МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Магістерської та аспірантської підготовки
Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Дослідження браузерів на наявність вразливостей і розробка програмного забезпечення для їх усування»

Виконав студент 2 курсу групи МК- 61
спеціальності 122 Комп'ютерні науки та інформаційні технології
Гліклiх Артур Мечиславович
Керівник доцент кафедри інформаційних технологій к.т.н.
Онищенко Сергій Михайлович
Консультант _____
Рецензент доцент начальника НДЧ к.геогр.н. Лужбін Анатолій Михайлович

Одеса 2018

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

EDP – парадигма програмування, в якій виконання програми визначається подіями (Event-driven programming).

FIFO – спосіб організації і маніпулювання даними щодо часу і пріоритетів (First In, First Out).

GC – одна з форм автоматичного управління пам'яттю (Garbage collection).

IPC – обмін даними між потоками одного або різних процесів (Inter-process communication).

JIT – технологія збільшення продуктивності програмних систем (Just-in-time).

OAuth – відкритий стандарт авторизації (Open Authorization).

SOP – це важлива концепція безпеки для деяких мов програмування на стороні клієнта (Same Origin Policy).

OAuth – відкритий стандарт авторизації (Open Authorization).

URI – компактний рядок літер, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс (Uniform Resource Identifier).

URL – стандартизована адреса певного в інтернеті (Uniform Resource Locator).

ВСТУП

На сьогоднішній день більшість користується браузерами, але мало хто знає, як уникнути злом и, зчитування приватних даних. Дана тема є актуальною, так як захисники браузерів не справляються самостійно і завжди потрібен пильний контроль користувача.

Для виконання даної роботи був проведений аналіз браузерів в мережі інтернет. Були знайдені такі проблеми як зчитування даних через файли Cookie, збережені паролі і історію браузера. Актуальність використання режиму "Інкогніто". Для розробки програми необхідно було перевірити програми вже існуючі, щоб знайти їх уразливості, проблеми то, що допоможе поліпшити захист браузера.

Метою і завданням даної роботи є знайти актуальні проблеми і уразливості браузера. Розглянути такі поняття як Атаки на загальних циклах подій і Аналіз безпеки політики правління ресурсами розширень. Створення програмного забезпечення для усунення вразливостей на основі готових програм, які надають захист браузера.

1 АНАЛІЗ МЕТОДІВ ВИКОРИСТАННЯ ВРАЗЛИВОСТЕЙ БРАУЗЕРА

1.1 Розширення браузера

Розширення браузера - найпопулярніший метод в даний час доступний для розширення функціональних можливостей сучасних веб-браузерів. Розширення існують для більшої частини браузерів сімей, включаючи основні веб-браузери, такі як Firefox, Chrome, Safari і Opera. Їх можна легко завантажити і встановлюються користувачами з центрального сховища (наприклад, Інтернет-магазин Chrome або Firefox Add Ons). На жаль, розширення також схильні до неправильного використання. В факт, через їх тісному зв'язку з середовищем браузера, вони можуть бути ображені противником, щоб для збору широкого спектра приватної інформації - таких як файли cookie, історію переглядів, дані на системному рівні або навіть паролі. У зв'язку з цим обсяг досліджень, які вивчають наслідки для безпеки і уразливості розширень браузера швидко збільшилися в останні роки [1].

Коли браузери були вперше введені, веб-сайти змогли отримати доступ до всіх своїх місцевих ресурсів. Як Наслідки, зловмисні актори почали використовувати цей вільний доступ даних для перерахування розширень, які користувач має встановлених в її системі або навіть для використання вразливостей в межах встановлених розширень. Щоб пом'якшити це збільшення загроза, Firefox представив доступний контент прапор і Chrome нову версію маніфесту для реалізації деяка форма контролю доступу до ресурсів розширення. В іншій частині статті ми будемо посилатися на ці заходи безпеки в якості параметрів контролю доступу. Розробники Сафарі вирішив прийняти інший механізм, який складається в рандомізації під час виконання частини розширення URI. Ми будемо посилатися на цей другий клас захисту як рандомізація URI. Інформація веб-браузера була використана для кількість зловмисних або «сумнівних» цілей. Для Наприклад, Panopticlick створює унікальний відбиток для браузера використовуючи встановлені шрифти, серед інших функцій. PluginDetect витягує замість цього список встановлених плагінів в браузері. Гірше того, цей метод недавно був використаний в двох повідомленнях, шкідливих кампаній . Завдяки існуючим контрзаходів безпеки браузера описані вище, до сих пір розширення були захищені проти цих методів відбитків пальців. Два дуже простих недавно були запропоновані перерахування для вилучення невелика кількість

встановлених розширень в браузерах, які прийняті настройки контролю доступу. Ці методи скористалися доступними ресурсами розширень присутнім в Chrome і Firefox для визначення невеликого числа популярних розширень. Крім того, XHOUND також недавно було запропоновано перерахувати розширення і виконувати відбитки пальців, вимірюючи зміни в DOM веб-сайту. У цій статті ми представляємо першу поглиблену захист вивчення всіх політик управління ресурсами розширень використовуваних сучасними браузерами. Наш аналіз показує, що все сімейств браузерів, які в даний час підтримують розширення, є вразливі для будь-якої форми перерахування. Зокрема, в той час як два варіанти дизайну (тобто контроль доступу настройки або рандомизация URI) захищені від теоретична точка зору, їх практична реалізація страждає від безлічі різних проблем. Ми обговорюємо дві наступальні методи, щоб підірвати ці політик управління, один з яких заснований на тимчасовій бічній атаці і один, заснований на мимовільної витоку випадкового URI, який впливає на багато розширення. У той час письма, ці атаки підривають безпеку всіх браузерів. Ми також обговорюємо набір заснованих атак по цим методам, які дозволяють третім сторонам виконувати точна за відбитками пальців або виконання різних типів цілеспрямованих атак, виконуючи тести на доказ концепції деякі з них. Ми вже повідомляли про виявлені проблеми залучені розробники браузерів і розширень, і ми в даний час обговорюють з ними можливі виправлення. Таким чином, в цьому документі зроблено такі вклади:? Ми пропонуємо перший тимчасовий перерозподіл розширень атака, яка може отримати повний список розширень встановлених в браузерах, які використовують управління доступом Налаштування. Цей метод в значній мірі перевершує будь-які представлена раніше методологія відбитки пальців зустрітися. ? Ми розробляємо інструмент статичного аналізу для розширень Safari, і використовувати його для позначення сотень потенційно вразливі випадки, в яких розробники просочилися URI випадкового розширення. Через вичерпний аналіз ручного коду на підмножині розширень, ми підтверджуємо, що це дійсно дуже поширене проблема, яка зачіпає значну частину всіх розширень Safari. ? Ми показуємо, що контроль ресурсів розширення браузерів політики дуже складно правильно спроектувати і реалізувати, і вони схильні до тонким помилок, які підривають їх безпеку. Наші дослідження привели до численних обговорення з розробниками всіх основних браузерів і розширень, в тому числі вразливих до нашим атакам і тим, які все ще знаходяться в проектування або тестування.

Всі браузери, які підтримують розширення, реалізують деякі форми захисту для запобігання довільних веб-сайтів від перерахування встановлених розширень і вільний доступ їх ресурсів. Після великого огляду декількох традиційних і сімейства мобільних браузерів ми визначили два основні класи механізмів захисту, які в даний час використовуються: параметри доступу в приміщення і рандомізація URI.

1.2 Управління подіями програмування (EDP)

Управління подіями програмування (EDP) складається з визначення відповіді на такі події, як дії користувача, сигнали введення / виведення, або повідомлення з інших програм. EDP є переважаючим парадигми програмування для графічних призначених для користувача інтерфейсів, веб-клієнтів, і це швидко набуває серверне та мережеве програмування. Наприклад, Стандарт HTML5 передбачає, що призначені для користувача агенти будуть реалізовані використовуючи EDP, аналогічно, Node.js, memcached і Nginx, також покладаються на EDP. У EDP кожна програма має цикл подій, який складається з черги FIFO і процесу управління (або потоку) який прослуховує події. Події, які прибувають, черги послідовно відправляються елементам управління відповідно до політики FIFO. Ключова особливість EDP полягає в тому, що операції з високою затримкою (або блокуванням) такі як бази даних або мережеві запити, можуть оброблятися асинхронно: вони з'являються в черзі тільки як запуск і завершення сигналізації подій, тоді як блокування самої операції обробляється в іншому місці. В цьому випадку EDP забезпечує чуйність і дрібнозернистий паралелізм необхідний для сучасного інтерфейсу користувача і мережі серверів, без обтяження програмістів з явним контролем паралелізму.

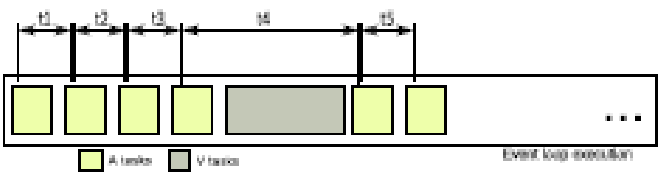

Рисунок 1.1 – Загальний цикл подій. Безліч коротких замикань задає і записує час обробки кожного з них. Різниця в часі між двома послідовними завдання показують, що V поставив завдання між ними і як довго вони виконувалися

Ми показуємо, що системи на основі EDP сприйнятливі для побічних атак. Ці цикли подій утворюють ресурс, який можна розділити між взаємно не довіряти програмами. Отже, твердження цього ресурсу однієї програми можна спостерігати інші за допомогою зміни в часі процесу контролю приймаємо для відправки їх подій. рис. 1.1 ілюструє такий сценарій для циклу, який спільно використовується зловмисником A і жертвою B.

Напади, засновані на спостережуваному порушення загальних ресурсів мають довгу історію [11] і активний подарунок; проте атаки проти загальних циклів подій досі розглядалися тільки з теоретичної точки [10]. Тут ми виконуємо перші атаки проти реальних EDP-системи. Зокрема, ми спрямовуємо публічну подію циклів в двох центральних процесах Google Chrome веб-браузер: хост-процес, чий цикл подій між усіма запитами про загальні ресурсах, такими як мережевий і призначений для користувача інтерфейс; і процеси рендеринга, чиї петлі можуть бути розділені між Javascript завданнями різних вкладок або iframe. Ми будуємо інфраструктуру, яка дозволяє нам стежити за шпалерами петлями з шкідливою HTML-сторінки. Це полегшується за моделлю асинхронного програмування, що використовується в обох Chrome і Javascript. Асинхронні виклики функцій запускають нові завдання, які додаються в одну чергу, в контрастують з синхронними викликами, які просто штовхаються в стек викликів поточного завдання і виконується без preemption, блокуючи цикл.

Для циклу подій рендеринга ми покладаємося на API PostMessage, який є функцією Javascript для міжмережевий зв'язку на основі асинхронного.

Зворотні виклики. Відправляючи повідомлення собі ми можемо контролювати цикл подій з дозволом від 25 мс, причому тільки одну задачу в циклі на кожен момент часу.

Для циклу подій хост-процесу ми покладаємося на два різні механізми: мережеві запити до незворотних IP-адреси, які входять в цикл і дуже швидке переривання, забезпечуючи дозвіл 500 мс; і SharedWorkers, чиї повідомлення проходять через цикл подій хост-процесу, що забезпечує дозвіл 100 мс. Ми використовуємо інформацію, отриману з використанням цих методів в трьох різних атаках:

1) Ми показуємо, як затримки подій під час завантаження фаза, що відповідає запитам ресурсів, розбір, рендеринга і Javascript, можна використовувати для однозначно ідентифікувати веб-сторінку. рис. 1.2 візуалізує це з використанням трьох представницьких веб-сторінок. Хоча це атака розділяє мету з атакою Memento, канали абсолютно різні: по-перше, на

відміну від Memento, ми знаходимо, що відносне впорядкування подій необхідної для успішної класифікації, що мотивує використання динамічного перетворення часу в якості міри відстані. По-друге, ми показуємо, що ідентифікація сторінки через цикл подій вимагає тільки мінімального навчання: ми досягаємо рівень визнання до 75% і 23% для заходу циклів процесу візуалізації і хоста, відповідно, для 500 головних сторінок від кращих сайтів Alexa. Ці ставки виходять з використанням тільки одного зразка кожної сторінки для етап навчання.

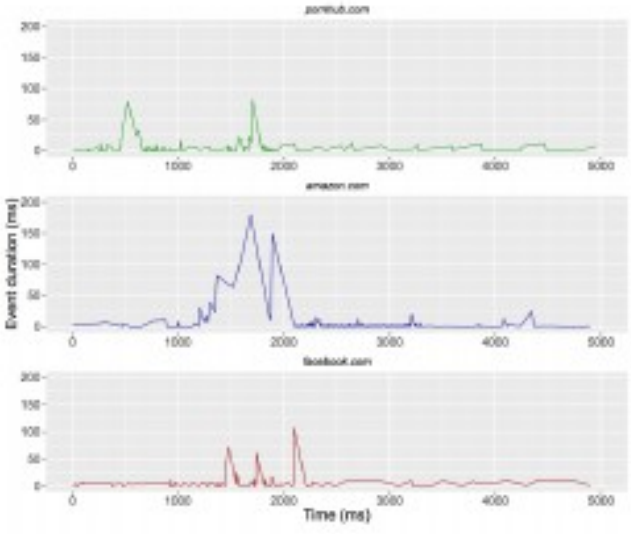

Рисунок 1.2 – Затримки, які спостерігаються при завантаженні іншої мережі сторінок на вкладці зловмисника, спільно використовує процес рендеринга

2) Ми проілюструємо, як дії користувача на сторінках крос-оригіналу можуть бути виявлені на основі затримок, які вони вводять в цикл подій. Зокрема, ми встановлюємо атаку проти Реєстраційні форми Google OAuth, в яких ми вимірюємо час між натисканнями клавіш, коли користувач вводить пароль. Вимірювання часу, які ми отримуємо з циклу подій значно менше шуму або вимагає менших привілеїв ніж з інших каналів.

Горизонтальна вісь зображує минуле реальний час, вертикальна вісь відображає час, витрачений циклом події для обробки завдання, вставлені зловмисником. Всі сторінки чітко помітні, як людським оком, так і класифікацією методів.

3) Ми демонструємо, що загальні цикли подій можуть використовуватися для передачі інформації між перехресним походженням сторінок. Зокрема, ми реалізуємо прихований канал з пропускна здатність

200 біт / с через головний цикл подій потоку, а інший – робочі перехресні процеси 5 біт / с. Наші атаки показують, що цикли подій можуть бути успішно шпигувати навіть за допомогою простих засобів. Вони працюють під припущення, що цикли подій поводяться як черги FIFO; в реальність, однак, цикл подій Chrome має більш складну структуру, покладаючись на кілька черг і політичний планувальник. Ми вважаємо, що цю структуру можна використовувати для більш потужних атак в майбутньому.

2 LOOPHOLE: ТИМЧАСОВІ АТАКИ НА ЗАГАЛЬНИХ ЦИКЛАХ ПОДІЙ В CHROME

Програмування на основі подій (EDP) є поширеною парадигмою для графічних призначених для користувача інтерфейсів, веб-клієнтів і воно швидко набуває важливість для серверної та мережевої програмування. Центральними компонентами EDP є цикли подій, які діють як черги FIFO, які використовують процеси для зберігання і відправки повідомлень, отриманих з інших процесів. Ми демонструємо, що загальні цикли подій уразливі для побічних атак, коли шпигунські процеси контролюють схему використання циклів інших процесів шляхом реєстрації подій і вимірювання часу, який потрібен для їх відправки. Зокрема, ми демонструємо атаки проти двох центральних циклів подій в Chrome від Google веб-браузера: потік введення-виведення хост-процесу, який мультиплексирует всі мережеві події і дії користувача, і основний потік процесів рендеринга, який обробляє операції рендеринга і Javascript. Для кожного з цих циклів ми показуємо, як шаблон використання можуть контролюватися з високою роздільною здатністю і низькими накладними витратами, і як цим можна зловживати для зловмисних цілей, таких як ідентифікація веб-сторінки, виявлення поведінки користувача, і таємного спілкування.

2.1 Політика ізоляції і спільне використання подій Петлі в Chrome

У цьому розділі ми переглядаємо одну і ту ж політику походження і її варіанти. Потім ми обговоримо взаємозв'язок цих політик з архітектурою Chrome, де ми поміщаємо спеціальний зосередьтеся на способах спільного використання циклів подій.

2.1.1 Однакова політика походження

Політика одного і того ж походження (SOP) є центральною концепцією в моделе веб-безпеки: політика обмежує скрипти на веб-сторінку, щоб отримати доступ до даних з іншої сторінки, якщо їх походження відрізняються. Дві сторінки мають однакове походження, якщо протокол, порт і хост рівні. Попит на гнучку перехресну зв'язок викликало впровадження таких функцій, як домен релаксація, API PostMessage, ресурс

Cross-origin Спільне використання (CORS), Обмін повідомленнями каналу, Suborigins або Fetch API. Ця функція повзучості включає збільшення складності браузера і поверхню атаки, яка мотивувала постачальникам браузерів рухатися до більш надійним багатопроцесорним архитектурам.

2.1.2 Огляд архітектури Chrome

Архітектура Chrome розділена на різні операційні системних процесів. Обгрунтування цієї сегментації двійковій: ізолювати веб-контент від хоста, а також для підтримки політики примусового здійснення походження за допомогою ОС. Для досягнення цієї сегментації, Chrome використовує два процеси:

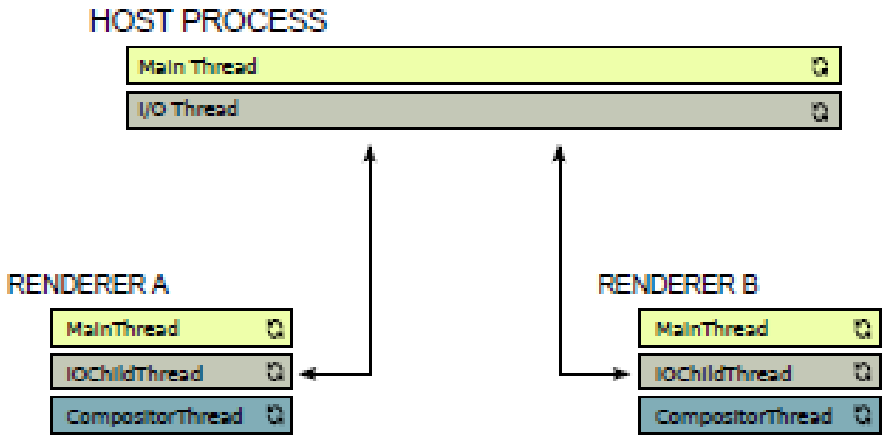

Рисунок 2.3 – Огляд архітектури Chrome

Хост-процес запускає вікно браузера верхнього рівня. Він має доступ до системних ресурсів, таким як мережу, файли системи, події призначеного для користувача інтерфейсу і т. д., Які він управляє від імені процесу непривилегированного рендеринга. Хост-процес запускає кілька потоків; найбільш важливими з них є:

– потік CrBrowserMain, який обробляє, наприклад, події взаємодії з користувачем

– IOThread, який обробляє, наприклад, IPC, мережа стек і файлову систему.

Процеси рендеринга обробляються обробленими пісочницями процесами для розбору, рендеринга і виконання Javascript. Зв'язок з хост-процесом виконується через систему взаємодії між процесами (IPC) на основі

повідомлення передача. Кожен рендер виконує кілька потоків; Найбільш важливими з них є:

– MainThread, де розбір ресурсів, розрахунок стилю, макет, живопис і неробочий Javascript працює,

– IOChildThread, який обробляє IPC-зв'язок з хост-процесом і

– CompositorThread, який покращує чуйність під час фази рендеринга, дозволяє прокрутку і перегляд анімації, в той час як основний потік зайнятий, завдяки знімку сторінки. Кожен з потоків процесів хоста і визуализатора підтримує, щонайменше, один цикл подій, який в значній мірі є FIFO чергою. Міжпоточної і межпроцессной комунікації здійснюється через повідомлення, що проходить через ці черги. Далі ми обговоримо сценарії, де сторінки різного походження можуть спільно використовувати цикли подій процесів хоста і рендеринга. У розділі 3 ми покажемо, як можна використовувати цей обмін для підслуховування.

2.1.3 Спільне використання в процесах рендеринга

Chrome підтримує різні політики, які визначають, як веб-сайт програми зіставляються з процесами візуалізації, і впливає на те, чи є загальні цикли подій. Політика за замовчуванням називається process-per-siteinstance. Для цього потрібне використання виділеного кошти візуалізації процесу для кожного екземпляра сайту. Тут сайт визначається як зареєстрований домен плюс схема. Наприклад, https://docs.google.com і https://mail.google.com:8080 знаходяться на одному сайті - але не від одного і того ж походження, оскільки вони різняться в субдомені і порте. Примірник сайту являє собою набір сторінок з того ж сайту, на якому можна отримати посилання один на одного (наприклад, одна сторінка відкрила іншу в новому вікні, використовуючи Javascript). Інші підтримувані політики більш дозвільного. Для Наприклад, групи політик процесу за сайт всі екземпляри сайту в тому ж процесі рендеринга, надійність торгівлі для більш низьких витрат пам'яті. Процес за вкладку політика виділяє один процес рендеринга кожній групі вкладки, пов'язані з скриптом. Нарешті, політика єдиного процесу дозволяє хосту і візуалізатором працювати в одній ОС процес (використовується тільки для цілей налагодження). Навіть в режимі обмеження за замовчуванням для кожного сайту політики, є деякі ситуації, які змушують Chrome для розміщення документів з різних сайтів в той же процес рендеринга, змушуючи їх ділитися подіями цикл:

– IFrames в даний час розміщені в тому ж процесі, що і їх батьків.

– провідники, ініційовані рендерер, такі як кліки посилань, форми, а пересланні сценарії будуть повторно використовувати той же рендерер, що і вихідна сторінка.

– коли кількість процесів рендеринга перевищує певний поріг, Chrome починає повторно використовувати існуючі замість створення нових.

В (64-розрядної) OSX і Linux поріг для повторного використання рендерера обчислюються шляхом поділу половини фізичного ОЗУ серед рендеринга, в припущенні, що кожен споживає 60 МБ. У наших експериментах на машині з 4 ГБ оперативної пам'яті ми могли б створити 31 нову вкладку до того, як будь-якої рендер був розділений, тоді як на машині з 8 ГБ ОЗУ ми спостерігали поріг приблизно 70 рендерів. Немає видимої угруповання політики для сторінок, які можуть спільно використовувати процес, коли це перевищує поріг, за винятком вкладок в режимі інкогніто не змішуючись з «звичайними» вкладками. Зокрема, ми не спостерігаємо жодних переваг для подібних витоків, однакові сайти, безпечні або ненадійні сторінки. Насправді навіть сторінки файлової системи (завантажені file: //) можуть спільно перебувати з довільним HTTP-сайтом.

2.1.4 Спільне використання в хост-процесі

Chrome обмежує доступ рендерер до привілейованих дій. Зокрема, візуалізатори повинні повідомляти з хост-процесом для мережевих запитів або призначений для користувача введення. Відповідні повідомлення всіх рендеринга пройти через цикл подій хост-процесу I / O нитку. Ми проілюструємо це повідомлення, використовуючи два різних приклади: як призначені для користувача дії передаються від хоста до відповідного процесу візуалізації і, навпаки, як мережу запитує потік від засобу візуалізації до хост-процесу.

– Потік призначеного для користувача інтерфейсу: дії користувача, такі як руху миші або кліки входять в браузер через основний потік хост-процесу. Основний потік хоста повідомляє призначене для користувача подія в відповідний рендерер шляхом передачі повідомлень між їх циклами введення-виведення, і рендер підтверджує отримання цього повідомлення. Навіть події без Прослуховувач Javascript займають цикл події основного потоку рендеринга для вимірного інтервалу.

– Мережевий стек: чистий стек Chrome - складна кроссплатформенная мережева абстракція. Будь-мережевий запит рендерер передається потоку введення-виведення хоста процесу, який перенаправляє його на глобальний диспетчер ресурсів який передасть його працівнику для виконання запиту. Цей робочий відкриє з'єднання, якщо необхідно і запросить ресурс. Після запиту виконується, заголовки відповідей приймаються і відправляються назад в процес рендеринга, який буде відповідати за ACK після прочитання, Нарешті, тіло і відповідні виклики викликаються.

2.2 Підслуховування на циклах подій в Chrome

У цьому розділі ми описуємо, як порушувати SOP підслуховування циклів подій хоста Chrome і процесів візуалізації. Для кожного з цих процесів ми описуємо потенційні сценарії загроз і представляємо просту HTML-сторінку, яка виконує Javascript, який може використовуватися для шпигунство. Потім ми представляємо наш інструмент моніторингу для візуалізації циклу подій в браузері.

2.2.1. Цикл подій процесу рендеринга

2.2.1.1 Сценарії загроз

Існує кілька сценаріїв, в яких сайт противника A може спільно використовувати цикл подій основного потоку рендеринга з місцем жертви V. Ці сценарії засновані на політиці Chrome для зіставлення сайтів з рендерер, див. Розділ 2.3. Ми наводимо два приклади:

– шкідлива реклама. У цьому сценарії A запускається як реклама iframed в V. SOP захищає Конфіденційність V і його юридичне використання шляхом логічної ізоляції обох середовищ виконання. Однак, iframe A здатний виконувати Javascript в циклі подій V, дозволяючи йому збирати інформацію про поведінку користувача в V.

– Keylogger. В цьому випадку A видає реєстраційну форму для аутентифікації своїх користувачів через OAuth V. Оскільки операція не вимагає особливих привілеїв і пароль ніколи не відправляється A, жертва може довіряти цьому і заповнити форму. Тим часом, монітори сторінок A, який може бути використовуються для відновлення паролів користувачів.

2.2.1.2 Методи моніторингу

Для контролю циклу подій визуализатора досить постійно виконувати асинхронні завдання і вимірювати часовий інтервал між наступними парами подій. Ми вимірювати дозвіл моніторингу в термінах інтервалу між двома наступними подіями вимірювання на інакше порожній цикл. Найбільш поширений спосіб розміщення асинхронних завдань програмним способом в Javascript є setTimeout. Однак, дозвіл може становити понад 1000 мс для неактивних вкладки, роблячи цей підхід марним для цієї мети шпигунства. Щоб збільшити розмір, замість цього ми використовуємо API PostMessage для відправки асинхронних повідомлень для себе. Код в лістингу 1 показує, як це досягається. Виклик функції performance.now () в рядку 2 function loop повертає мітку часу з високою роздільною здатністю який зберігається, як описано нижче. Заклик до self.postmessage (0, '*') в рядку 3 повідомлення повідомлення

```
1  function loop() {
2     save(performance.now())
3     self.postMessage(0,'*')
4  }
5  self.onmessage = loop
6  loop()
```

Рисунок 2.4 – Код Javascript для відстеження циклу подій основного потоку за допомогою API postMessage

«0» в циклі подій рендеринга, де другий аргумент «*» Вказує на відсутність обмежень на походження приймача. Рядок 5 реєструє цикл функції як прослуховувач подій, який дозволяє йому отримувати повідомлення, які він опублікував. Це призводить до того, що цикл рекурсивно публікує завдання, зберігаючи рендеринг, оскільки інші події все ще залишаються обробленими. Щоб звести до мінімуму шум, сам сценарій вимірювання, функція збереження в рядку 2 використовує попередньо виділений збірний масив (Float64Array) для зберігання всіх часових вимірів. Всупереч нормальному Рідкісні масиви Javascript, типізовані масиви уникають перерозподілу пам'яті і таким чином галасливі раунди збору сміття, див. Нижче. При цьому ми

досягаємо середньої затримки між двома послідовними завданнями близько 25 мс на нашій цільовій машині. Цього дозволу досить, щоб ідентифікувати навіть короткі Заходи. Наприклад, одна подія переміщення миші (без явного Прослуховувач подій) споживає близько 100 мс.

2.2.1.3 Інтерференція

В сучасних браузерах є кілька джерел шуму які впливають на точність вимірювання, крім очевидного ефекту базової апаратної платформи і ОС. Они включають:

– Компіляцію «точно вчасно» (JIT). JIT може запускати код оптимізації або деоптімізаціі, в разі Chrome за допомогою компіляторів CrankShaft і Turbofan, в моменти часу, які важко передбачити. Для наших вимірів ми покладаємося на етап розминки близько 150 мс для отримання повністю оптимізованого коду.

– Збір сміття (GC). У разі V8 GC включає невеликі колекції (так звані очисники) і основні колекції. Захвати - періодичні і швидкі (<1 мс); але основні колекції можуть займати> 100 мс, розподілених поетапно. За нашими даними, очищення легко ідентифікуються через їх періодичності, в той час як основні колекції можуть бути виявлені через їх характерних розмірів. У деяких браузерах, таких як Microsoft Internet Explorer, GC-раунди можуть бути активовані програмно, що допомагає усунути шум з вимірювань, що дозволяє більш точних атак.

Хоча всі ці функції знижують ефективність наших атак, цікаво їх розглядати як потенційні бічні канали. Наприклад, спостережуваний GC і події JIT можуть відображати інформацію про програму пам'яті і коду, відповідно.

2.3.2. Хід події процесу хоста

2.3.2.1 Сценарії загроз

Chrome гарантує, що всі кошти рендеринга події мережевого і призначеного для користувача взаємодії проходять через хост процесу "циклу введення-виведення, див. 2.4. Ми описуємо два де це може бути використано.

– Прихований канал. Сторінки різного походження на різних (відключених) вкладках можна використовувати загальні цикли подій для реалізації прихованого каналу, що порушує механізми ізоляції браузера. Це буде працювати, навіть якщо одна (або обидві) сторінки працюють в інкогніто Режимі. Цей канал може використовуватися для відстеження користувачів через сеанси або для добування інформації з підозрілих веб-сторінок без мережевого трафіку.

– фінгерпрінт. Вкладка, на якій запущена сторінка-вигнанець A, може визначити, які сторінки відвідують користувач в інших вкладках, відстежуючи загальний цикл подій. Виявлення початок навігації полегшується факт, що потоки введення-виведення блокуються на мить, коли користувач вводить URL-адресу та натискає кнопку введення.

2.3.2.2 Методи моніторингу

Існує безліч способів розміщення асинхронних завдань в цикл подій хост-процесу; вони відрізняються в термінах дозвіл, за допомогою якого вони дозволяють контролювати подія циклу і накладні витрати, які вони мають на увазі. Нижче ми описуємо два з тих методів, які ми використовували.

Мережеві запити. Перший спосіб - використовувати мережу запити на систематичний контроль циклу подій потоку введення-виведення хост-процесу. Дійсний запит мережі може зайняти кілька секунд, тільки з початку і кінцеві операції, видимі в циклі, який забезпечують недостатнє дозвіл для моніторингу. Щоб збільшити розмір, ми використовуємо невипробувані IP-адреса. Відповідні запити цикл подій потоку введення-виведення ідентифікуються як неприпустимі в браузері і викликати зворотний виклик без будь-яких Дозвіл DNS або створення сокетів. Цей механізм забезпечує дозвіл монітора 500 мс і має додатковий перевага незалежно від мережевого шуму. У лістингу 2 показаний код нашої процедури моніторингу. Ми покладаємося на API Javascript Fetch для публікації мережевих запитів. API-інтерфейс Fetch надає інтерфейс для збору ресурсів з використанням обіцянок, які ідеально підходять для управляти асинхронними обчисленнями завдяки їх простим синтаксисом для обробки зворотних викликів. У рядку 2 ми просимо і збережіть тимчасову мітку з високою роздільною здатністю. У рядку 3 ми просимо немаршрутізіруемий IP-адреса і встановлюємо

зворотний виклик відхилення обіцянки самому собі, щоб рекурсивно працювати, коли запит не виконується.

```
1  function loop() {
2     save(performance.now())
3     fetch(new Request('http://0/')).
4        catch(loop)
5  }
6  loop()
```

Рисунок 2.5 – Код Javascript для моніторингу введення-виведення хостапотока з використанням мережевих запитів

Загальні працівники. Другий метод заснований на мережі робочих, який є механізмом для виконання Javascript на задньому тлі. Веб-співробітники, які кілька сторінок, як правило, ОС-процес; це означає, що вони спілкуються через IPC і, отже, може використовуватися для шпигунства на потік введення-виведення хоста обробки. Цей механізм забезпечує дозвіл для моніторингу 100 мс. У лістингу 3 показаний код нашого моніторинг на робочому місці. Перший фрагмент визначає роботу працівника, яка полягає у відповіді на кожне отримане повідомлення. У другому фрагменті ми реєструємо працівника в рядку 1. У рядках 2-7 ми записуємо тимчасову мітку і рекурсивно відправляти повідомлення працівникові, аналогічні лістингу 1. Як в результаті ми вимірюємо час проходження в обидві сторони від сторінки до робітник, який відображає скупчення в подію уведення - виведення петлі. Зверніть увагу, що можна додатково збільшити вимір дозвіл шляхом запису часу в кожній кінцевій точці і злиття результатів.

```
1  onconnect = function reply(e) {
2     let port = e.ports[0]
3     port.onmessage = function() {
4        port.postMessage(0)
5     }
6  }
```

```
1  const w = new SharedWorker('pong.js')
2  function loop() {
3      save(performance.now())
4      w.port.postMessage(0)
5  }
6  w.port.onmessage = loop
7  loop()
```

Рисунок 2.6 – Javascript-код для моніторингу хоста Потік введення-виведення з використанням SharedWorkers

2.3.2.3 Інтерференція

Існує безліч різних джерел шуму і невизначеності в потоці введення-виведення хост-процесу. Найвідоміший включають чергування з основними потокомі хоста і повідомлення від інших засобів візуалізації, але також і графічний процесів і браузерів. Хоча ці перешкоди потенційно можуть використовуватися як бічні канали, шум стає швидко заборонним, коли цикл стає переповненим.

2.3.3 інструмент LoopScan

Ми реалізуємо описані методи перехоплення в розділах 3.1 і 3.2 в інструменті LoopScan, який дозволяє досліджувати характеристики бічного каналу викликаних спільними подіями. LoopScan заснований на простій HTML-сторінці, яка відстежує цикли подій процесів хоста і візуалізації. Він покладається на D3.js і забезпечує інтерактивну візуалізацію за допомогою міні-карти, масштабування і прокручування, що полегшує огляд слідів. Наприклад, рис. 8 заснований на скріншоті від LoopScan. Функціональність LoopScan в принципі покрита за допомогою потужного інструменту профілювання подій Chrome Trace (близько: трасування), в якому представлені докладні графіки полум'я для всіх процесів і потоків. Однак LoopScan має перевагу в забезпеченні більш точного часу інформацію про трасування затримки подій, ніж профілювальник, починаючи з завантаження сторінки за допомогою інструменту профілювання подій трасування сильно спотворює вимірювання. Джерело LoopScan загальнодоступним за адресою https://github.com/cgvwzq/ loopscan.

2.4 Напади

У цьому розділі ми систематично аналізуємо бічний канал викликаних загальними циклами подій в трьох типах атак: атака з ідентифікацією сторінки, атака, в якій ми підслуховує на дії користувача і приховану атаку каналу. Для всіх атак ми стежимо за циклами подій рендеринга і хост-процеси, як описано в розділах 3.1 і 3.2. Ми здійснював ці напади протягом

року, завжди використовуючи останню стабільну версію Chrome (від V52-V58). Отримані нами результати в значній мірі стабільні різні версії.

2.4.1 Ідентифікація сторінки

Ми описуємо, як трасування затримки подій, отримана з шпигунства на циклі подій може використовуватися для ідентифікації веб-сторінок завантажується в інші вкладки. Почнемо з пояснення наш вибір даних і процес збору врожаю і вибрані методів аналізу, то ми опишемо наш експериментальний настройки і отриманих результатів.

2.4.1.1 Вибір вибірки

Ми починаємо з списку сайтів Alexa Top 1000, від який ми видаляємо дублікати. Тут дублікати - це сайти які поділяють субдомен, але не домени верхнього рівня (наприклад, «google.br» і «google.com»), і це, ймовірно, щоб мати аналогічні сліди затримки подій. З решти список, ми довільно вибираємо 500 сайтів в якості нашого набору зразків. Ця скорочення полегшує ретельне дослідження даних і простір параметрів.

2.4.1.2 Збір даних

Ми відвідуємо кожну сторінку в наборі проб 30 разів, рендерер для запису слідів eventdelays під час фази завантаження. Трасування затримки подій для процесу рендеринга складаються з 200 000 елементів даних кожен. На нашій тестовій машині, дозвіл вимірювання (т. Е. Затримка між двома наступними подіями вимірювання в іншому випадку порожні петля) становить приблизно 25 мс. Тобто кожен слід захоплює близько 5 секунд (200 000 × 25 мс = 5 с) процес завантаження сторінки в наборі зразків. Трасування затримки подій для хост-процесу складаються з 100.000 елементів даних кожен. Дозвіл виміру лежить в діапазоні 80 – 100 мс, тобто кожен трасувальні захоплення близько 9 з процесів завантаження сторінки. Ми автоматизуємо процедуру збирання для рендеринга процесу в такий спосіб:

1) Відкрийте нову вкладку через target = window.open (URL, '_blank ");

2) Контролюємо цикл подій, поки буфер трасування нічого очікувати заповнений;

3) Закриємо вкладку;

4) Відправляємо трасування на сервер;

5) Чекаємо 5 секунд і переходимо до 1 з наступним URL-адресою.

Процедура збирання для хост-процесу відрізняється тільки в тому, що ми використовуємо атрибут rel = "noopener" в порядку для створення нового рендеринга.

Ми провели вимірювання на наступних трьох машини:

1) Debian 8.6 з ядром 3.16.0-4-amd64, що працює на Intel i5 @ 3,30 ГГц x 4 з 4 ГБ ОЗУ і Хром v53;

2) Debian 8.7 з ядром 3.16.0-4-amd64, що працює на Intel i5-6500 @ 3.20GHz x 4 з 16 ГБ RAM і Chromium v57; а також

3) OSX, що працює на Macbook Pro 5.5 з Intel Core 2 Duo @ 2,53 ГГц з 4 ГБ оперативної пам'яті, і Chrome v54.

Вимірюємо час на екземплярі Chrome з двома вкладками, один для процесу шпигуна, а інший для цільової сторінки. Для процесу рендеринга ми збираємо дані про всі машинах; для хост-процесу на (2) і (3). У загальному і цілому, ми отримуємо, таким чином, п'ять корпусів по 15 000 слідів.

2.4.1.3 Класифікація

Гістограми затримки події. Наш перший підхід кластеризация спостережуваних подій затримок навколо k центрів і перетворити кожну трасування в гістограму, що представляє кількість подій, які потрапляють в кожен з k класів. Потім ми використовуємо евклідова відстань як міру подібності на k-мірних підписах. Цей підхід натхненний поняттям мембран в. Здається, він підходить для класифікації eventdelay сліди, отримані з циклів подій, тому що, наприклад, статичні сторінки з декількома зовнішніми ресурсами більше ймовірно, призведе до тривалих подій на початку і стабілізує в той час як сторінки з ресурсами Javascript і анімації, ймовірно, приведуть до більш неправильним шаблонами і виробляють більшу кількість тривалих затримок. На жаль, наші експериментальні результати бентежать: менше 15% від швидкості розпізнавання в невеликих наборах даних.

Динамічне деформування часу. Наш другий підхід для підтримки тимчасової інформації про спостережуваному Заході. Однак точні моменти, коли події відбуваються, схильні до дії шуму навколишнього середовища. Наприклад, затримки в мережі будуть впливати на тривалість мережі запитів і, отже, прибуття подій на захід петлі. Замість цього ми фокусуємося на

відносному впорядкування подій як більш надійна функція для ідентифікації сторінки. Це мотивує використання динамічного перетворення часу (DTW) як показник подібності на трасуванні затримки подій. DTW широко використовується для класифікації часових рядів, тобто послідовностей точок даних, прийнятих послідовно і однаково проміжні точки в часі. DTW є поняття відстані який розглядає як «близькі» залежать від часу дані подібних формі, але різна швидкість, тобто DTW є горизонтальним стисненням і розтягненням. Це корисно, наприклад, коли людина бажає призначити невелику відстань оцінки до тимчасового ряду «abc» і «abbbbc», нечутлива на тривалу тривалість «b». Формально DTW порівнює два тимчасових ряду: запит, X = (x1; :::; xn) і посилання, Y = (y1; :::; ym). Для цього ми використовуємо невід'ємні функція відстані f (xi; yi), певна між будь-якою парою елементів xi і yj. Метою DTW є пошук відповідності точок в X з точками в Y, такі, що (1) кожна точка (2) відносний порядок точок в кожній послідовності (монотонність), (3) і кумулятивний відстань (т. Е. Сума значень f) по всім точки збігу мінімізовані. Це зіставлення називається шлях деформації, а відповідне відстань - час деформується відстань d (X; Y).

На рис. 2.7 показаний шлях деформації між тимчасові ряди, що відповідають спостережуваним затримок при завантаженні різних веб-сторінок.

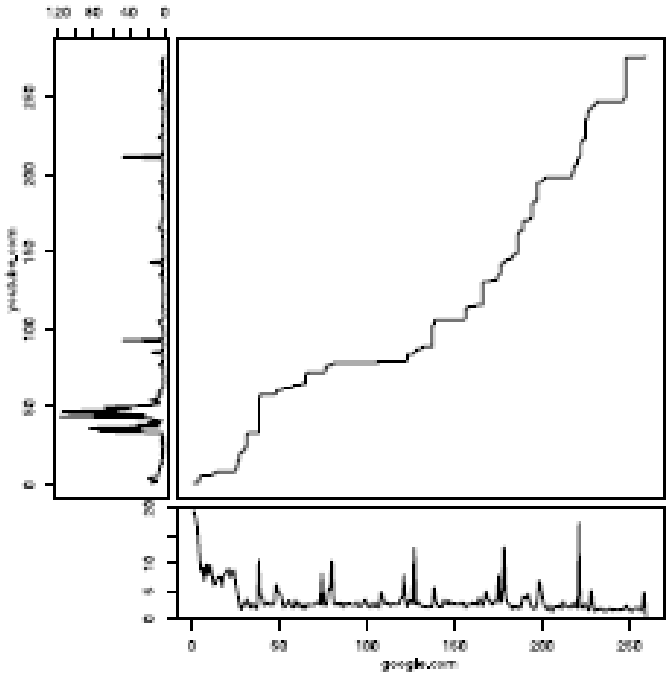

Рисунок 2.7 – Шлях у верхньому правому квадраті представляє оптимальне вирівнювання між точками часового ряду відповідає «google.com»

(горизонтальна вісь) з точок в часі ряду «youtube.com» (вертикальна вісь)

2.4.1.4 Методи прискорення

На жаль, час, необхідний для обчислення d (X; Y) квадратична по довжині вхідних послідовностей і робити не масштабируемую до необроблених даних, отриманих в наших вимірах. Ми покладаємося на два види технологій прискорення, один на рівні даних, а інший на рівні алгоритму: На рівні даних ми зменшуємо розміри наших даних шляхом застосування базового алгоритму вибірки: ми поділяємо необроблений слід в групи вимірювань, відповідних до тимчасових інтервалах тривалості P і замінити кожен з цих груп одним представником. Цей представник може бути обчислений шляхом підсумовування по групі або шляхом її середнього, максимального або мінімального. Загальна функція суми дає найкращі результати в різних вибірках функції і той, який ми використовуємо далі. відбір проб зменшує розмір слідів в P = t, де t середня тривалість затримки події. рис. 2.8 показує два графіка з вихідними даними, взятими з основного засобу рендеринга петля потоку і відповідні їй тимчасові ряди після відбору проб. На алгоритмічній рівні ми використовуємо два набору методів для обрізки пошуку оптимального деформування шляху, а саме віконні та ступінчасті шаблони [9].

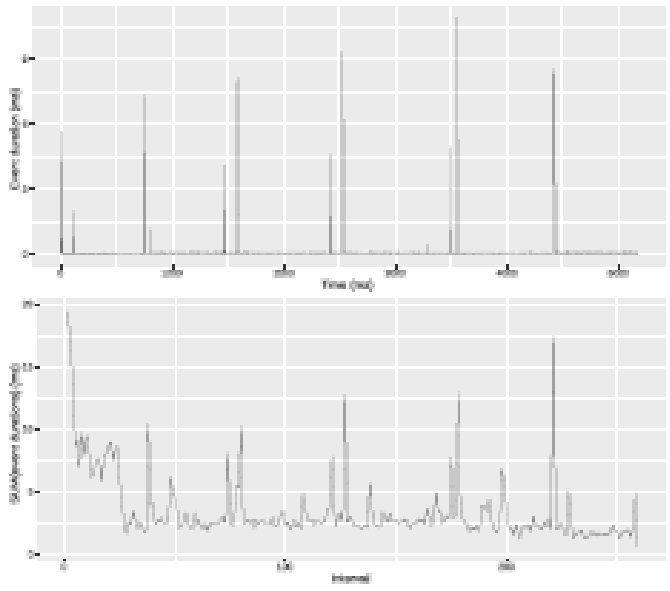

Рисунок 2.8 – Верхня фігура являє собою необроблений слід 200 000 вимір часу з основного потоку рендеринга при завантаженні «google.com». Нижня

цифра відображає ті ж дані після перетворення під час серії з P = 20 мс, тобто з використанням тільки 250 точок даних

– Вікно - це евристика, яка забезпечує глобальне обмеження на конверті траєкторії. Швидкість але не знайдуть оптимальних шляхів деформації, які лежать поза конверта. Два добре встановлених обмеження регіонами є група Сако-Чіба та паралелограм Ітакура, див. рис. 2.9.

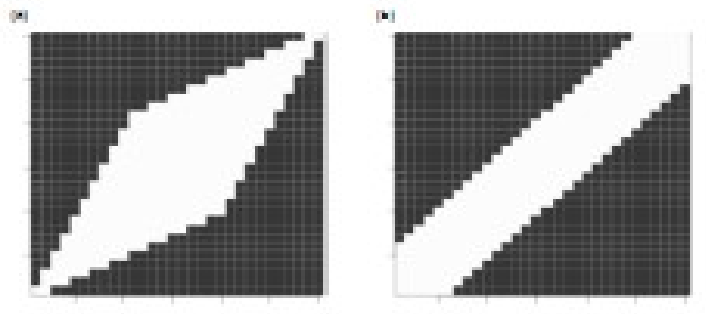

Рисунок 2.9 – Глобальне обмеження вікна визначає конверт обмеження простору пошуку для оптимальних траєкторій: (а) паралелограма Ітакура і (б) смуга Сако-Чіба

– Крокові шаблони - це евристика, яка ставить локальне обмеження на пошук шляху деформації з точки зору обмежень на його схилі. Зокрема, ми покладаємося на три добре відомі шаблони кроків, доступні в P. Інтуїтивно, симетричний шаблон 1 сприяє прогресу, близькому до діагоналі, шаблон сімметрічного2 допускає довільні стиснення і розкладання, і асиметричні сили кожної точки в посилання повинна використовуватися тільки один раз.

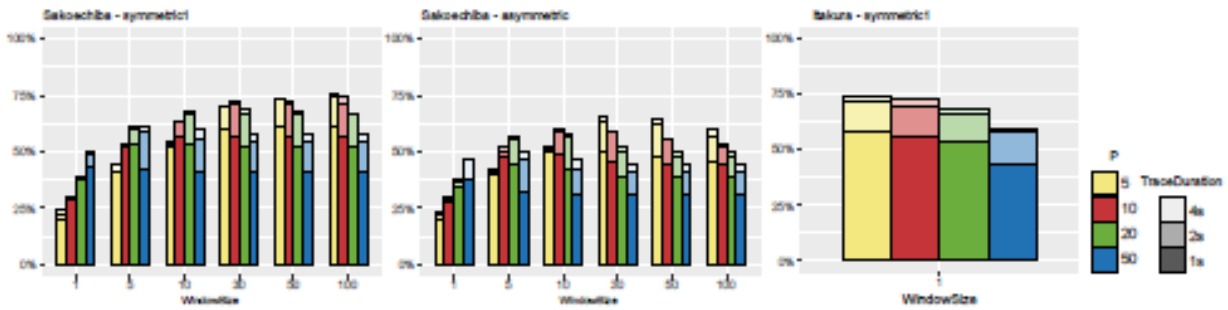

Рисунок 2.10 – Продуктивність ідентифікації веб-сторінки після настройки зі слідами від рендеринга на машині Linux (1). ефект P, traceDuration і windowSize, з трьома комбінаціями stepPattern і windowType

2.4.1.5 Налаштування параметрів

Ми систематично визначаємо оптимальну конфігурацію параметрів для кожного циклу подій на кожній машині. Уникати overfitting, ми ділимо наш набір даних на 30 трас (на сторінку, петлі і машини) в 15 трас для настройки і 15 для перехресної перевірки. Для кожної конфігурації параметрів ми виконуємо легку версію (з 3 раундами) оцінки фази, описаної в розділі 4.1.6. рис. 2.10 візуалізує витяг з результатів, які ми отримуємо для рендеринга процес машини Linux (1). Фаза настройки дає наступну інформацію:

– оптимальні параметри залежать від циклу, але з'являються бути стабільним на всіх машинах.

– вимірювання фази завантаження протягом 2 секунд достатню для розпізнавання веб-сторінки; виграш у визнанні від використання більш довгих трас мізерно.

– P і windowSize - це параметри з найбільший вплив на показник визнання. Однак вони також мають найбільший вплив на обчислювальну вартість (оптимальний вибір є найдорожчим).

– комбінація stepPattern = сімметрічний1 і windowType = sakoechiba зазвичай дає найкращі результати.

2.4.1.6 Експериментальні результати

Ми оцінюємо ефективність ідентифікації сторінки через загальні цикли подій хоста і засоби візуалізації процесу на кожній окремій машині, а також через рендеринга на двох різних машинах. Для цього ми вибираємо верхню конфігурацію для кожного корпусу з етапу настройки і виконати в 10 разів перехресну перевірку. У кожному з 10 раундів ми поділяємо валідацію, встановлену в навчальний набір, що містить одну трасування кожної сторінки і набір тестів, що містить три різні (з 14 доступних) слідів кожної сторінки. Для кожної з трас в тестовому наборі ми обчислюємо набір найближчих матчів в навчальному наборі згідно час деформування. Ми вимірюємо продуктивність в термінах k-match швидкість, яка представляє собою відсоток сторінок в наборі тестування для якого справжнє

відповідність знаходиться в межах набору k найближчих Матчі. Ми скорочуємо 1-матчную ставку за рахунок визнання відсоток, тобто відсоток сторінок, на яких найкраще збіг правильні. Результатом перехресної перевірки є середня к-матчева ставка за всіма 10 раундів.

– Ми можемо правильно ідентифікувати сторінку, стежачи за рендер від (1) до 76: 7% випадків і правильно звузити до набору з 10 кандидатів до 91: 1% випадків.

– Ми можемо правильно ідентифікувати сторінку, хоча господар процесу від (3) до 23: 48% випадків і аж до набору з 10 кандидатів у розмірі до 46: 6% від випадків.

– Ми підкреслюємо, що ці показники розпізнавання отримані використовуючи один слід для навчання.

– Розпізнавання простіше за допомогою засобу візуалізації, ніж через хост. Це пояснюється відмінністю шуму і вимірювання, див. Розділ 3.2.3. Крім того, більшість операцій на хості блокують I / O потік, сигналізуючи про початок і завершення, тоді як засіб візуалізації блокується протягом всього виконання кожного завдання Javascript.

– Ми спостерігаємо різні показники розпізнавання на різних машини. Однак однорідність апаратних засобів і програмне забезпечення Macbooks сприяють повторному використанню систему адаптації через машини, що може зробити віддалену ідентифікацію сторінки більш здійсненно.

– Ми отримуємо показники визнання нижче 5% для визнання через машини (1) і (3). Причина цього низька продуктивність - це те, що події на ноутбуці OSX частіше в 2 рази більше часу, ніж на настільному комп'ютері Linux. Ця різниця відображається в висоті піку (а не в їх положенні), які караються за DTW. Нормалізація вимірювань може поліпшити взаємне розпізнавання. Код та набори даних, що використовуються для настройки і перетягування доступні у вигляді бібліотеки R на https: // github.com/cgvwzq/rlang-loophole.

2.4.1.7 Загрози дійсності

Ми проводимо наші експерименти в сценарії з закритим світом з двома вкладками (шпигун і жертва), спільно використовують подія петля. У реальних сценаріях може бути більше сторінок одночасно запускаючи браузер, який виявлятиме Сильніше. Найгірший випадок для моніторингу приймаючої сторони процес відбувається, коли вкладка виконує потокове

мовлення, оскільки петля повністю заливається. Цикл рендеринга, проте, в цілому більш стійкий до шуму, викликаного іншими вкладки в браузері. З іншого боку, наші атаки не використовують вихідний код сторінки або докладні відомості про планування Chrome система з пріоритетними чергами, GC з періодичними очищеннями або завданнями рендеринга кадрів. Ми віримо що беручи до уваги цю інформацію, можна значно поліпшити можливості перехоплення противника і атаки навіть в галасливих сценаріях з відкритим світом.

2.4.2 Виявлення поведінки користувача

У цьому розділі ми показуємо, що можна виявити призначені для користувача дії, що виконуються на вкладці перехресного походження або iframe, коли процес рендеринга є загальним. Спочатку ми описуємо атаку відновлення інформації синхронізації часу між натисканнями клавіш Форми входу в систему Google OAuth, які забезпечують більш високу точність ніж існуючі мережеві атаки.

2.4.2.1. Тимчасова атака в режимі натискання клавіш в Google

Форма входу в OAuth Багато веб-додатки використовують протокол OAuth для користувача аутентифікація. OAuth дозволяє користувачам входити в систему, використовуючи свої ідентичність з довіреними постачальниками, такими як Google, Facebook, Twitter, або Github. У браузері цей процес зазвичай реалізується наступним чином:

1) Веб-додаток A відкриває форму входу в систему довірений постачальник T;

2) Користувач V вводить своє ім'я і пароль і відправляє форму T;

3) T генерує токен авторизації. Оскільки вікно форми входу в систему розділяє подія петлі з рендерер відкривача, зловмисний A може підслуховувати на події натискання клавіш, видані формою входу.

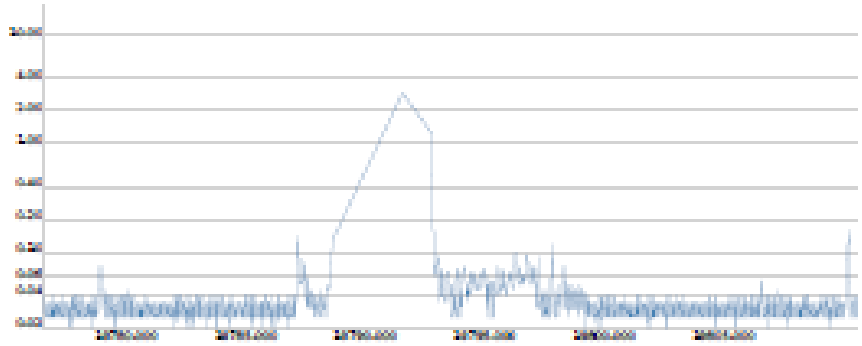

Рисунок 2.11 – Шаблон затримки, що генерується натисканням клавіші в форма входу в Google OAuth

На рис. 2.11 показана трасування затримки події натискання клавіші як видно з Прослуховувач в циклі подій рендеринга. Трасування містить дві характерні послідовні затримки, викликаних Прослуховувач подій keydown і keypress. Ми використовуйте це спостереження для ідентифікації натискань клавіш, шляхом сканування трасування затримки подій для пар послідовних затримок, які знаходяться в межах заздалегідь визначеного діапазону, відмовляючись від будь-якого навчання або автономної роботи. Лістинг 4 містить скрипт, який виконує ця операція. Визначимо 0: 4 мс в якості нижньої межі і 3: 0 мс в якості верхньої межі діапазону. Ми вибрали це до збору даних, шляхом ручного контролю з декількох натискань клавіш. Зверніть увагу, що ця калібрування може виконуватися автоматично, виходячи з взаємодій жертви з сторінкою, контрольованої зловмисником.

```
1  const L = 0.4, U = 3.0, keys = []
2
3  for(let i=1; i<trace.length-1; i++){
4      let d1 = trace[i] - trace[i-1],
5          d2 = trace[i+1] - trace[i]
6
7      if (L<d1<U && L<d1<U){
8          keys.push(trace[i])
9      }
10 }
```

Рисунок 2.12 – Код псевдо-Javascript для виявлення натискання клавіш у вигляді тимчасових міток. Ми класифікуємо тимчасову мітку як натискання клавіші, якщо відмінності з попередніми і наступні тимчасові мітки (d1 і d2) знаходяться в зумовлений діапазон

2.4.2.2 Експериментальна оцінка

Щоб оцінити ефективність цієї атаки, ми маємо реалізовано шкідливий додаток A, яке витягує інформація синхронізації часу між клавішами від користувача V за допомогою Google OAuth. У центрі нашої оцінки полягає у визначенні точності натискання клавіші тайминги можуть бути виміряні через цикл подій.

Ми моделюємо тимчасову атаку між натисканнями клавіш в 4 кроки, які описані нижче і проілюстровані на рис. 2.13.

Рисунок 2.13 – Експериментальна установка для оцінки ефективності автоматичного виявлення крос-рендери

1) Сценарій Selenium3, діючий як V, переходить до A, клацає на кнопці входу в систему (яка виводить OAuth від Google форма входу в систему), вводить пароль і відправляє форма.

2) Тим часом атакуючий A контролює основні потоку подій з використанням атаки, описаної в Розділ 4.2.1.

3) V і A посилають сервера мітки часу реальні і виявлені натискання клавіш, відповідно.

4) Ми обчислюємо точність виявленого натискання клавіш, де ми використовуємо тимчасові мітки справжні натискання клавіш як істини. Відповідність тимчасові мітки вимагають обліку затримки (6 - 12 мс на нашій машині) між Selenium запуск події, а Chrome – його отриманням. Ми використовуємо в якості інтервалів між натисканнями клавіші випадкові затримки рівномірно від 100 до 300 мс. Цей вибір натхненний, які повідомляють про середню затримку між натисканням клавіш 208 мс.

Використання випадкових затримок досить для оцінки точність підслуховування натискань клавіш, але він, очевидно, не розкриває ніякої інформації про пароль, крім його довжини.

2.4.2.3 Експериментальні результати

Ми проводимо експерименти з витяганням 10 000 паролів з набору даних RockYou, де ми отримуємо наступні результати:

– у 91,5% випадків наша атака правильно ідентифікує довжину пароля. У 2,2% випадків, атака пропускає один або кілька символів, а в 6,3% випадків він повідомляє неправдиві символи.

– для паролів, довжина яких була правильно ідентифікована, середня різниця в часі між істинним натискання клавіші і виявлене подія натискання клавіші становить 6,3 мс, які ми в основному відносимо до впливу Селена.

Це вплив скасовується при обчисленні середнє розходження між істинним меж'ядерних рухом затримки і виявленої затримки між натисканнями клавіш, що становить 1,4 мс. Шум цих вимірювання низький: ми спостерігаємо стандартне відхилення 6,1 мс, тоді як автори повідомляють про 48,1 мс для їх мережевих вимірювань. В цілому, наші результати показують, що спільний захід петлі в Chrome дозволяють значно точніше відновлювати час натискання клавіш, ніж мережеві атаки. Більш того, цей сценарій полегшує визначення часу, коли події натискання клавіш вводять цикл (від спливання до форми подання), яке вважається основною перешкодою для міжсайтових атак на мережевий трафік. Атрибути часу натискання клавіш на основі моніторингу procfs або кеші процесора можуть витягувати більш дрібнозернисті інформації про натиснення клавіш, таких як стримування в певних піднаборі ключів. Однак вони вимагають доступ до файлової системи або більш сприйнятливий до шуму, обумовлений до ресурсу, який розподіляється між усіма процесами в система. Навпаки, наша атака дозволяє здійснювати цілеспрямоване прослуховування без особливих привілеїв.

2.4.2.4. Відкриті виклики для розпізнавання користувальницьких подій

На закінчення ми обговоримо дві розпізнавання користувальницьких подій, а саме виявлення користувача подій за межами натискання клавіш і виявлення подій в хост-процес браузера.

Виявлення користувальницьких подій за межами клавіш. Безперервний рух миші призводить до послідовності подій, кожен з яких несе інформацію про координати траєкторії курсора. Ці події видаються за допомогою затримка між подіями 8 мс і (порожній) прослуховувач подій операція блокує цикл протягом приблизно 0,1 мс. Конкретна частота і тривалість цих подій роблять миша руху (або подібні дії, наприклад прокрутка), легко визначається з LoopScan, як показано на рис. 2.14.

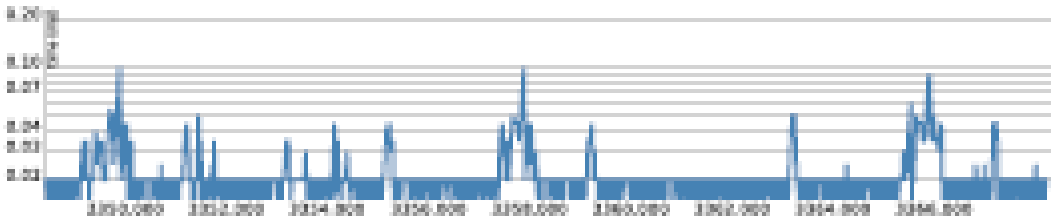

Рисунок 2.14 – Рух миші, захоплене інструментом LoopScan. На графіку показані 3 затримки тривалістю 0,1 мс (при t одно 3350, 3358 і 3366) із затримкою між подіями 8 мс

Аналогічно, події клацання миші, відповідні «вгору», або «вниз», можна визначити за допомогою LoopScan. Їх форма залежить від конкретного слухача подій шпигунства веб-сторінки і клацання елементу HTML. Ми очікуємо що події з конкретними слухачами легше ніж події без зареєстрованих Прослуховувач подій, тобто дії користувача, що не запускають виконання Javascript. Однак ми можемо використовувати контекст, в якому відбувається подія, що зменшує простір пошуку. Наприклад, більшість клацань миші з'являються тільки між двома послідовностями подій руху миші. В даний час ми вивчаємо методи, що дозволяють автоматична ідентифікація таких шаблонів в eventdelay потоки. Перспективною відправною точкою для цього є існуючі он-лайн варіанти динамічного деформування часу.

Виявлення користувальницьких подій в хост-процесі Наша дискусія досі зосередився на виявленні користувальницьких подій в цикл подій процесу рендеринга. Однак, всі користувачі події відбуваються в основному потоці хост-процесу і відправляються до конкретного рендерингу через подію цикл потоку введення-виведення хоста. Отже, будь-яка дія користувача може в принципі, виявляється шпигунством на хості. На жаль, наші поточні методи не точні досить для цього завдання, оскільки потік

введення-виведення хоста більше шум, ніж основний потік рендеринга, і ефект дію користувача в хост-процесі обмежується короткою сигналізацією, тоді як основний потік рендеринга вплинув на виконання відповідного Javascript прослуховувач подій.

2.4.3 Таємний канал

У цьому розділі ми покажемо, як загальні цикли подій в Chrome може використовуватися для реалізації прихованих каналів, тобто канали для незаконної зв'язку за походженням. Спочатку розглянемо випадок з перехресними сторінками поділ циклу подій основного потоку рендеринга перед ми переходимо до випадку з перехресними сторінками, цикл подій потоку введення-виведення хост-процесів.

2.4.3.1 Процес рендеринга

Ми реалізуємо канал зв'язку для передачі повідомлення від сторінки S-відправника до приймача з крос-початком сторінки R працює в тому ж процесі рендеринга. Для цього ми використовуємо просту односпрямовану передачу схему без корекції помилок. Зокрема, ми кодируем кожен біт з використанням тимчасового інтервалу фіксованою тривалості tb. Оптимальний Зміною tb залежить від системи. У нашому експерименті ми спробували різні значення, з tb = 5 мс, даючи гарні результати на різних платформах: Chromium 52.0 на 64-бітової версії Debian і Chrome 53 на OSX. У кожному з цих інтервалів ми робимо наступне:

– відправник S простоює для передачі 0; він виконує завдання блокування тривалості t <tb для передачі 1.

– приймач R контролює цикл події рендеринга основна нитка з використанням описаних методів в розділі 3.1; він декодує а 0, якщо довжина спостережуваного завдання нижче порога (пов'язаного з t), і а 1 в іншому випадку.

Передача починається з S відправки 1, яка використовується агенти синхронізувати свій годинник і почати підрахунок тимчасові інтервали. Передача закінчується тим, що S відправляє null байт. З цієї базової схемою ми досягаємо 200 біт = с. Ці цифри можуть бути значно поліпшені з використанням більш складних схем кодування з механізмами корекції помилок; тут нас цікавить тільки в доказі концепції. Відзначимо, що існує

ряд альтернативних прихованих каналиов для передачі інформації між сторінок, які працюють в одному і тому ж рендерер, наприклад, використовуючи window.name, location.hash, history.length, положення смуги прокрутки або window.frames.length. Які розрізняє канал на основі циклу подій, так це те, що він робить не вимагати підключення відправника і приймача, тобто їм не потрібно вести посилання один на одного, щоб спілкуватися.

2.4.3.2 Хост-процес

Ми також реалізуємо канал зв'язку для передачі повідомлення між двома процесами спільного рендеринга розділяючи хост-процес. Передача однонаправлена від відправника S до приймача R. На рис. 2.15 – показано, як це канал може використовуватися, навіть якщо одна зі сторін переглядає в режимі інкогніто.

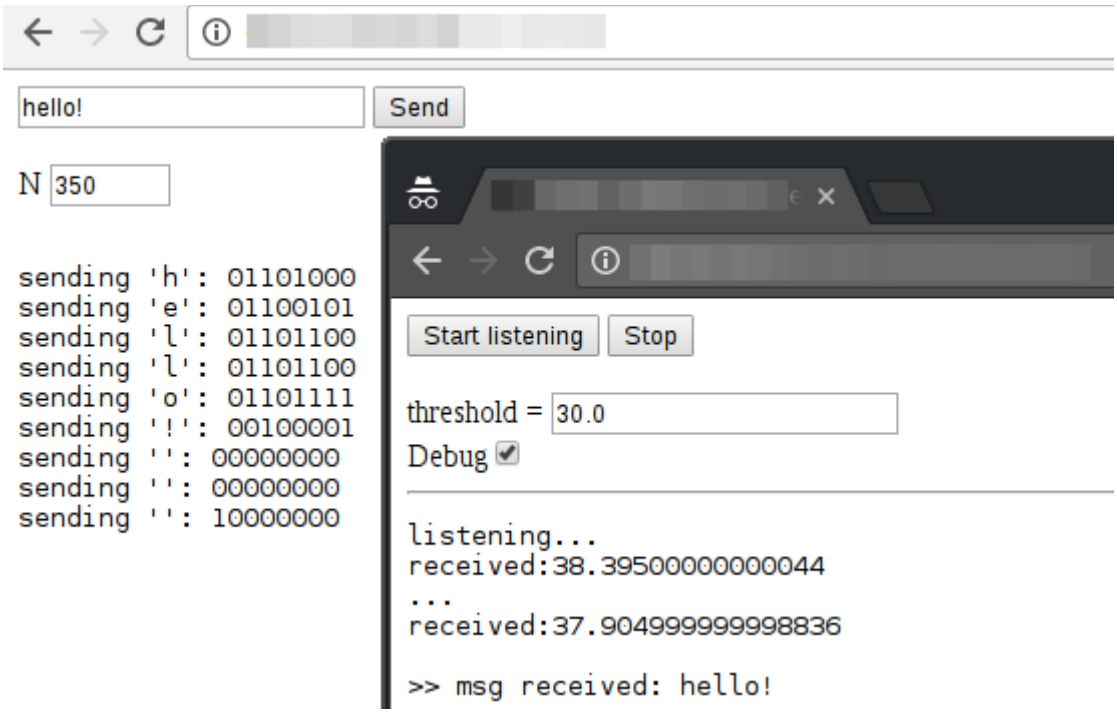

Рисунок 2.15 – Прихований канал через цикл подій введення-виведення хост-процесу Chrome. Виберіть в іншому рендерер процесів (один з них переміщається в режимі інкогніто) спілкуються

Як і раніше, ми кодируем кожен біт з використанням тимчасових інтервалів фіксованою тривалості tb. Протягом кожного інтервалу ми робимо наступне:

– відправник S простоює для передачі 0; він публікує N вибірки запитів в чергу потоку введення-виведення для відправки а 1.

– приймач R контролює цикл подій введення-виведення потік хост-процесу з використанням описаних методів в розділі 3.2. Він декодує 0, якщо число спостережуваних подій протягом тимчасового інтервалу tb нижче порогове значення і 1 в іншому випадку. Оптимальні значення N і tb сильно залежать від машини. У наших експериментах ми досягаємо добрих результатів, що працюють на різних системах, з tb = 200 мс і N = 350, які дають нам швидкість передачі 5 біт / с. Цей показник значно нижче, ніж для зв'язку за допомогою засобу візуалізації циклу подій, що пояснюється різницею в шуму і моніторингу обох каналів, як обговорювалося в розділі 3.2.3.

Сценарій загрози цього прихованого каналу більше ніж попередній для циклу візуалізації. Наприклад, він може використовуватися для отримання інформації про збільшення з атакованого домену (на вкладці, яка виконує шкідливі Javascript). ІспользованіеWorkers (які є фоном потоку, який запускаються незалежно від призначеного для користувача інтерфейсу), ми може передавати інформацію за походженням, не зачіпаючи користувача і без створення мережі трафік.

2.5 Обговорення

Ми показали, як об'єднання циклів подій призводить до синхронізації бічних каналів і продемонстрували різні атаки на Chrome. Ми повідомили про наші висновки на захист Chromium команди, яка вирішила не вживати заходів в даний час. Проте, наші результати вказують на фундаментальну безпеку проблеми в керованої подіями архітектури браузерів, які в кінцевому підсумку необхідно вирішити фундаментальним чином. Нижче ми обговоримо, як впливають інші платформи і представити можливі контрзаходи.

2.5.1 За межами Chrome

Ми концентруємося на Chrome в нашому аналізі, тому що це найбільш широко використовуваний браузер, і оскільки він був першим для реалізації багатопроцесорної архітектури. Однак, є вагомі причини очікувати подібних бічних каналів в інші браузери, так як всі вони слідують тим же подіям парадигми і покладатися на аналогічні архітектури. Наприклад, останні версії Firefox з багатопроцесорним support5 також покладається на привілейований процес браузера і кілька процесів контенту, які, на відміну від рендеринга в Chrome, діють як пул потоків для кожного origin (кожен зі своєю власною чергою повідомлень). Незважаючи на це різниця, тести з LoopScan в Firefox версії 55 показують, що перевантаження на обох циклах подій наблюдаема через початок і вкладки. Зокрема, ми застосували метод моніторингу для рендерер, описані в розділі 3.1.2, на мікрооб'єкті з набором з 30 сторінок з 15 трасами кожен. Ми досягли рівня визнання 49%, що нижче рівень визнання, досягнутий в Chrome для набору 500 сторінок. Справедливе порівняння між обома архітектурою потребують кращого розуміння політики Firefox для зіставлення сайтів з потоками і подіями.

2.5.2 Контрзаходи

Атаки, представлені в цьому документі, засновані на двох можливостях противника: (1) можливість відправляти завдання в петлі з високою частотою і (2) здатність точно вимірювати відповідні різниці в часі.

Обмеження швидкості. Очевидний підхід до лічильника (1) полягає в тому, щоб накласти обмеження на швидкість, з якою можуть відправлені в цикл подій. На жаль, обмеження швидкості на увазі штрафи за виконання, що особливо проблематично для асинхронного коду. На рівні рендеринга одна з можливостей полягає в тому, щоб покладатися про скупчення і обслуговуванні політики [10]. З цією політикою, цикл подій накопичує всі вхідні завдання в буфері на період T, а потім обробляти і обслуговувати всі накопичені робочі місця від партії A, за якими слідують всі роботи від B. Це має ту перевагу, що обмежує кількість витоку інформації при збереженні високої амортизації пропускної здатності. На рівні хост-процесу, де вибір ресурсів є однією з основних проблем, пов'язаних з продуктивністю, пов'язаний зі швидкістю обробки, неприйнятний. Тут він видається більш розумним контролювати діяльність МПК всіх рендерер і карати або відзначати тих, хто демонструє погане або аномальна поведінка, наприклад, по лініях.

Зменшіть дозвіл годин. Очевидний підхід до Лічильнику (2) повинен обмежувати дозвіл доступних годин. Це вже було застосовано постачальниками браузерів для пом'якшення інших видів каналів синхронізації, але ці зусилля навряд чи вдасться, як показано в: Сучасні у браузерів є значне число способів вимірювання без явних годин. Наприклад, деякі останні подвиги використовують таймери з високою роздільною здатністю на основі верх SharedArrayBuffers. Нинішнє дозвіл продуктивність обмежена 5 мс, що робить мікроархітектурнимі тимчасові атаки складні, але не виключають виявлення подій Javascript.

Повна ізоляція. Як описано в Розділі 2.2, Chrome многопроцессорная архітектура намагається використовувати інший визуализатор для різного походження, за винятком деякого кута випадків. «Проект ізоляції сайту» - це постійне зусилля для забезпечення повної політики процесу на кожному сайті, це означає: забезпечення наскрізних навігаційних процесів, перехресні процеси Javascript і внепроцессние процеси плаваючі фрейми. Все це, не нав'язуючи занадто багато накладних витрат. Одне відкрите питання - як обробляти процес системи межа, а саме, які сайти повинні мати перевагу ізоляції, або евристика для повторного використання процесу повинна бути використовуваний. Недавнє пропозицію «IsolateMe» ставить розробників відповідає за прохання ізолюватися від інших веб-контент (навіть якщо він не дає твердої гарантії).

Дроселювання ЦП. Chrome v55 являє API, який дозволяє обмежити, скільки ЦП фонова сторінка дозволено використовувати і дросселіровать завдання, коли вони перевищують цю межу. Це впливає на фонові вкладки, які намагаються зазирнути на основний потік рендеринга, але все ж дозволяє шпигувати на (і з) будь-якого iframe і спливаючого вікна, а також на Потік введення-виведення хост-процесу через спільних робочих. Крім того, фонові вкладки зі звуковою активністю не так як вони завжди позначаються як передній план. Оскільки сторінки Chrome v57 (або вкладки) піддаються тільки дроселювання через 10 секунд в фоновому режимі, що теж щоб запобігти атакам.

2.6 Пов'язана робота

Тимчасові атаки на веб-браузери відносяться до Felten і Schneider, які використовують кеш браузера для отримання інформації про історію перегляду користувачів. Зовсім недавно, так звані міжсайтовий тактові атаки

використовували той факт, що браузер підключається cookie для всіх запитів, навіть якщо вони виконуються за походженням. Наявність або відсутність цих файлів cookie можуть бути визначені шляхом вимірювання часу, що показує інформацію про стан користувача на довільних сайтах. Особливим випадком є атаки на міжсайтовий пошук, які обійшовши політику одного і того ж походження для отримання конфіденційної інформації, шляхом вимірювання часу, який потрібен для браузера для отримання відповідей на пошукові запити. Використовуються інші класи шкідливих атак, заснованих на браузері тимчасові відмінності в операціях рендеринга або просто використовуйте браузер в якості точки входу для Javascript який використовує тимчасові канали базового обладнання, для приклади кешей буфери DRAM або CPU твердження. З цих підходів пов'язане з нашою роботою в що він ідентифікує веб-сторінки на вкладках браузера, грунтуючись на час Javascript і класифікатор з використанням динамічного часу викривлення. Однак, оскільки атака залежить від процесора в якості каналу, це вимагає великого навантаження на всі ядра для моніторингу. Навпаки, наші атаки цикл подій браузера як канал, який можна контролювати шляхом включення одного події за раз. Це робить нашу атаку приховано і більш незалежно від виконання платформи. Наскільки нам відомо, ми спочатку монтуємо бічні атаки, які використовують архітектуру, керовану подіями веб-браузерів. Наша робота натхненна доказом того, концептуальна атака, яка краде секрет з кресторігіна веб-додаток, використовуючи однопоточні Javascript. Ми ідентифікуємо архітектуру, керовану подіями Chrome. як першопричину цієї атаки, і ми показуємо як це спостереження узагальнюється, в трьох різних атаках проти двох різних циклів подій в Chrome. Нарешті, центральне відмінність між класичним фінгерпрінт сайту і наша сторінка Ідентифікаційна атака - це модель противника. По-перше, наш противник потрібно тільки, щоб його сторінка відкривалася в браузер. По-друге, замість шаблонів трафіку в мережі, наш противник дотримується тільки тимчасові затримки в черзі подій браузера жертви. Ми вважаємо, що наші попередні результати з до 76% визнання з використанням одного єдиного зразка для навчання в закритому світі з 500 сторінками, можуть бути значно поліпшені розробка методів класифікації домена.

3 РОЗПОДІЛ РОЗШИРЕНЬ: АНАЛІЗ БЕЗПЕКИ ПОЛІТИКА УПРАВЛІННЯ РЕСУРСАМИ РОЗШИРЕНЬ БРАУЗЕРА

Всі основні веб-браузери підтримують розширення браузера щоб додавати нові функції і розширювати їх функціональні можливості. Проте, розширення браузера були метою кількох атак через їх тісних відносин з браузером. Як наслідок, розширення в минулому зловживали шкідливими завданнями, такими як конфіденційна інформація, перегляд історії пошуку або крадіжка паролів - що призвело до ряду серйозних атак. Незважаючи на те, що в крім безпечних розширень, тепер всі браузери реалізують захисні контрзаходи, які теоретично захищають розширення і їх ресурси від доступу третіх сторін. У в цьому документі ми представляємо дві атаки, які обходять цей контроль методів в кожному великому сімействі браузерів, що дозволяє перерахування атак на список встановлених розширень. Зокрема, ми представляємо тимчасовий бічний канал атака з настройками контролю доступу та атака який використовує погану практику програмування, впливаючи велика кількість розширень Safari. Через шкідливої природи наших висновків, ми також обговорюємо можливі контрзаходи проти наших власних атак і наші висновки і контрзаходи для різних учасників бере участь. Ми вважаємо, що наше дослідження може допомогти забезпечити впровадження і допомогти розробникам уникнути подібних нападу в майбутньому.

3.1 Налаштування контролю доступу

Найпопулярніший підхід для захисту ресурсів розширення від несанкціонованого доступу полягає в тому, щоб дозволити самі розширення визначають, які ресурси вони повинні бути закритими і які можуть публікуватися доступний. Всі браузери, які приймають це рішення, покладаються на набір параметрів конфігурації, включених в файл маніфесту який поставляється з кожним розширенням. З міркувань безпеки, за умовчанням всі ресурси вважаються закритими. Однак розробники можуть вказати в маніфесті список доступних ресурсів. Це рішення в даний час використовується всіма браузерами на Chromium, всі з яких засновані на Firefox і Microsoft Край.

Сімейство хромові. Сімейство Chromium включає всі версії Хром (наприклад, Google Chrome) і всі браузери заснований на двигуні Chromium (наприклад, Opera, Comodo Дракон і браузер Yandex). Розширення в цьому сімействі написані з використанням комбінації HTML, CSS і JavaScript . Вони є не потрібно використовувати будь-яку форму власного коду, так як замість цього випадку для плагінів або інших форм розширень браузера. Кожне розширення Chromium включає JSON файл з ім'ям manifest.json, який визначає набір властивостей такі як ім'я розширення, опис і версія номер (див. рис. 3.1 для прикладу маніфесту).

```
"name": "description",
"example": "Example extension",
"version": "1.0",

"browser_action": {
  "default_icon": "icon.png",
  "default_popup": "popup.html"},

"permissions": [
  "activeTab",
  "https://ajax.googleapis.com/"],

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

Рисунок 3.1 – Фрагмент розширення Chrome файл manifest.json

Маніфест використовується браузером для ознайомлення з функціональністю пропоновані розширенням і дозволами, необхідними для виконати ці дії.

У першій версії маніфесту не було ніяких обмежень над ресурсами доступних розширень з сторонніх сайтів. Через це різні були випущені інструменти, щоб скористатися цією слабкістю перерахувати розширення користувачів і використовувати їх вразливості. Щоб пом'якшити цю загрозу, Google вирішив ввести спеціальні настройки контролю доступу в другому версії файлу маніфесту. Це розширення використовує параметр (веб-доступні ресурси), щоб вказати шляхи упакованих ресурсів, які можуть використовуватися в контексті веб-сайту. Ресурси доступні через URL chrome-extension: // [extID] / [шлях]. Однак, будь-який доступ до навігації для

розширення або його ресурсів блокується браузером, якщо тільки розширення ресурс був раніше вказаний як доступний в його manifest.json. Це рішення було явно розроблено для мінімізації поверхні атаки, Конфіденційність.

Сімейство Firefox. Розширення сімейства Firefox (або надбудови, оскільки вони називаються на жаргоні Mozilla) може додавати нові функції в Інтернет браузера, змінити його поведінку, розширити GUI або взаємодіяти з вмістом веб-сайтів. У доповнень є доступ до потужного API під назвою XPCOM, який дозволяє використання декількох вбудованих сервісів і додатків через інтерфейс XPConnect. У сімействі Firefox (який включає, наприклад, Firefox Mobile, Iceweasel і Pale Moon), розширення записуються в комбінації мови інтерфейсу JavaScript і XML (XUL). Розширенням також дозволено використовувати функціональність від сторонніх довічних файлів або створювати свої власні виконавчі компоненти. Нещодавно Mozilla змінила свою розробку розширень з впровадженням SDK Add-on Проект JetPack. Цей комплект API високого рівня, полегшуючи процес розробки і деякі проблеми безпеки попереднього Firefox розширення. Реєстрація та розподіл різних розширень виконується через реєстр Chrome який також відповідає за настройку елементів призначеного для користувача інтерфейсу вікна програми, які не перебувають у вікнах область вмісту (наприклад, панелі інструментів, рядка меню, прогрес барів або вікон заголовків). Кожне розширення містить Файл chrome.manifest, який вказує параметри, пов'язані з три основні категорії - контент, мова і шкіра - як наведений в наступному фрагменті:

```
contentext                          src/content/
skin      ext   classic             src/skin/
localeext       en-US               src/locale/en-US/
content pck chrome/ext/pck          contentaccesible=yes
```

Як це було у випадку з розширеннями Chromium, з самого початку не було виконано жодного контролю для запобігання зовнішніх веб-сайтів від доступу до різних ресурсів розширення. А також в цьому випадку розробники вирішили вирішити проблему, включивши нову опцію в chrome.manifest (званий contentaccessible і зображено в останньому рядку попереднього прикладу), що вказує, які ресурси можуть публічно використовуватися. Однак, ресурси мають обмежений доступ за замовчуванням, якщо тільки contentaccessible = yes вказано в маніфесті. Firefox тепер розробляє новий спосіб обробки аддонів званих WebExtensions.

Ця технологія розроблена в основному для крос-браузерної сумісності, підтримки розширювальний API для хрому. Розширення портів між двома платформами будуть потрібні незначні зміни в код надбудови. Нові розширення також використовуйте manifest.json, включаючи деякі додаткові дані для Firefox (див. рис. 3.2). Щоб отримати доступ до різні ресурси розширення, Firefox буде використовувати moz-extension: // схема. Оскільки WebExtensions в даний час знаходяться на ранній стадії, ми не включають їх до наших тестів, але ми повідомили їх розробників.

Microsoft Edge Edge стане першим браузером Microsoft для повної підтримки розширення. Він буде слідувати розширення, сумісного з Chrome. модель на основі HTML, JavaScript і CSS. Ця означає, що процес міграції в Microsoft Edge для

```
"applications": {
  "gecko": {
    "id": "{the-addon-id}",
    "strict_min_version": "40.0.0",
    "strict_max_version": "50.*"
    "update_url": "https://foo/bar"
  }
} ...
```

Рисунок 3.2 – Фрагмент маніфесту Firefox WebExtension нові дані

Розробники розширення Chrome зажадають мінімальних зусиль. Крім загальних веб-API, спеціальне розширення API забезпечить більш глибоку інтеграцію з браузера, що дозволяє отримати доступ до таких функцій, як вкладки і вікна. У маніфесті бути названим manifest.json і буде використовувати той же Структура і загальні властивості JSON реалізація Хром. URL для доступу ресурси розширення слідують за ms-browser-ex-напруга: // [extID] / [шлях]. Оскільки проект знаходиться на попередніх етапах, і це не але в повній мірі працюючи, ми не включаємо його в наш аналіз.

3.1.1 Рандомізація URI

Оскільки Safari був одним з останніх великих браузерів для прийняття розширень, його розробники впровадили управління ресурсами з самого

початку, щоб уникнути перерахування або уразливості експлуатація встановлених розширень. Замість того, щоб покладатися в настройках, включених в файл маніфесту, як і всі інші основні браузери, розробники Apple прийняли URI рандомізації. У цьому рішенні немає відмінності між приватними або державними ресурсами, але замість цього базовий URI розширення довільно регенерується в кожної сесії. Розширення Safari кодуються з використанням комбінації HTML, CSS і JavaScript. Щоб взаємодіяти з мережею браузера і вмісту сторінки, надається JavaScript API і кожне розширення працює у своїй власній «пісочниці». Щоб розробити розширення, розробник повинен забезпечити: (i) глобальний код HTML-сторінки, (ii) контент (HTML, CSS, JavaScript-носії), (iii) пункти меню (мітки і зображення), (iv) код введених скриптів, (v) таблиці стилів і (vi) необхідні значки. Ці компоненти згруповані у дві категорії: перший з яких включає глобальну сторінку і пункти меню, а другий включає в себе контент і введений скриптів і таблиць стилів. Ця друга група не може отримати доступ будь-який ресурс в папці розширення, використовуючи відносний URL-адреси, як це робить перша група. Замість цього ці компоненти розширення повинні використовувати JavaScript на рис.3.3

```
1  <script type = "text/javascript">
2  var myImage = safari.extension.baseURI +
3  "Images/paper.jpg";
4  document.body.style.cssText =
5  "background-image: url(" +myImage+ ")";
6  </script>
```

Рисунок 3.3 – Приклад завантаження фонового зображення в CSS використовуючи абсолютні URL-адреси в розширенні Safari

для доступу до рандомізованого URI, який змінюється кожним разу Сафари запущений. Абсолютні URI зберігаються в safari.extension.baseURI.

3.2 Аналіз безпеки

У попередньому розділі ми представили два додаткових підходи, прийняті всіма основними родинами браузерів для захищати доступ до ресурсів розширення. Перше рішення спирається на URI загальнодоступного ресурсу, доступ до якого захищений централізованим кодом в браузері згідно

до налаштувань, зазначеним розробниками розширень в файл маніфесту. Друге рішення замінює централізоване перевіряйте шляхом рандомізації базового URI при кожному виконанні. У цьому випадку розширення повинно отримати доступ до власного ресурсів за допомогою спеціального Javascript API. Хоча їх дизайн абсолютно інша, обидва рішення забезпечити ті ж гарантії безпеки, атакуючому з перерахування встановлених розширень і доступ до їх ресурсів. Тепер ми розглянемо ці два підходи більш детально і обговорити два серйозних обмеження що часто підриває їхню безпеку. Це важливо відзначити, що ці атаки можуть також використовуватися в будь-якому типі пристрій з браузером з можливістю розширення, наприклад смартфонами або смартфонами.

3.2.1. Тимчасової фазовий канал контролю доступу

Перевірка параметрів Як уже згадувалося, переважна більшість браузерів прийняти централізований метод запобігання третіх сторін від

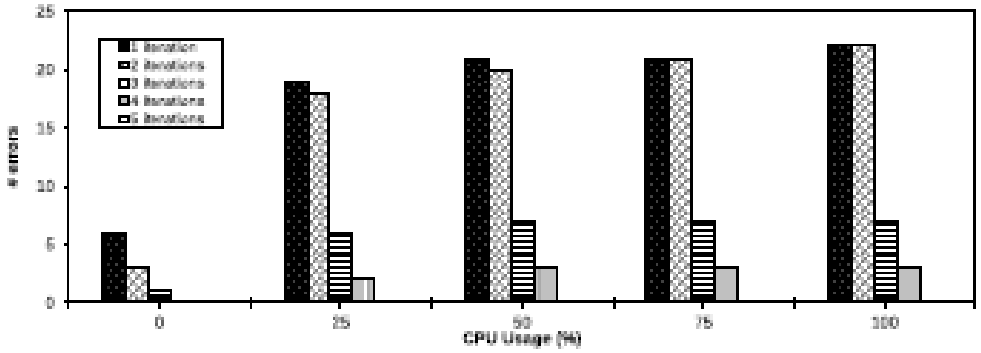

Рисунок 3.4 – Порівняння кількості ітерацій і помилок при використанні різних CPU (%)

доступу до будь-якого ресурсу розширень, які не відкрито позначені як загальнодоступні. Тому, коли веб-сайт намагається завантажити ресурс, відсутній в списку доступні ресурси, браузер заблокує запит. Незважаючи на те, що з точки зору дизайну це рішення може здатися безпечним, ми виявили, що всі їх реалізації страждають від серйозної проблеми, яка виникає з того факту, що ці браузери повинні виконувати дві різні перевірки: (i) перевірити, чи є яка -або розширення і (ii) отримати доступ до їх налаштувань управління для ідентифікації якщо запитаний ресурс є загальнодоступним (див. рис. 3.5

для простого логічного робочого процесу для забезпечення дотримання обробити). Якщо ця двоетапна перевірка невірна реалізований, він схильний до тимчасової бічній атаці каналу що противник може використовувати для визначення фактичних причин за відмовою запиту: розширення відсутнє або його ресурси залишаються конфіденційними. З цією метою ми використовували User Timing API1, реалізований в кожному великому браузері, для вимірювання продуктивності веб-додатків. Наприклад, зловмисник може кодувати кілька рядків Javascript для вимірювання часу відповіді при виклику підроблене розширення (див. Випадок A на рис. 3.5). Наприклад, в Chromium запитаний URI може виглядати так: це:

chrome-extension: // [fakeExtID] / [fakePath]

Потім зловмисник може генерувати другий запит вимірювати час відповіді при запиті розширення що насправді існує, але з використанням неіснуючого ресурсу шлях (приклад B на рис. 3.5):

chrome-extension: // [realExtID] / [fakePath]

Порівнюючи дві мітки часу, зловмисник може легко визначити, чи встановлено розширення чи ні в браузері. Подібне час відповіді означає, що центральний код підтвердження пройшов один і той же шлях виконання на два запити і, отже, розширення не встановлено в браузері. В іншому випадку значно відрізняються час виконання означає, що тільки другий тест не пройшов і, тому ви запросили розширення присутній в браузер. Ми провели експеримент для емпіричного налаштуйте поріг різниці в часі і кількість правильних запити, необхідні для забезпечення правильності нашої атаки. Зокрема, була використана наступна конфігурація:? Ми налаштували 5 різних застосувань ЦП: 0%, 25%, 50%, 75% і 100%. Експеримент виконаний на 2,4 ГГц Intel Core 2 Duo з 4 ГБ RAM товар комп'ютер. ? Атака була налаштована на повторення від 1 до 10 ітерацій. Зверніть увагу, що кожна ітерація виконує два дзвінки в браузер: той, який запитує підроблене розширення і той, який запитує фактичне розширення з підробленим шляхом. ? Ми повторювали кожне тестування атаки 500 раз, щоб уникнути будь-якої ухил. Таким чином, ми виконали: 2 дзвінка? 10 конфігурації ітерацій? 500 раз? 5 Використання процесора, в результаті чого загальне число 275 000 дзвінків. Ми помітили, що, коли шляху виконання були різними, часом відповіді відрізнялося більш ніж на 5%. Це важливо відзначити, що наш метод використовує пропорційний різницю між двома різними викликами замість використання попередньо обчисленого часу для конкретного пристрою. На рис. 3.4 показана точність в різних CPU навантаження і різну

кількість ітерацій. П'ять ітерацій були достатніми для досягнення 100% успіху навіть при 100% використанні ЦП.

Порушені браузери. Ми протестували нашу атаку по бічних каналах на два сімейства браузерів (засновані на хром і Firefox) які використовують настройки контролю доступу.

Наші експерименти підтверджують, що всі версії Ця вразливість порушена хромом. Браузери таких як Chrome, Opera, браузер Yandex (найбільший пошукова система в Росії) і браузер Comodo (найбільший емітент сертифікатів SSL) включені в цей група. Як уже згадувалося, ми не включаємо Edge і FirefoxWebExtensions, тому що вони все ще ранні етапи розвитку. Однак, оскільки вони йдуть тим же Механізм управління розширенням як Хром, вони також ймовірно, буде вразливий для нашої атаки по бічних каналах. Дивно, але не-WebExtensions в Firefox страждають від іншої помилки, яка ще простіше виявляє встановлених розширень. У браузері виникає виняток, якщо веб-сторінка запитує ресурс для невстановленого розширення (випадок A на рис. 3.5), але не в тому випадку, коли ресурс шлях не існує (випадок B на рис. 3.5). У той час виключення не викликає ніякого видимого ефекту на сторінці, зловмисник може просто инкапсулировать виклик в блок try-catch, щоб розрізняти два виконання шляхів і надійно перевірити наявність даного розширення.

Перелік розширень. Поділяючи дві централізовані перевірки, які є частиною перевірки правильності налаштувань розширення (або через бокового каналу або через поведінку різних виключень), можна повністю перерахувати всі встановлені розширення. Для зловмисника досить просто зонд в петлі всі існуючі розширення точно перерахуйте ті, які встановлені в системі. Для порівняння, попередні методи обходу були здатні виявляти лише невелика підмножина існуючих розширення. Щоб точно оцінити точність поліпшення в порівнянні з цими попередніми методами, ми провів експеримент по набору з 21 240 розширень. Для цього тесту ми вирішили зосередитися на двох браузерах з найбільшою кількістю доступних розширень: Chrome і Firefox (у Opera також є власний магазин розширень, але кількість популярних розширень дуже мало в порівнянні з іншими браузерами). У разі Chrome, розширення діляться на три групи: розширення, додатків та ігор. Хоча одна з груп явно звані розширеннями, всі вони встановлюються як chrome-extension і стежити за тим же контролем доступу модель налаштувань. На момент написання статті рекомендується кількість рекомендованих розширення в категорії ігор (найменша з трьох) – 3540

чоловік. Тому, щоб зберегти збалансований набір даних, ми вибрали також верхні 3540 останніх двох, що призводить до збалансованого набору даних з 10 620 найбільш рекомендовані розширення. Для Firefox процес вибору був простіше, тому що його магазин не робить відмінностей між різними категоріями. Тому ми вибрали 10 620 найпопулярніших Firefox розширень, щоб зберегти наш повний набір даних однаково збалансованим між двома браузерами. Щоб виміряти обхват попередніх методів обходу і порівняти його з повним охопленням нашого обходу техніки, ми об'єднали методи, описані в. Ці методи, наскільки нам відомо, є тільки ті, які існують, здатні перераховувати розширення шляхом підміни налаштувань контролю доступу. Ці методи заснований на перевірці наявності доступного ззовні ресурсів в розширеннях. Щоб перевірити їх, ми проаналізували файли маніфесту всіх розширень, які ми скачали, дивлячись для будь-яких доступних ресурсів. Таблиця 1 показує отриманий охоплення з використанням попереднього методи. Розширення Chrome були простіше перераховувати ніж ті, які зберігаються в магазині Firefox. Однак покриття цих старих методів дуже мало в порівнянні з повним охоплення, отриманий нашим методом.

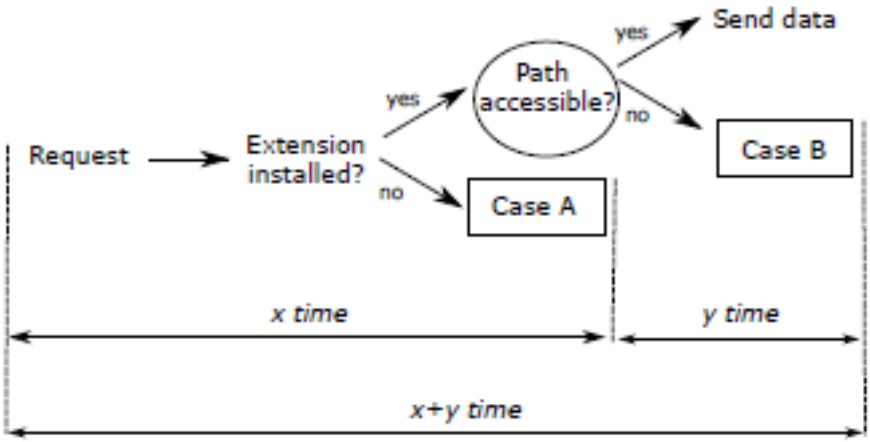

Рисунок 3.5 – Схема керування доступом до ресурсів

3.2.2 Витік URI

Навіть якщо управління рандомизацией URI повністю централізовано, це сильно залежить від розробників, щоб зберегти ресурси від доступу третіх сторін. Фактично, розширення часто використовуються для введення додаткового контенту, елементів управління або просто попереджайте панелі

на веб-сайті. Це новостворене контент може ненавмисно витік випадкового розширення URI, тим самим минаючи заходи контролю безпеки і відкриття доступу до всіх ресурсів іншої код працює на тій же сторінці. На додаток пропущений випадковий URI може бути використаний третіми сторонами для однозначного ідентифікувати користувача під час перегляду під час тієї ж сесії. Простий приклад, взятий з розширення Web Trust2 показаний на рис. 3.6. Фрагмент коду створює новий iframe (рядок # 11), встановлює свій атрибут src для baseURI випадковий адресу розширення (рядок #14) і додає рамку в тіло документа (рядок # 19). В результаті, будь-який інший код JavaScript, що працює на одній сторінці (і, отже, потенційно під контролем зловмисника) може отримати адресу введеного iframe і використовувати щоб отримати доступ до будь-якого ресурсу розширення. Фактично, одного разу випадковий токен відомий, браузер не пропонує ніяких інших.

```
1  wot.rating = {
2  toggleframe: function(id, file, style){
3   try {
4     var frame = document.getElementById(
          id);
5     if (frame) {
6      frame.parentNode.removeChild(frame);
7      return true;
8     } else {
9      var body = document.
           getElementsByTagName("body");
10     if (body && body.length) {
11      frame = document.createElement("
           iframe");
12      if (frame) {
13       frame.src = safari.extension.
            baseURI+file;
14       frame.setAttribute("id", id);
15       frame.setAttribute("style", style)
            ;
16       if (body[0].appendChild(frame))
17        {return true;}
18     }
19    }
20   }
21  } catch (e) {
22    console.log("failed with"+e+"\n");}
23  return false;
24 }
```

Рисунок 3.6 – Функція розширення веб-довіри Trust Safari, яка створює iframe на веб-сайті з baseURI random змінна як джерело

Оцінка масштабів проблеми Приклад використання веб-довіри, розглянутий вище, складається з єдина функція з 30 рядків коду, але не у всіх випадках настільки очевидні для ідентифікації без складного статичного аналізу розширення. Щоб оцінити, наскільки поширена проблема, ми реалізували аналізатор прототипів, який повідомляє про випадки кандидатів URI-витоку у всіх розширеннях Safari.

Abstract Syntax Tree (AST) всіх різних JavaScript компоненти аналізованого розширення. Джерело і раковини розташовані, просто шукає конкретний код у вузли дерева, в той час як інформаційний потік обчислюється слідуючи різним фрагментам коду, які насправді мати доступ до даних по іншому виконанню шляху. Зокрема, аналіз виконується в трьох кроки:

1) На першому етапі інструмент ідентифікує розташування джерела де код звертається до довільного розширення URI (пошук викликів методу baseURI).

2) Інструмент потім окремо аналізує всі компоненти який може використовувати отримане значення. Після потік інформації (тобто функції, які називаються або викликають), цей процес виконується рекурсивно поки не буде знайдено ніяких з'єднань.

3) Для кожного ідентифікованого компонента інструмент знаходить раковини, тобто місце розташування, в яке вводиться новий контент на веб-сторінці (наприклад, через елемент createElement і appendChild). Якщо є з'єднання між доступом базиURI і ін'єкцією елемента на веб-сайті, розширення позначені як підозрілі і повідомляються для подальшого аналізу.

Схема на рис. 3.7 показує спрощений приклад розширення, яке витоку baseURI з використанням функції A файла A для отримання значення, функція B файлу B як проміжну фазу, а функцію D файлу D - нарешті, зробити ін'єкції на сайті. Цей метод призначений для використання в якості фільтруючого фільтра і НЕ як точний метод виявлення. Дійсно, факт що розширення витягує baseURI, а потім використовує його створити певний контент недостатньо, щоб визначити, повна інформація фактично протікає. Наприклад,

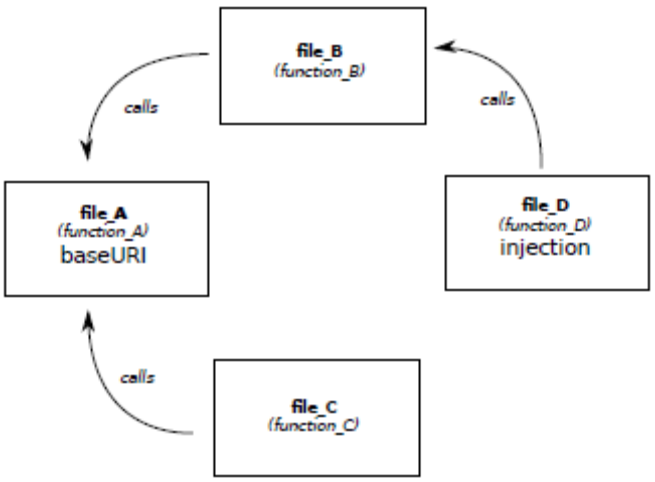

Рисунок 3.7– Спрощена схема прикладу розширення що витікає baseURI

знайдено розширення, яке використовувало baseURI, щоб отримати його номер версії, а потім ін'єкційний iframe з Номер версії включений безпосередньо як частина його URL, але без витоку повної базиURI.

Щоб оцінити наш інструмент, ми скачали та проаналізували всі доступні розширення в розширенні Safari Gallery4. Розширения 718 були 15 різними (наприклад, безпека, покупки, новини, соціальні мережі, і інструменти пошуку). На рис.3.8 приведены отримані результати. В цілому, більш ніж 40% галерей розширення Safari були потенційно уразливими для нашої перерахування методів. Мы решили вручну проаналізувати деякі результати для визначення дійсних або заявлених розширень дійсно виконані витоку або ні. Оскільки категорія безпеки входить в число з найбільшим відсотком розширень з потенціалом відхилень, а також особливо чутливою за тип інформації, до якої ці розширення зазвичай відносяться (наприклад, паролі користувачів), ми вирішили вручну перевірити всі результати для розширень у цій категорії. З великими зусиллями ми виконали вичерпний перегляд коду вручну всіх розширень безпеки, вибір тих, які були повністю функціональними, виключаючи ті, що вимагали оплати за свої послуги. серед 68 розширений в цій групі, 29 були відзначені як підозрілі утечки і 39 не протікали. Из подозрительных, 20 из 29 на самом деле просочилась секретная базаURI. Крім того, ми ідентифікували один помилковий негатив, який просочився в інформації, але не був ідентифікований нашим інструментом статичного аналізу. В зокрема, це розширення отримало повний URL-адреса, включаючи baseURI, але зберег його локально. В межах розширень, які є вразливими для нашої атаки, ми знайшли розширення, такі як Adblock5, Ghostery6, Web Of Trust7 та Adguard8. Список також включає в себе пароль менеджерів, таких як LastPass9, Dashline10, Keeper11 і TeedyID12 і комбінації двох функціональних можливостей (наприклад, Blur з Abine13). Таким чином, відповідна кількість розширень Safari уразливих для нашої техніки, включаючи кілька важливих і дуже популярних пов'язаних з безпекою розширення. Як пояснили в розділі 3.5, ми зараз знаходимося в процесі перевірки всіх результатів і контактів з розробниками зазначених розширень для виправлення їх коду.

Category	# Total Ext.	# P. Leak
Shopping	95	57.89%
Email	13	53.85%
Security	84	52.38%
News	20	45.00%
Photos	25	44.00%
Bookmarking	61	42.62%
Productivity	147	40.82%
RSStools	5	40.00%
Entertainment	37	37.84%
Translation	8	37.50%
Social	80	30.00%
Developer	57	29.82%
Other	42	26.19%
Search	42	21.43%
urlshorteners	5	0.00%
Total	*721*	*40.50%*

Рисунок 3.8 – Відсоток потенційної витоку baseURI в сафарі розширення

3.3 Вплив

У попередньому розділі ми обговорили безпеку налаштувань контролю доступу та рандомізації URI і ми показали, як кожна механізм, браузери можна легко обійти в практиці. Є кілька можливих наслідків зловживання інформацією, наданою нашими двома методами.

3.3.1 Фингерпринтинг и аналитика

Сама точна і протилежна форма відбитка пальця спрямована на створення унікального ідентифікатора для кожного користувача пристрою, такого як Panopticlick. Він вважається пасажиром техніки, оскільки для створення та унікального ідентифікатора ці методи не вимагають зберігання щось на машині користувача (в відмінності від стану таких, як Cookies). Щоб створити унікальний ідентифікатор, кілька функцій вилучаються з комп'ютера користувача та об'єднуються в унікальний відбиток пальця. Цей процес може повторюватися на декількох веб-сайтах, а ідентифікатор завжди буде однаковим для однієї і тієї ж комп'ютера, дозволяючи трекам визначити історію перегляду користувачів, серед інших завдань. Використання набору встановлених розширень може збільшити унікальність отриманого відбитка пальця. Щоб виміряти точну силу відбитків пальців перерахування, дослідження повинно бути виконане для вимірювання дискримінаційної сили найпопулярніших розширень, доступного для кожного браузера. З цією метою ми провели попереднє дослідження такого роду аналізу в розділі 3.3.4. Методи, запропоновані в цій статті, також можуть бути використані для

виконання абсолютно точної броузера, друк без перевірки User-Agent. До цього конца наш метод можна використовувати для перевірки вбудованих розширень. Ці розширення попередньо встановлені та представлені майже кожним великим веб-браузером, і немає можливості для користувача, щоб їх видалити. Тому, якщо ми перевіряємо методи одного з цих вбудованих розширень, які не існують в інших браузерах, веб-сайт може точно ідентифікувати сімейство браузерів з 100% точністю. Установленное перечисление розширений в комбінації з вищесказаною ідентифікацією браузера може бути використана для визначити демографію користувачів. Розширення, які використання певних користувачів може бути легко виявлено на веб-сайтах або сторонніх послуг. Встановлені розширення інформація про конкретні інтереси користувача, проблеми, і звички перегляду. Наприклад, користувачі з безпекою і розширення конфіденційності, встановлені в їх браузерах, такі як Ghostery або PrivacyBadger потенційно більш обізнані про їх конфіденційності, ніж інші користувачі. Те ж саме відбувається з персоналізацією розширень, ігор або будь-яких можливих комбінації інших категорій розширень. Щоб виміряти можливість виконання аналітики через розширень, ми провели тест на доказ-концепцію в розділі 3.3.4.

3.3.2 Шкідливі програми

Інформація, отримана з встановлених розширень також може використовуватися для шкідливих цілей, оскільки інформація етап збору інформації про потенційних жертв зазвичай перший крок для цілеспрямованої атаки. Наприклад, зловмисники можуть вводити код перерахування розширення в скомпрометований веб-сайт і пошук користувачів з покупками розширень управління і менеджерів паролів для вниз по їх поверхні атаки тільки тим користувачам, чиї інформація про кредитну картку має більш високу ймовірність бути вкрадений. Інша можливість полягала б у виявленні присутності розширення основного постачальника антивіруса для персоналізації комплект експлойта або вирішити, чи є шкідлива корисне навантаження повинен бути доставлений або не визначений певному користувачеві. На додаток до вже представленим атакам, в недавньому робота, Buyukkayhan et al. представив CrossFire, метод, що дозволяє зловмисникові здійснювати шкідливі дії використовуючи законні розширення. Частина, яка залишилася без відповіді на папір, як

зловмисник може ідентифікувати набір встановлених розширень для використання в її цілях. Від використовуючи наш метод перерахування, зловмисник може створити повністю функціональні шкідливі розширення, знаючи всі встановлені розширення жертви завчасно. Через мінливості можливих розширень інформація конкретного користувача можуть бути використані в різних соціальні атаки (автоматизовані чи ні). Наприклад, шкідливий веб-сайт може використовувати інформацію про конкретних розширення, встановлені для уособлення і підробки законні повідомлення про ці розширеннях, з наміром обдурити користувача і змусити її встановити шкідливе ПЗ. Наприклад, якщо шкідливий веб-сайт виявляє, що користувач використовує конкретний пароль розширення управління, він може створити підроблене вікно для попросіть користувача повторно ввести пароль. Ця атака особливо серйозний у випадку з Safari, оскільки зловмисник може фактично отримати доступ до всіх ресурсів розширення, яке протікає його baseURI. Отже, навіть обережний користувач, який вирішує аналіз джерела веб-сайту не може легко зрозуміти, якщо певне вікно або кадр створюється встановленим розширенням або сайтом, повторно використовує ресурси розширення. У той час як обхід управління рандомизацией URI не надав повної можливість перерахування, коли розширення витік його випадкового токена, він відкриває всі свої внутрішні ресурси зловмисникові. Це потенційно дуже шкідливо так як він збільшує поверхню атаки, дозволяючи зловмисникові доступ і використання будь-уразливості в одному з внутрішніх компоненти розширення. Наприклад, Котович і Осборн представив версію розширення Chrome рамкі14, які могли б використовуватися, коли це було можливо для доступу до всіх різних ресурсів розширення.

3.3.4 Вивчення життєздатності

Ми вивчили життєздатність оціненого впливу для деяких з розглянутих раніше випадків. Зокрема, ми проаналізували їх потенціал для виконання аналітики а також можливості відбитків пальців для розширень. Ми опустили шкідливі тематичні дослідження через їх властиві етичних проблем. Крім того, ми вважаємо, що їх реалізація простіша, ніж в які ми тестували і оцінювали.

У разі можливостей розширень для аналітики ми вирахували популярність різних категорій встановлених в Інтернет-магазині Chrome для

кожного з розширень що ми раніше аналізували в розділі 3.2.1. Зокрема, ми проаналізували 63 категорії, присутні в 10 620 найпопулярніші розширення Chrome. Найпопулярнішою категорією була «продуктивність» з 29,90% використання. Проте, визначення цієї категорії неясно, тому що він включає в себе широкий діапазон типів розширень, таких як блокувальники реклами, планувальники або службові інструменти. У всякому разі, можлива подкатегорізація може можливо за допомогою доступного опису кожного розширення. Решта 10 найпопулярніших є точніше і можуть бути корисні для виконання аналітичних такі завдання, як цільова реклама або персоналізація веб-сайту. Наприклад, кількість відвідувачів з «Шопінг», «бізнес-процеси,» або «спортивні» розширення, може допомогти власнику веб-сайту персоналізувати свій контент або відповідно, поліпшивши кількість відвідувачів або доходи від реклами. Однак не тільки найпопулярніші розширення можуть допоможіть власнику веб-сайту краще зрозуміти її відвідувачів і діяти відповідно. Дійсно, менш популярні розширення, тому що їх більше високий ступінь дискримінації серед користувачів, також можуть бути використані для цієї задачі. Наприклад, використання розширень з категорії «креативні інструменти» вказує, що відвідувач схильний створювати контент, наявність розширень в рамках «академічних ресурсів» ймовірно, вкаже, що відвідувач знаходиться поруч з академічне середовище, «інструменти вчителя» можуть означати, що відвідувач доставляє по крайней заходу кілька лекцій і «блогів», має на увазі, що відвідувач є блогером. Таким чином, ми вважаємо, що розширення є потужними інструмент для виконання дрібнозернистої користувальницької аналітики через їх різноманіття. Крім того, інформація, отримана з установлення розширення веб-відвідувача в поєднанні з класична інформація аналітики може привести до кращого для користувача аналітика для власників веб-сайтів.

Фінгерпрінт пристрої. Щоб зрозуміти і виміряти можливості розширень для відбитків пальців пристрою, ми впровадили сторінка, яка перевіряє встановлені користувачем розширення топ-1000 найбільш популярних з Інтернет-магазину Chrome і веб-сайти Add-ons Firefox, використовуючи тимчасової канал каналу описаної в розділі 3.2.1. Оскільки наше дослідження включало перерахування кількох встановлених користувачами, ми інформували користувачів про процедура, включаючи зібрану інформацію. Тільки після того, як користувач погодиться виконати експеримент і поділитися зібрана інформація, перерахування її розширень.

Ми також встановлюємо cookie для користувача браузера, щоб запобігти повторну повторну передачу з одного і того ж користувач. Крім того, для захисту конфіденційності користувачів ми збираємо тільки анонімні дані. Ми поширюємо URL-адресу сторінки через соціальну мереж і друзів, попросивши їх взяти участь в вивчення і подальшого поширення контакти. Таким чином, ми зібрали список встановлених розширень з 204 учасників з 16 різних країн. Хоча це число менше, ніж в попередньому ми хотіли б відзначити, що відбитки пальців це не справжня мета статті, а просто можливе застосування наших атак. Фактично, цей аналіз просто призначені для визначення життєздатності нашої техніки для відбитки пальців пристрою, або як метод сам по собі, або за допомогою доповнюючи інші існуючі технології відбитків пальців. Дотримуючись стандарту, прийнятому в попередніх роботах, ми проаналізували безлічі анонімності розширення користувачі з відбитками пальців, які визначаються як кількість користувачів з тим же самим відбитком пальця, тобто з тим же набором розширень (розподіл наборів анонімності показано на рис. 3.9). В цілому, з 204 користувачів, які брали участь в нашому дослідженні, 116 користувачів представили унікальний набір встановлених розширень, що означає, що 56,86% учасників однозначно можна ідентифікувати, просто використовуючи їх набір розширень. Крім того, ми також порівнюємо дискримінаційний рівень цього методу докази відбитків обчислюючи нормовану ентропію Шеннона і порівнюючи його з іншими атрибутами відбитків пальців в попередніх дослідженнях. Зокрема, рис. 3.10 порівнює різні значення ентропії в шести верхніх відбитках пальців методи або атрибути, виміряні в роботі Laperdrix et al. з нашої внутрішньої печаткою на основі розширень метод. Ми можемо помітити, що розширення

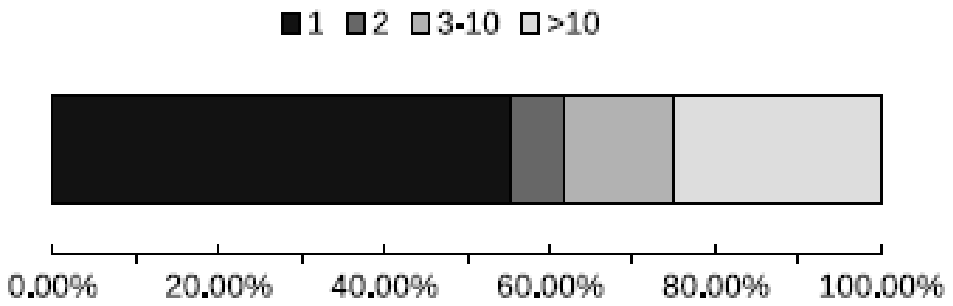

Рис. 3.9 – Розподіл обсягів анонімності щодо розширення

Method	Entropy
Extensions	0.869
List of Plugins	0.718
List of Fonts	0.548
User Agent	0.550
Canvas	0.475
Content Language	0.344
Screen Resolution	0.263

Рисунок 3.10 – Порівняння між розширеннями з іншими відбитками пальців Атрибути

надіслав найвищу ентропію аналізованого відбитка пальця атрибути - роблячи їх більш точними, ніж використання список шрифтів або технології на полотні.

3.4 Розкриття інформації про уразливість і Контрзаходи

3.4.1 Освітлення, ефекти атаки

Представлено два різних класи атак проти політики управління ресурсами, прийнятої всіма родинами браузерів на ринку. На рис. 3.11 загальний вплив наших методів. Як уже згадувалося, охоплення нашого перерахування атака завершена в разі бокового каналу синхронізації атака на сімейства браузерів на основі контролю доступу (т. Е. Сімейства Chromium і Firefox), в той час як приблизно близько 40% в браузерах ранжирування URL-адрес (Safari).

Ефекти приватного режиму. «Інкогніто» або приватний режим присутні в більшості сучасних браузерів і захищають і обмежують кілька доступів до ресурсів браузера, таким як файли cookie або перегляд історія. Тому ми вирішили проаналізувати, якщо наші атаки може перераховувати розширення, навіть якщо цей режим активується. Ми виявили, що всі наші атаки точно ідентифіковані список встановлених розширень також в межах приватного Режим. Цей факт пояснюється декількома причинами. У разі Браузер сімейства Chromium, браузер перевіряє наявність розширень в

режимі інкогніто, хоча розширення не дозволений доступ до веб-сайтів. Firefox і Safari зробили не включають перевірку на наявність розширень і, отже, обидва сайти і розширення мають доступ до один одному.

Browser	Extensions Enumeration	Resource Access
Chromium Family	✓	
– Chrome	✓	
– Opera	✓	
– Yandex	✓	
...	✓	
Firefox Family	✓	
– Firefox Mobile	✓	
– Iceweasel	✓	
– Pale Moon	✓	
...	✓	
Safari	≤ 40%	≤ 40%
Microsoft Edge	in discussion	
Firefox WebExtensions	in discussion	

Рисунок 3.11 – Поточні браузери, порушені нашими атаками. останні два рядки належать до розширень, які все ще знаходяться в розробці

3.4.2 Тимчасова атака бокового каналу

Перший клас атак є наслідком реалізація налаштувань управління доступом до браузеру: Використання розширень браузерів Firefox може бути використано визнати причину невдалої резолюції, і канал хроматичної синхронізації Chromium дозволяє зловмисникові точно розповісти про всі двох індивідуальних перевірках виконується движком браузера. Наслідком цього в обох випадках є досконала техніка для перерахування всіх розширень, встановлених користувачем. З огляду на природу цих двох браузерів з відкритим вихідним кодом, ми вручну ідентифікували функції, відповідальні за проблему і вказав, як виправити кожен з них.

Сімейство хромові. Ми зв'язалися з командою Chromium, щоб повідомити про терміни проблема. Розробники були дуже здивовані атаки, оскільки вони вважали, що тимчасові різниці в контрольна фаза не була достатньо значною, щоб цей тип побічної атаки часу. Перевіряючи функція, відповідальна за перевірку доступності конкретний шлях розширення (див. рис. 3.12), два різних кроки, описані в розділі 3, можуть бути чітко визначені.

По-перше, браузер перевіряє наявність розширення (рядок # 4) і закінчується, якщо розширення не існує. Якщо розширення існує, воно виконує різні перевірки щоб переконатися, що шлях доступний, повертаючи помилку повідомлення якщо немає. Ці перевірки дозволяють різницю в часі, використовувану в атаці. Ми запропонували можливий спосіб виправити код, щоб уникнути вимір часу шляхом зміни управління розширенням механізм комбінування внутрішньої перевірки розширення і перевірка ресурсів разом в одному атомному т.е. шляхом зміни існування продовження перевірка рядка. Це вимагає заміни розширення список з хеш-таблицею, що містить розширення і повний шлях їх ресурсів. Хоча може здатися простим вирішити проблему, контрольний атом, проблема залишається, якщо атака виконується за допомогою реальних шляхів розширення (легкодоступних) а не підроблені шляху. Різниця в часі була б такою ж, як той, який представлений на рис. 3.5, з єдиною різницею що перша перевірка буде перевіряти повний шлях а не тільки розширення. На момент написання, так як це проблема з дизайном, вона як і раніше не виправлена. Крім того, оскільки нові веб-додатки Firefox і Microsoft Edge (обидва знаходяться на ранніх стадіях) використовувати ті ж механізми контролю за Chromium, ми також повідомили їх розробників про те, щоб зробити їм відомо про проблему, описаної в цьому документі. Ми сподіваємося що наші зусилля допоможуть цим двом новим версіями інтегрувати розробити необхідні контрзаходи, щоб уникнути ці проблеми безпеки з самого початку.

Сімейство Firefox. Ми також відповідально повідомили про те, Проблема з WebExtensions, яка робить наше перерахування атака можлива для її розробників, які визнали і в даний час обговорюють, як діяти. Зокрема, на рис. 3.13 показано функція, яка викликає відгуку щодо існування продовження. Помилка повертається, коли шлях ресурсу не існує (рядок №4 і рядок № 12 на рис. 3.13) не піднімає виняток. Отже, рішення простий: return помилка NS ERROR DOM BAD URI (т. е. Одна і та ж який викликається, коли розширення не встановлено). Ця виправлення не викличе проблем з веб-сайтами з використанням розширення шляхів, зберігаючи функціональність без змін. Що стосується WebExtensions, розробники Firefox недавно змінили спосіб доступу до розширень в для вирішення проблеми бокового каналу і другіхсвязанних з атаками. Зокрема, вони змінили початкову схема (moz-extension: // [extID] / [шлях]) до Moc-розширення: // [випадковий UUID] / [шлях]. На жаль, в той час як ця зміна дійсно ускладнює для перерахування призначених для користувача розширень він вводить набагато більше

небезпечна проблема. Фактично маркер випадкового UUID може тепер можна використовувати для точного відбитку пальців користувачів, якщо він просочився за допомогою розширень. Веб-сайт може отримати цей UUID і використовуйте його, щоб однозначно ідентифікувати користувача, як тільки він згенерований випадковий ідентифікатор ніколи не змінюється. Ми повідомили про це помилка для розробників Firefox.

```
3  const Extension* extension=
       RendererExtensionRegistry::Get()->
       GetExtensionOrAppByURL(resource_url)
       ;
4  if (!extension) {
5    return true;
6  }
```

Рисунок 3.12 – Неможливо виконати команду політики запиту ресурсів Хром, який викликає різницю між існуючими і не існуючі розширення

```
1  GetFlagsFromPackage(const nsCString&
       aPackage,uint32_t* aFlags){
2    PackageEntry* entry;
3    if (!mPackagesHash.Get(aPackage, &
         entry))
4      return NS_ERROR_FILE_NOT_FOUND;
5    *aFlags = entry->flags;
6    return NS_OK;
7  }
8
9  GetSubstitutionInternal(const nsACString
       & root, nsIURI **result){
10     nsAutoCString uri;
11     if (!ResolveSpecialCases(root,
           NS_LITERAL_CSTRING("/"), uri)) {
12         return NS_ERROR_NOT_AVAILABLE;}
13     return NS_NewURI(result, uri);
14 }
```

Рисунок 3.13 – Функції Firefox, які викликають різницю між існуючих і не існуючих розширень

3.4.3 Витік URI

Другий клас атак, представлених в досить різний. Фактично, метод розширення Safari контроль використовує для забезпечення належної доступності ресурсів, в принципі, є правильним. Однак, Safari делегатам розробників розширень відповідальність щоб зберегти секретний секретний код URI. Ми вважаємо, що це дуже ризиковане рішення, оскільки більшості

розробників не вистачає правильне розуміння проблеми. Як наслідок, наші експерименти підтверджують, що відповідне число (40% в наші попередні експерименти) розширень, ймовірно, для витоку baseURI, підриваючи всі рішення безпеки. Зокрема, ми виявили, що важлива безпека розширення, такі як кілька менеджерів паролів або блокувальники реклами страждають від цієї базиURI-витоку уразливості і, отже, вони уразливі для цього атака. У разі розширень безпеки це особливо важливо турбуючись через типу інформації, яку вони управляють зазвичай дуже чутлива. У цьому випадку проблему ще важче вирішити, тому що це не є наслідком помилки в розширенні але сотні помилок, які розповсюджуються по різних розширень. Отримання і навчання всіх розробників розширень це складне завдання, але Apple повинна надати більше інформація про належне способі обробки baseURI і про наслідки цього процесу в плані безпеки. Крім того, ми вважаємо, що Safari може отримати вигоду з приймаючи легке статичне аналітичне рішення (аналогічне до тієї, яку ми обговорюємо в розділі 3.2) для аналізу розширень на своєму ринку і відзначте ті, які просочуються випадковий токен. Це дозволить відразу визначити потенційно витоку, які можуть зажадати більш точні ручна перевірка. Тим часом ми почали повідомляти проблема з деякими розширеннями безпеки ми вже підтверджено вручну, щоб допомогти їм вирішити проблему витоку URI проблема.

3.4.4 Пропозиція щодо продовження терміну дії

Щоб поліпшити безпеку і конфіденційність браузера розширень, ми пропонуємо рішення, яке вирішує всі різні проблеми, представлені в цій статті.

1) Всі браузери повинні слідувати схемі розширення який включає випадкове сгенерированное значення в URL-адресу: X-розширення: // [randomValue] / [шлях]. Ця випадкове значення має бути змінене протягом і під час тієї ж сесії і повинна бути незалежною для кожного розширення встановлено. Наприклад, готелі повинен змінити його в кожному розширенні в кожному доступі. Таким чином, випадкове значення не може використовуватися для відбитка пальця користувачі.

2) Браузери також повинні здійснювати контроль доступу (наприклад, доступний в Інтернеті ресурс), щоб уникнути небажаного доступ до всіх ресурсів розширень навіть коли випадкове значення ненавмисно просочується.

3) Розширення повинні бути проаналізовані на предмет можливих витоків перш ніж зробити їх загальнодоступними для користувачів. Більш того, керівництва для розробників повинні проблеми, які можуть викликати витік будь-якого випадкового генерується значення.

3.5 Пов'язана робота

Безпека розширень браузера Дослідницьке співтовариство зробило велику кількість вкладів, які аналізують властивості безпеки розширення браузерів. Ряд недавніх досліджень сфокусовано на моніторинг виконання браузера під час виконання розширення. Louw et al. запропонували цілісність checker і політичне виконання для застарілих розширень Firefox. Пізніші рамки, Sentinel, забезпечив дрібномасштабний контроль над користувачами в порівнянні зі спадщиною розширень, дозволяючи їм визначати настроюються політики безпеки блокуючи загальні атаки для цих розширень. Інші підходи були спрямовані на забезпечення безпеки аналіз розширень браузерів для виявлення безпеки недоліки. На стороні статичного аналізу IBEX структуру для аналізу властивостей безпеки за допомогою статичної методології, а також дозволяє розробникам створювати дрібномасштабні політики контролю доступу та потоку даних. VEX [1] замість цього статичний аналізатор для Firefox Розширення JavaScript, які застосовують аналіз потоку інформації для виявлення вразливостей розширення браузера. Аналіз динамічних розширень включає в себе роботу Джеріко і ін., в якій автори запропонували використовувати динамічного аналізу для відстеження даних всередині браузера і виявляти шкідливі розширення. Dhawan et al. аналогічний підхід для виявлення розширень, які скомпрометовані середовища браузера. У тому ж дусі, Wang et al. використовував інструментальний браузер для аналізу Розширення Firefox. Халк - динамічний аналіз структура, яка контролює активність перегляду розширень, що використовують методи пухнастості і Honey- Сторінки, адаптовані до розширень. Халк був використаний для аналізу більше 48 000 розширень Chrome, відкриття кілька шкідливих. Незважаючи на те, що ці підходи корисні для виявлення шкідливих або скомпрометованих розширень, вони, на жаль, марно проти зовнішніх атак або інформації протікання. Наш аналіз привів до найбільш повного набір атак на контроль доступності ресурсів і baseURI, дозволяючи в обох випадках розширення перерахування, які можуть бути використані як частина великі загрози. Подібно нашої

власної роботи, XHOUND недавно показав, що розширення змін виконуються на DOM досить для перерахування розширень. Використовуючи цю техніку, автори також розробили новий відбиток пристрої техніки та виміряв її вплив. Однак, цей підхід має набагато більш обмежену застосовність. У порівняння, наші методи досягають більшого охоплення, успішно перераховуючи 100% розширень для доступу контрольних браузерів і близько 40% для тих, хто використовує Рандомизация URI.

Веб-тайм-атаки.Атаки веб-синхронізації використовувалися для різних цілей, як на стороні клієнта, так і на стороні сервера. Фельтен і Шнайдер [3] представив цей тип атак як інструмент для компрометації особистих даних користувачів і, зокрема, їх історію веб-перегляду. Таким чином, зловмисник може отримати цю інформацію, використовуючи різні форми методів кешування веб-браузера. Від вимір часу, необхідного для доступу до певних даних з не пов'язана веб-сайт, дослідники могли б визначити, якщо що конкретні дані були кешованими чи ні, що вказує на попередню доступ. Пізніше Бортц і ін. [2] організовані тимчасові атаки в два різні типи атак: (i) пряма синхронізація, що складається в вимір різниці часу в HTTP-запити на веб-сайти і (ii) міжсайтового час, яке дозволяє отримати дані з клієнтської сторони . Перший тип може виявити дані веб-сайту, які можуть бути використані для підтвердження дійсності ім'я користувача на певному захищеному веб-сайті. Другий тип атаки дотримуються тієї ж роботу попередньої роботи Фельтеном і Шнайдером. Вони також провели кілька експериментів що було встановлено, що ці тимчасові уразливості були більш поширені, ніж очікувалося. Крім того, Котчер і інші. [5] виявив, що крім атак попередніх обговорювалося використання фільтрів CSS виявлення чутливої інформації, такої як текстові жетони використовуючи тимчасові відмінності для візуалізації різних дерев DOM. Два останніх дослідження показують, що ці атаки далекі від вирішення. Jia et al. [4] проаналізували можливість визначення географічних місць розташування користувачів до налаштування послуг, які виконуються веб-сайтами. Вміст, чутливе до місця розташування, кешируєтся так само, як і будь-який інший контент. Тому зловмисник може визначити місце жертви, перевіривши цей конкретний даних і не покладаючись ні на яку іншу техніку. Крім, Van Goethem et al. [8] запропонували нові методи синхронізації заснований на оцінці розміру ресурсів перехресного походження. Оскільки вимірювання починається після ресурсів завантажуються, він не страждає від несприятливої мережі умови.

Дослідження також показує, що ці атаки може використовуватися на різних платформах, збільшуючи атаку поверхні і числа потенційних жертв. Однак жодна з цих методик синхронізації не була раніше використовувалися для ідентифікації компонентів мережі браузера. Наші нові атак з бічним каналом перші атаки, здатні визначати з точністю 100% які розширення встановлені в браузері, незалежно використання ЦП.

4 ВИСНОВКИ

У даній роботі були розглянуті два актуальних методу для використання вразливостей браузера: Атаки на загальних циклах подій, використання EDP і Керувати розширеннями браузера, контроль їх ресурсів.

При виконанні даної роботи були розглянуті статті про вразливості, в яких згадується, що важливим критерієм захисту браузера є його оновлення, так як творці браузера, намагаються захистити користувача від зловмисників.

Були вивчені програмні забезпечення комерційних виробників. Знайдено такі проблеми як некоректне видалення всіх паролів, проблеми з функціоналом розширень браузера.

При створенні програмного забезпечення були дотримані вище перераховані проблеми. Створена програма є аналогом з доданими функціями.

Надалі програма буде розвиватися і будуть додано більшу кількість функціоналу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bandhakavi, S., Tiku, N., Pittman, W., King, S. T., Madhusudan, P., and Winslett, M. Vetting browser extensions for security vulnerabilities with vex. communications of the ACM 54, 9, 2011. – pp. 91-99.
2. Bortz, A., and Boneh, D. Exposing private information by timing web applications. In Proceedings of the 16th international conference on World Wide Web (WWW), 2007. – pp. 62-628.
3. Felten, E. W., and Schneider, M. A. Timing attacks on
4. web privacy. In Proceedings of the 7th ACM conference on Computer
5. and communications security, 2000. – pp. 25-32.
6. Jia, Y., Dong, X., Liang, Z., and Saxena, P. I know where you've been: Geo-inference attacks via the browser cache. IEEE Internet Computing 19, 1, 2015. – pp. 44-53.
7. Kotcher, R., Pei, Y., Jumde, P., and Jackson, C. Crossorigin pixel stealing: timing attacks using css filters. In Proceedings of the 2013 ACM SIGSAC conference on Computer &communications security, 2013. – 1055-1062 p.
8. Onarlioglu, K., Buyukkayhan, A. S., Robertson, W., and Kirda, E. Sentinel: Securing legacy firefox extensions. Computers & Security 49, 2015. – pp. 147-16.
9. Ter Louw, M., Lim, J. S., and Venkatakrishnan, V. Enhancing web browser security against malware extensions. Journal in Computer Virology 4, 3, 2008. – pp. 179-195.
10. Van Goethem, T., Joosen, W., and Nikiforakis, N. The clock is still ticking: Timing attacks in the modern web. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015. – pp. 1382-1393.
11. Giorgino, T. Computing and visualizing dynamic time warping alignments in r: The dtw package. JSS 31, 7, 2009. – pp. 1-24.
12. Kadloor, S., Kiyavash, N., and Venkitasubramaniam, P. Mitigating timing side channel in shared schedulers. IEEE/ACM Trans. Netw. 24, 3, 2016. – pp. 1562-1573.
13. 11.Lampson, B. W. A note on the confinement problem. Communications of the ACM 16, 10, 1973. – pp. 613-615.

Д О Д А Т К И

ДОДАТОК А

Графічна частина магістирської роботи

На рис. А.1 зображене фонове меню программи. В собі містить такі компоненти:

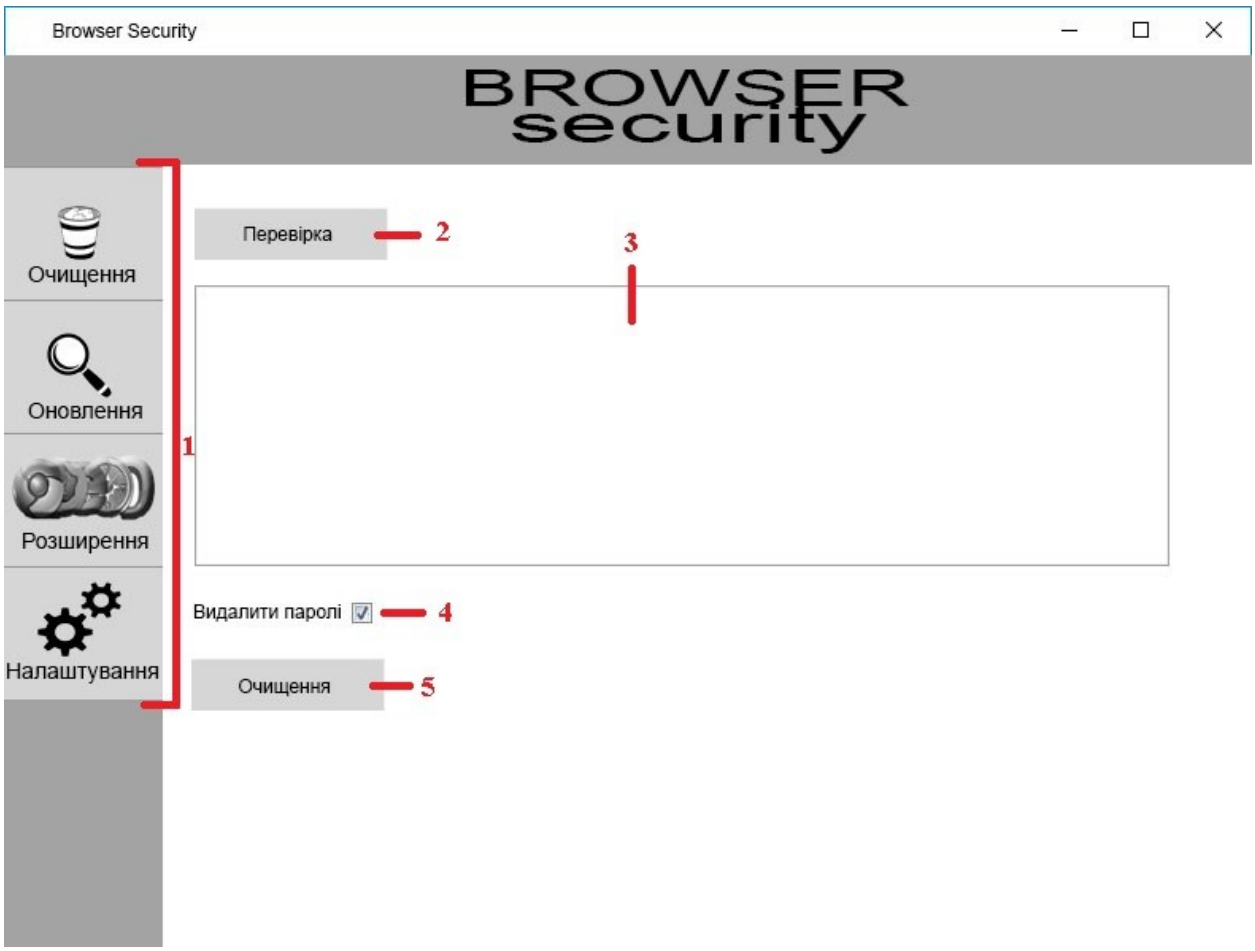

Рисунок А.1 – Фонове меню

1) Кнопки навігації;

Відповідають за вибір потрібної вкладки для роботи з програмою. У меню доступно 4 кнопки навігації: Очищення, Оновлення, Розширення і Налаштування.

2) Кнопка перевірки;

Відповідає за перевірку файлів браузерів такі як Історія, Cookie, Кеш.

3) Вікно для виведення інформації;

Виводиться інформація про знайдені файлах.

4) Чек бокс для функціївидалення паролей;

Використовується для повного видалення всіх паролів.

5) Кнопка очіценія;

Відповідає за виконання функції очищення всіх знайдених файлів і паролів.

Після натискання кнопки Перевірка на рис. А.2 програма починає розшук файлів браузерів та видає загальну кількість файлів для кожного розділу. Після повної перевірки кнопка Очищення стаєактивною для натискання.

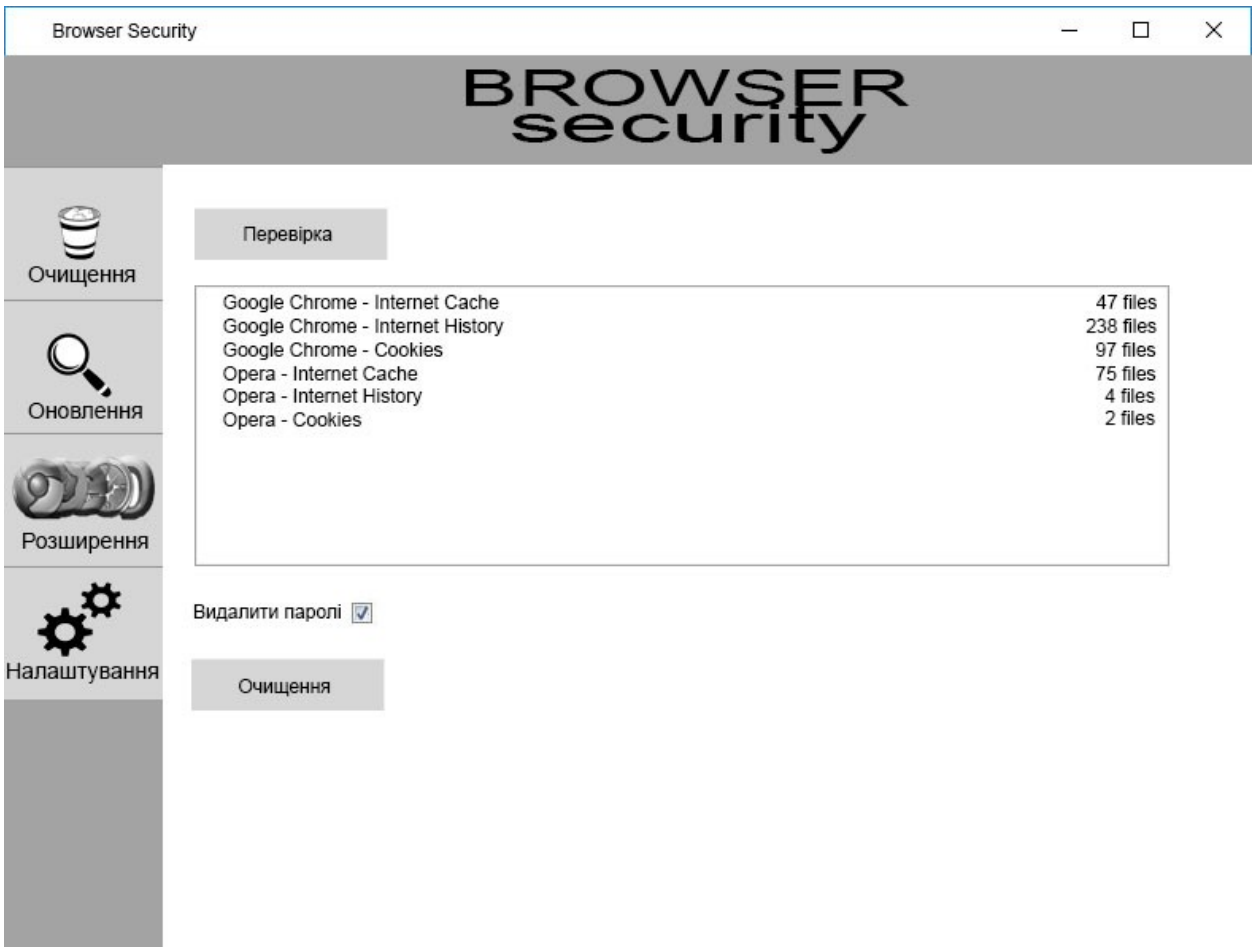

Рисунок А.2 – Заповненя вікна інфомації

На рис. А.3 зображена шкала виконання очищення.

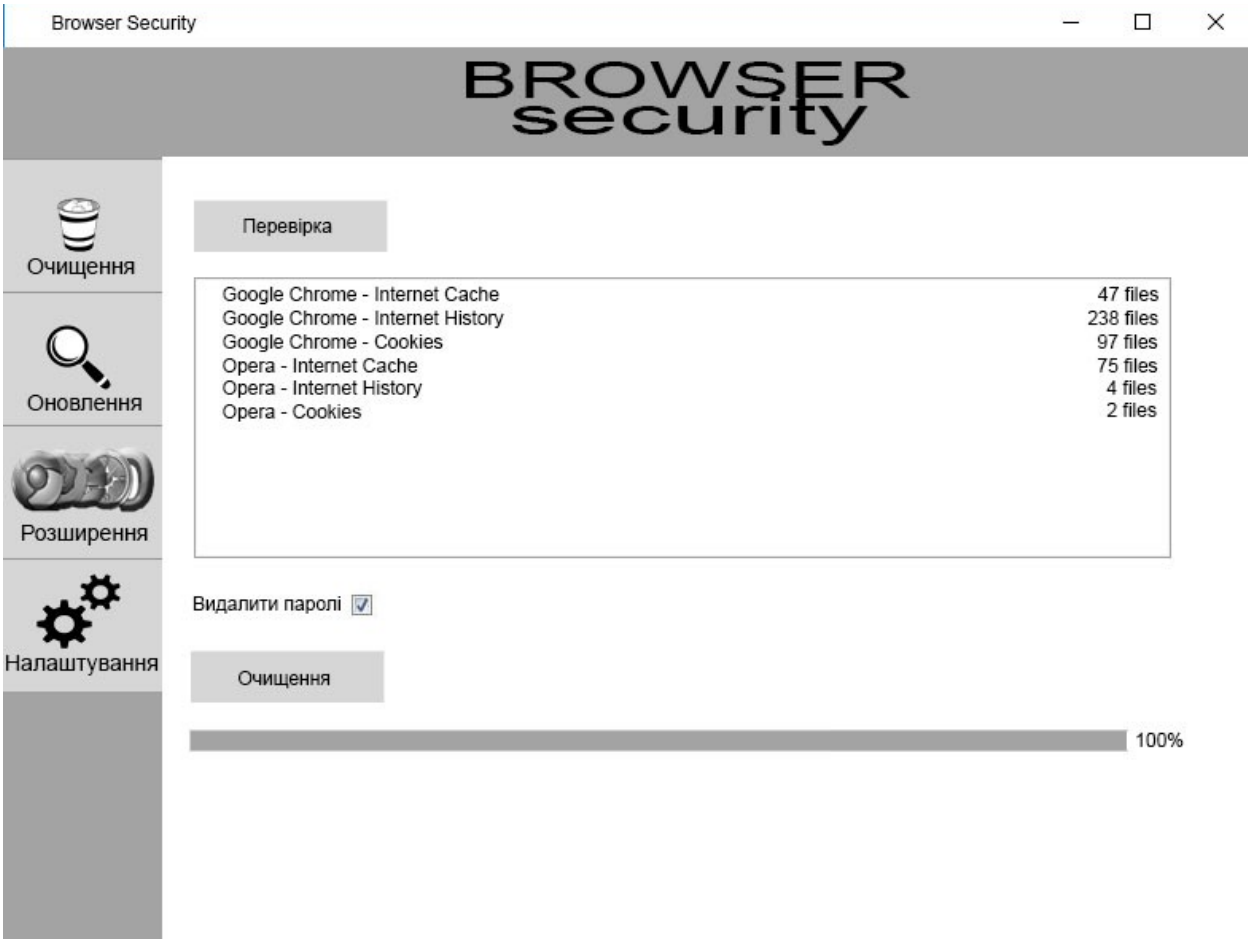

Рисунок А.3 – Очищення файлів

Меню Оновлення містить кнопку Перевірки на рис. А.5, вікно для виведення інформації, а також (1) 3 радіо-кнопки, що дозволяють вибрати необхідний браузер для перевірки оновлення.

При натисканні кнопки перевірки проходить перевірка версії, видає результат дійсної версії, та починає порівнення с останньою версією браузера.

Після натискання кнопки Так на рис. А.4 починається процес завантаження та установки оновлення.

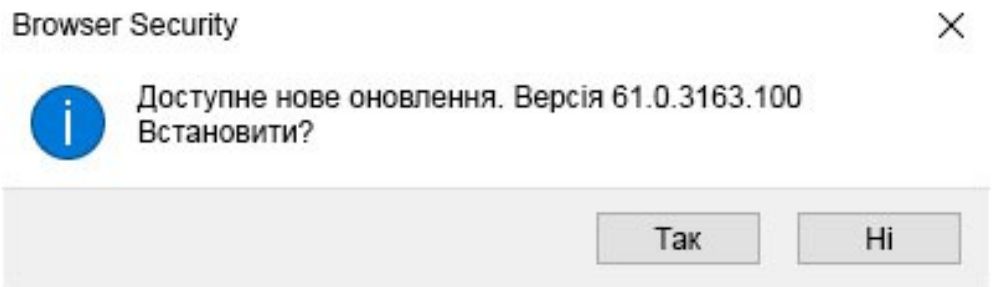

Рисунок А.4 – Спливаючі вікна

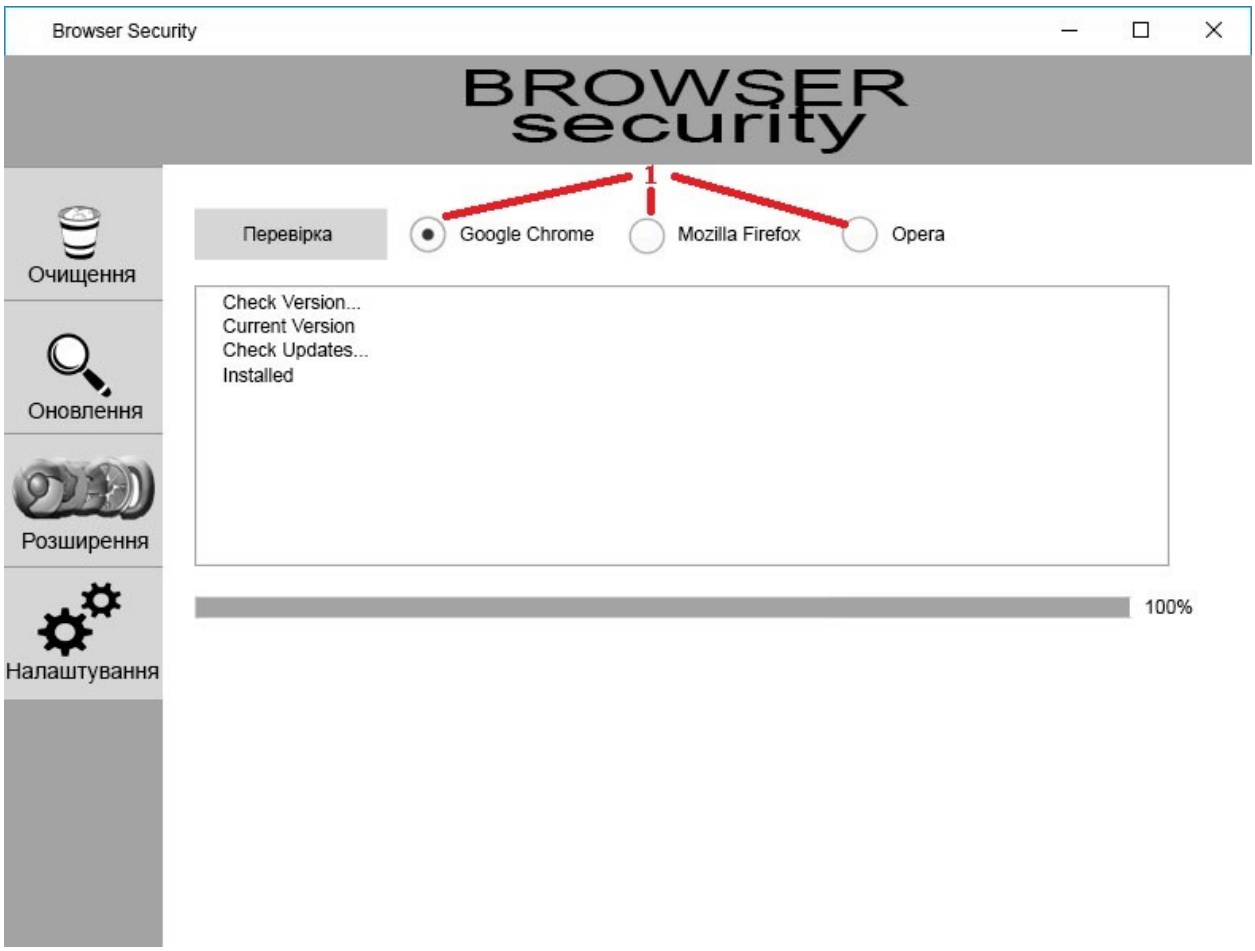

Рисунок А.5 – Меню Оновлення

Після натискання кнопки Так на рис. А.6 починається процес видалення розширення.

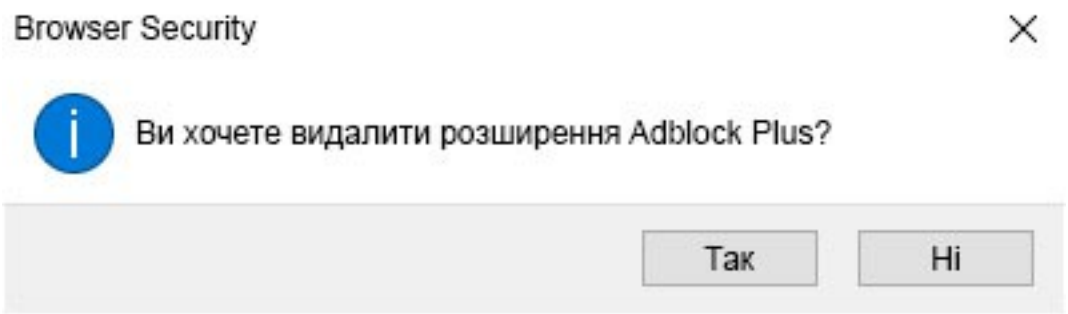

Рисунок А.6 – Спливаючі вікна

Дане меню предназначене для видалення розширень на рис. А.7, які вже встановленні.

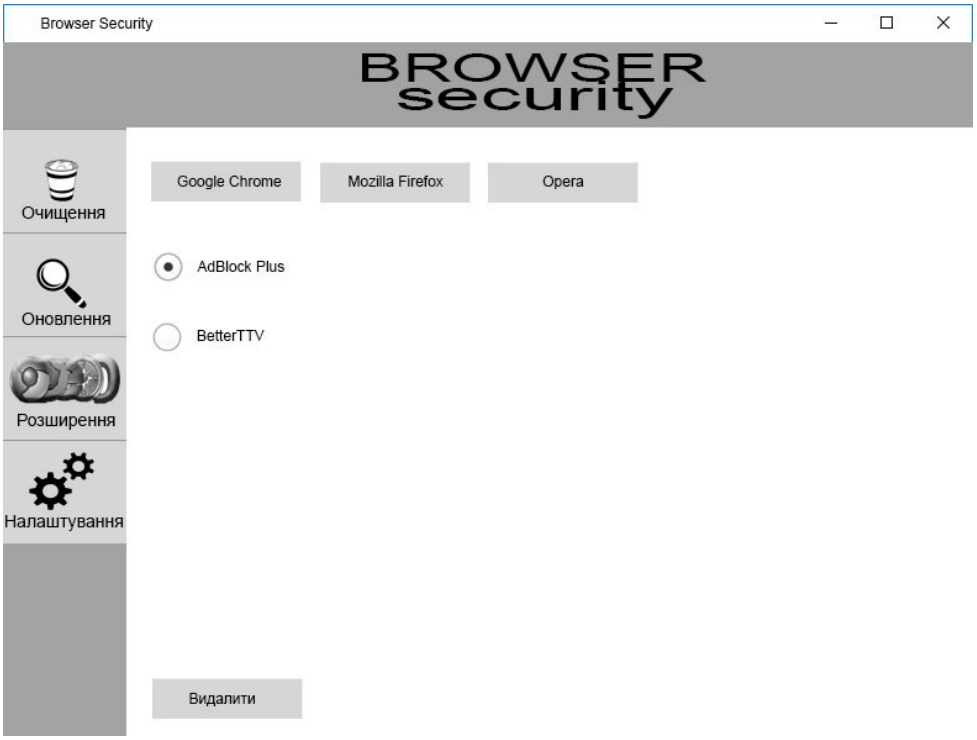

Рисунок А.7 – Меню Розширення

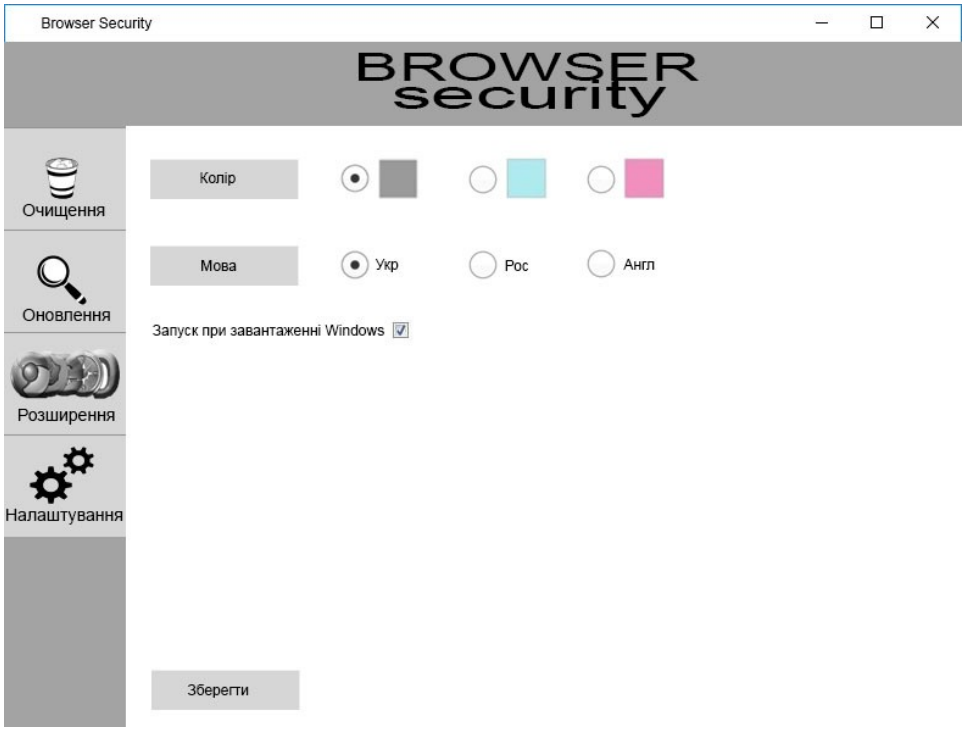

Рисунок А.8 – Меню Налаштування

У меню Налаштування на рис. А.8 можливо вибрати необхідний колір інтерфейсу, зручну мову, а також функцію запуску при завантаженні Windows.