť budúce úpravy a opravy.

Poďakovanie

Chcel by som sa poďakovať vedúcemu mojej práce, Ing. Liborovi Polčákovi, Ph.D., za umožnenie pracovať na vlastnom zadaní a za prínosné konzultácie.

Literatúra

[1] eyeo GmbH. How to write filters. [online]. [cit. 2021-03-25]. Dostupné z: https://help.eyeo.com/adblockplus/how-to-write-filters.

[2] Poulain B. Introduction to webkit content blockers. [online], 6 2015. [cit. 2021-03-25]. Dostupné z: https://webkit.org/blog/3476/content-blockers-first-look/.

[3] jun7. wyebadblock. [online], 7 2017. Revidované 7. 5. 2020. [cit. 2021-03-25]. Dostupné z: https://github.com/jun7/wyebadblock#element-hiding.

[4] Brave Software Inc. Brave browser features. [online]. [cit. 2021-03-27]. Dostupné z: https://brave.com/features/.

[5] Cliqz GmbH. Not all adblockers are born equal. [online], 12 2019. [cit. 2021-03-26]. Dostupné z: https://0x65.dev/blog/2019-12-20/not-all-adblockers-are-born-equal.html.

[6] Brave Software Inc. Brave improves its ad-blocker performance by 69x with new engine implementation in rust. [online], 6 2019. [cit. 2021-03-26]. Dostupné z: https://brave.com/improved-ad-blocker-performance/.

[7] suckless.org community. Software philosophy. [online]. [cit. 2021-03-26]. Dostupné z: https://suckless.org/philosophy/.

[8] Cliqz GmbH. Adblockers performance study. [online], 2 2019. [cit. 2021-03-27]. Dostupné z: https://whotracks.me/blog/adblockers_performance_study.html.

[9] Software Freedom Conservancy. Selenium performance testing. [online]. Revidované 22. 4. 2021. [cit. 2021-03-27]. Dostupné z: https://www.selenium.dev/documentation/en/worst_practices/performance_testing/.