Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження методів захисту Java додатків та їх програмна реалізація
(тема)

Виконав:
Випускник 2 курсу, групи ІПЗм-19-2
Біленький В.С.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Керівник проф. Качко О.Г.
(посада, прізвище)

Допускається до захисту
Зав. кафедри _____ З.В. Дудар
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова програма
(освітньо-професійна або освітньо-наукова)

Освітня програма Інженерія програмного забезпечення
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
« 26 » березня 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Біленького Владислава Сергійовича
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів захисту Java додатків та їх програмна реалізація»

затверджена наказом університету від 26.03.2021 № 385 Ст

2. Термін подання студентом роботи до екзаменаційної комісії «08» травня 2021 р.

3. Вихідні дані до роботи *методи захисту додатків на Java, середовище об'єктно-орієнтованого проєктування*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної галузі і постановка задачі, методи захисту додатків на Java, особливості захисту вебдодатків*

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, ілюстрацій (слайдів) *мета завдання, обґрунтування доцільності розроблення, постановка задачі, методи і алгоритми, структурно-логічна схема взаємодії даних, опис отриманих результатів, інтерфейс програмної системи, демонстраційні матеріали*

6 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	Качко О.Г.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної галузі	25.01.21 – 19.02.21	*Виконано*
2	Огляд існуючих методів та алгоритмів	20.02.21 – 10.03.21	*Виконано*
3	Алгоритми захисту ПЗ	11.03.21 – 25.03.21	*Виконано*
4	Підготовка пояснювальної записки	26.03.21 – 15.04.21	*Виконано*
5	Спецчастина	15.04.21 – 17.04.21	*Виконано*
6	Підготовка презентації та доповіді	18.04.21 – 19.04.21	*Виконано*
7	Попередній захист	20.04.21	*Виконано*
8	Нормоконтроль, рецензування	21.04.21 – 22.04.21	*Виконано*
9	Занесення диплома в електронний архів	23.04.21	*Виконано*
10	Допуск до захисту у зав. кафедри	07.05.21	*Виконано*

Дата видачі завдання 25 січня 2021 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Качко О.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Кваліфікаційна робота магістра містить: 127 с., 19 рис., 17 джерел.

ДЕОБФУСКАЦІЯ, ЗАХИСТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ОБФУСКАЦІЯ, API, AOT, JAVA, JIT, JVM, REVERSE ENGINEERING, RMI.

Об'єктом дослідження є методи захисту Java додатків.

Метою роботи є аналіз існуючих методів захисту Java додатків та їх програмна реалізація.

Методи розробки базуються на таких технологіях, як Java.

Результатом роботи є рекомендації по ефективному використанню методів захисту Java додатків та програмна реалізація цих методів, яка може бути вбудована в будь – які додатки для їх захисту

DEOBFUSCATION, SOFTWARE PROTECTION, OBFUSCATION, API, AOT, JAVA, JIT, JVM, REVERSE ENGINEERING, RMI.

The object of study is the approaches of Java software protection.

The purpose of the work is implementation of these approaches.

Development methods are based on technologies such as Java.

The result of the project is implementation and a set of patterns, best practices and guidelines of Java software protection effective application.

Я, *Біленький Владислав Сергійович,* студент групи ІПЗм-19-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів захисту Java додатків та їх програмна реалізація», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи

плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AOT – Ahead-of-Time;

API – Application Programming Interface;

DES – Data Encryption Standard;

J2EE – Java Enterprise Edition;

JIT – Just-In-Time;

JVM – Java Virtual Machine;

JVMPI – Java Virtual Machine Profile Interface;

RMI – Remote Method Invocation;

URL – Uniform Resource Locator;

ІБ – Інформаційна безпека;

ПЗ – Програмне забезпечення;

ПП – Програмний продукт.

ЗМІСТ

ВСТУП

В даний час актуальною є проблема захисту програмного забезпечення. Під захистом програмного забезпечення розуміється протидія зловмисних дій, спрямованих на те, щоб змусити програмне забезпечення працювати некоректно, розкрити конфіденційну інформацію, скоїти крадіжку інтелектуальної власності, відновити логіку і алгоритм роботи програми, здійснити незаконну модифікацію. Зловмисні дії супроводжуються процесом дослідження і аналізом коду програмного забезпечення, званим також реверсивної інженерії.

З моменту популяризації Java технології, виникла необхідність захисту програмного забезпечення, реалізованого з використанням даної технології. Варто відзначити, що технологія Java використовує керований код, який виконується у віртуальній машині. Для забезпечення цього, Java компілює вихідні коди в тимчасовий інтерпретований код – байткод (byte-code), який містить метаінформацію про структуру коду. У зв'язку з цим, реверсивна інженерія програмного забезпечення, яке розроблено з використанням технології Java, на порядок простіше, ніж для програмного забезпечення, вихідні коди якого компілюються в машинний код.

Дана робота присвячена аналізу таких методів захисту клієнтських Java додатків:

– заплутування коду (obfuscation);

– шифрування коду.

Також дана робота присвячена аналізу і методу захисту серверних Java додатків від наступних видів атак:

– міжсайтовий скриптинг (cross-site scripting);

– фіксація сесії (session fixation).

Результатом даного дослідження є розроблене ПЗ та рекомендації щодо ефективного використання методів захисту програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Способи захисту інтелектуальної власності

Швидкий розвиток інформаційних технологій в останнє десятиліття потребує в захисті такої інтелектуальної власності, як програмні продукти (ПП).

На даний момент розробка ефективного способу захисту певного програмного продукту стає одним із важливих завдань більшості програмістів, що беруть участь у розробці програмного забезпечення, оскільки це дозволяє їм продавати свою інтелектуальну роботу, і виключають можливість його незаконного використання. серед споживачів.

Витрати виробників на створення ефективного методу захисту своїх ПП окупляться та компенсують потенційну шкоду, заподіяну незаконним копіюванням та використанням програмних засобів.

Існують два основних способи захисту інтелектуальної власності, і, отже, самих програмних продуктів: юридичний (законний) і технічний.

1.1.1 Юридичний спосіб захисту інтелектуальної власності

Даний спосіб захисту полягає у створенні певних актів, відповідно до закону, які будуть охороняти інтелектуальну власність від нелегального використання. Цей спосіб включає в себе такі методи як патентування, оформлення авторських прав на інтелектуальну власність і т.д. Також він передбачає можливість реєстрації і отримання патенту на ПП, так, наприклад більшість ПП поставляються разом з ліцензією, яка підтверджує право користувача використовувати цей ПП, тобто, купуючи ліцензійну копію програми, користувач в деякій мірі виробляє покупку ліцензії на право роботи з її копією.

1.1.2 Технічний спосіб захисту інтелектуальної власності

Він реалізується шляхом включення до програмного забезпечення будь-якого з існуючих методів захисту, що забороняє його незаконне використання. Порівняно з юридичним методом захисту від надзвичайних ситуацій, він є найпоширенішим, оскільки є практичним та порівняно недорогим у впровадженні.

1.2 Особливості платформи Java

Вихід компілятора Java не є виконуваним кодом, що дозволяє їй вирішувати проблеми безпеки та мобільності. Це називається байткод [1].

Байткод – це високооптимізований набір інструкцій, призначений для виконання системою виконання Java, яка називається віртуальна машина Java (JVM).

Після того, як пакет виконання виконується в певній системі, будь-яка програма Java може працювати на ньому. Хоча деталі реалізації машини JVM можуть відрізнятися на інших платформах, всі вони можуть обробляти однакову обробку байткоду.

Архітектуру платформи Java зображено на рисунку 1.1.

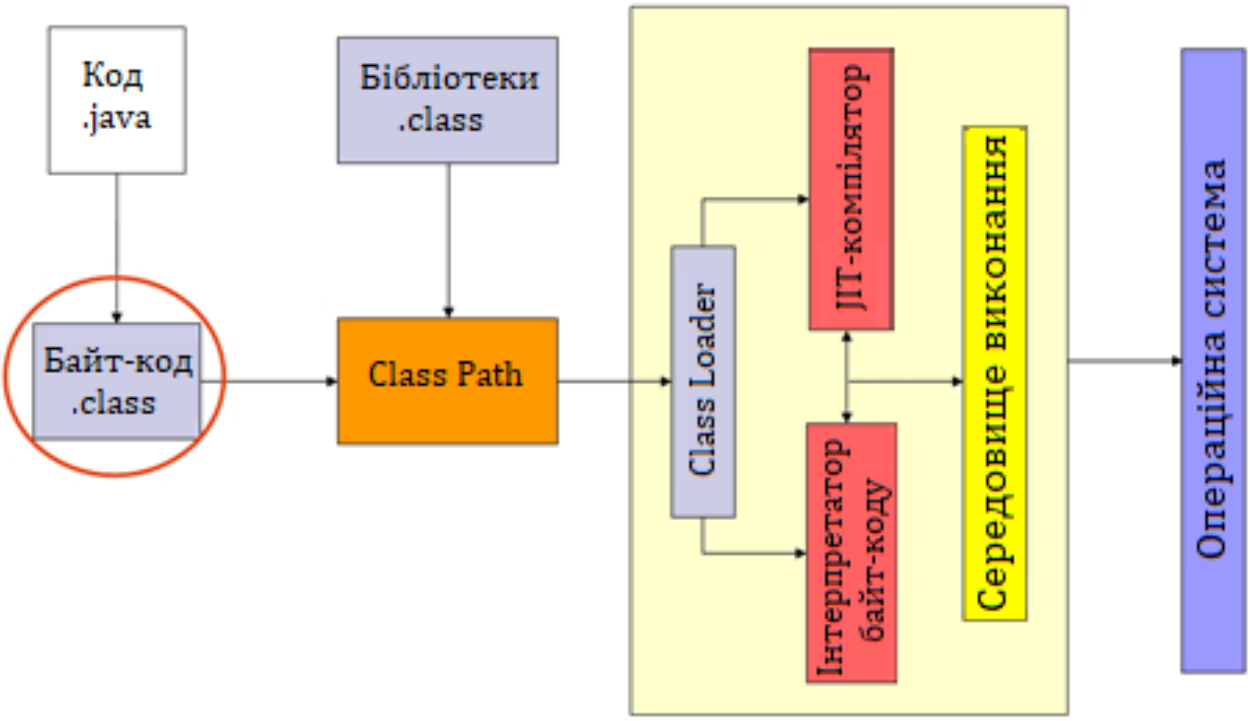

Рисунок 1.1 – Архітектура платформи Java

Той факт, що програма Java виконується JVM, також сприяє її безпеці. Оскільки JVM керує виконанням програми, він може ізолювати програму та запобігти її побічним ефектам поза системою.

Загалом, програма компілюється в тимчасову форму, а після чого йде інтерпретація JVM, через що робота йде повільніше, ніж якщо вона буде скомпільована у виконуваний код. Спочатку мова програмування Java був задуманий як інтерпретований, що дозволяє їй компілювати байткод у внутрішній код для підвищення продуктивності [2].

Тож незабаром після того, як була представлена Java, Sun почала поширювати свою технологію HotSpot. Ця технологія забезпечує компілятор байткодів Just-in-Time (JIT). Коли компілятор JIT є невід'ємною частиною машини JVM, відібрані частини байткодів по черзі компілюються в виконуваний код в процесі роботи з відповідними запитами. Компілювати відразу всю програму Java у виконуваний код недоцільно, оскільки Java виконує такі перевірки, які можна виконати лише під час виконання. Натомість під час виконання компілятор JIT компілює код за необхідності. Більш того, компілюються такі фрагменти байткоду, які отримують вигоду від компіляції програми. Решта коду просто інтерпретується.

Однак підхід JIT все ще забезпечує значний приріст продуктивності. Навіть коли програма динамічно компілюється в байткод, мобільність та характеристики безпеки зберігаються, оскільки JVM все ще відповідає за цілісність середовища виконання.

1.3 Постановка задачі

Як випливає з попереднього, Java програма перетвориться в байткод. Бібліотеки програм (jar-бібліотеки) також представляють собою байткод. Перетворення байткоду у вихідний текст може бути виконано значно простіше, ніж перетворення здійсненною програми в вихідний код, тому проблема захисту Java додатків є актуальною і є предметом дослідження в даній роботі.

Таким чином, для виконання завдання дослідження, потрібно виконати такі поставлені вимоги:

– дослідити методи захисту Java програм для настільних додатків;

– дослідити методи захисту Java програм для вебдодатків;

– визначити переваги і недоліки різних методів;

– виконати практичну реалізацію всіх розглянутих методів;

– дослідити якість програм до і після застосування окремих методів, а також їх сукупності.

Основною метою цієї роботи є дослідження методів захисту додатків з використанням технології Java, їх програмна реалізація, а також рекомендації щодо їх ефективного використання.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ЗАХИСТУ JAVA ДОДАТКІВ

2.1 Шифрування програмного коду

Шифрування програмного коду використовується, для того щоб запобігти втручанню в програму, а також ускладнити вивчення хакером того, як влаштована програма, як вона працює, як в ній реалізований метод захисту і т.д.

Цей спосіб захисту передбачає шифрування програмного коду, після чого він доставляється в зашифрованому вигляді кінцевим користувачам (іноді ефективно шифрувати лише найважливіші, найважливіші розділи коду, а не весь програмний код). Коли користувач запускає таку програму, спочатку буде запущена процедура розшифровки програми, якій потрібен ключ, за допомогою якого буде розшифрована програма [3].

Сам ключ зазвичай являє собою сукупність електронних даних (символів), який генерується в результаті певних (математичних) операцій.

Останнім часом стає актуально для розшифровки програми використовувати електронні ключі, вони є найбільш надійним і ефективним методом захисту дорогих ПП. Він надає високу стійкість до злому і не захищає від використання безкоштовної копії програми на різних комп’ютерах.

Електронний ключ являє собою невеликий пристрій, який приєднується до одного з портів комп'ютера, таких як COM (паралельний), LPT (послідовний), і також USB.

Шифрування програмного коду має такий недолік, коли зловмисник має можливість після придбання ліцензійної копії додатка витягувати з пам'яті розшифровані частини коду під час роботи програми.

2.2 Обфускація

У більшості випадків, щоб обійти захист, зловмиснику потрібно вивчити, як

працює його код і як він взаємодіє із самою захищеною програмою, цей процес навчання називається зворотним інженерним процесом. Цей процес часто залежить від властивостей психіки людини, отже, використання цих властивостей дозволяє знизити ефективність самого зворотного інженерного процесу.

Обфускація – це процес затуманення програмного коду, який призводить вихідний текст або виконуваний код в форму, який ускладнить процес зчитування та модифікації алгоритмів програми, як правило, такий процес усуває логічні зв'язки в ньому.

З цього випливає, що лише затуманення не має на меті забезпечити найбільш повний та ефективний захист програмних продуктів. Тому обфускація зазвичай використовується разом із одним із існуючих методів захисту (шифрування коду тощо), це може значно підвищити рівень захисту надзвичайного стану в цілому.

Використання обфускації в малому збільшує вартість ПП, що дотримується принципу економічної доцільності, так само, це дозволяє заощадити кошти, зменшити плагіат і захистити код від крадіжки [4].

Для програм, розроблених для платформи Java, обфускація можливо як на рівні вихідного коду, так і на рівні байткоду.

Основними методиками обфускації Java програм є:

– стирання налагоджувальної інформації. Class файли містять інформацію, яка необхідна для налагодження, така як: таблиця номерів рядків і таблиця локальних змінних. Стирання цієї інформації не відбивається на валідності байткода;

– обфускація контролю. Заважає потоку управління, тобто послідовності виконання програмного коду. Більшість його реалізації спирається на використання непрозорих предикатів, таких як послідовності операцій, результат яких важко визначити. Недоліки управлінням обфускації – зниження продуктивності коду, збільшення часу роботи алгоритму;

– заміна строкових констант на byte [] ускладнює пошук констант;

– лексична обфускація. Видалення коментарів, зміна форматування тексту, зміна імен змінних, методів і класів в набір безглуздих символів. Для людини, яка

аналізує такий код, семантика імен буде втрачена. Якщо використовувати обфускацію на рівні байткода, то можлива заміна імен змінних, методів, класів на ключові, зарезервовані слова (class, null, if, else). У разі декомпіляції, вихідний код неможливо скомпілювати, не вносячи змін. Проблема даного підходу полягає в некоректній роботі обфусцірованих додатків з Reflection API, наприклад, виклик методу за його імені [5].

Обфускація управління або розділення даних забезпечує досить серйозний захист від зворотної інженерії. Атакуючому необхідно відтворити граф виконання, для чого він повинен визначити, які з предикатів є «реальними», а які – «заглушками». Автоматизувати подібне завдання досить складно.

Завдання обфускації легко автоматизується, її можна звести до задачі створення лексичного / синтаксичного аналізатора і перетворення отриманих від них структур. Оскільки всі без винятку компілятори включають такі аналізатори, то на рівні компілятора обфускація реалізується просто.

Однак обфускація не є універсальним методом, вона лише збільшує час, необхідний на відновлення вихідного коду.

Також доведено, що неможливо побудувати перетворення, яке буде ефективно обфусцирувати будь-яку вхідну програму. Проте, на практиці для більшості програм можна побудувати перетворення, які будуть забезпечувати досить високий рівень захисту.

Існує велика кількість методів визначення ефективності застосування певного процесу обфускації до конкретного коду. Ці методи зазвичай поділяються на дві групи: аналітичні та емпіричні. Аналітичні методи базуються на трьох значеннях, що характеризують ефективність конкретного процесу обфускації:

– стійкість. Вказує на ступінь складності зворотного проєктування коду, який пройшов процес забруднення;

– еластичність. Вказує, наскільки даний процес затухання захистить програмний код від використання деобфускаторів;

– вартість перетворення. Дозволяє оцінити, наскільки більше системних ресурсів потрібно для виконання коду минулого процесу затуманення, ніж для

виконання оригінального програмного коду.

Такі методи ефективно використовуються якщо порівнювати з іншими алгоритмами обфускації, але є питання, на який складно дати відповідь, яка ефективність того чи іншого методу до певного коду.

Емпіричні ж методи ґрунтуються на статистичному аналізі і одержуваних з нього результатів. Для проведення такого дослідження потрібно зібрати фахівців в області реверсивної інженерії, фрагмент коду, який захищений програмою, і підбір різних алгоритмів обфускації.

Результати даного дослідження будуть включати в себе мінімальну кількість часу, який буде потрібний групі людей, для того щоб вивчити кожен фрагмент коду минулого один з алгоритмів обфускації.

2.3 Деобфускація

З одного боку, процес оптимізації програмного коду можна віднести до процесу деобфускації, оскільки вони обидва, в тій чи іншій мірі, протилежні процесу обфускації. У процесі затуманення до програмного коду часто додаються непотрібні операції, які, як правило, жодним чином не впливають на результати самої програми і покликані ускладнити процес вивчення програмного коду [6].

Основні класифікації способів деобфускації:

– знаходження і оцінювання непрозорих конструкцій (так званих предикати), статичний аналіз яких важко виконаємо;

– відповідність шаблону. Він здійснюється різними способами, найпоширенішими є два з них. Перший полягає в тому, що беруться кілька однакових програм, які пройшли процес обфускації (оскільки процес обфускації в більшості випадків унікальний, їх код також буде іншим, хоча вони будуть виконувати однакові дії), а порівняння їх фрагментів коду, для виявлення обфускації надлишкового коду, вставленого в процесі реалізації, який у

майбутньому просто видаляється. Другий метод узгодження шаблонів, здійснюється пошук в програмному коді поширених конструкцій, які потім використовуються в процесі затухання. Дані конструкції потім можно використати, наприклад для того, щоб зберегти або оновити дані у базі даних, який допоже в вивченні роботи самого обфускатора;

– виділення фрагментів коду в програмі, які жодним чином не пов'язані з основними завданнями, які програма повинна виконувати, тобто виявлення непотрібних розділів коду

– статистичний аналіз – це динамічний аналіз програмного коду. Наприклад, пошук непрозорих предикатів може здійснюватися шляхом виділення та подальшого вивчення в аналізованому програмному коді тих предикатів, які завжди повертають одне і те ж значення під час його виконання. Статистичний аналіз також може бути виконаний для оцінки правильності процесу деобфускації, оскільки ця програма "А" запускається паралельно, а програма, отримана в результаті деобфускації "А", передає їм еквівалентні вхідні дані, а вихідні дані порівнюються. Якщо вихідні дані схожі, то можна зробити висновки, що процес деобфускації було проведено правильно;

– аналіз потоку даних, заснований на вивченні того, як дані (змінні, масиви) змінюються в програмі під час роботи програми.

Статичний аналіз – це сімейство технологій аналізу програм, де програму, яка аналізує, насправді не потрібно запускати, тоді як необхідна інформація про неї отримується за допомогою спеціальних програм. Наприклад, статичний аналіз програм, представлених у двійковій формі, може бути здійснений за допомогою декомпілятора та представлений в оригінальній формі за допомогою будь-якого текстового редактора. Технології статичного аналізу відрізняються від більшості існуючих, головна якість полягає в тому, що він є більш складним і базується на семантиці (визначає семантичне значення речень алгоритмічної мови) самого програмного коду.

За допомогою статичного аналізу можна вивчити причини можливої поведінки під час виконання, тобто результати такого аналізу можна назвати абсолютно точними.

Динамічний аналіз полягає в аналізі та тестуванні програми під час її виконання. Він вважається точним, оскільки вивчає фактичну поведінку програми під час її роботи.

Динамічний аналіз зазвичай швидший, ніж статичний, оскільки час його виконання часто залежить від швидкості виконання розглянутої програми. Статичний аналіз, навпаки, зазвичай вимагає великих обчислень і займає багато часу, особливо при аналізі великих програм. Недоліком динамічного аналізу є те, що отримані результати можуть не відповідати результатам, отриманим під час наступних запусків тієї самої програми.

Основні проблеми деобфускації, пов'язані з необхідною кількістю обчислень, і складністю її алгоритмів.

2.4 Винесення критичного програмного коду в окремий захищений модуль

Цей тип методів заснований на тому, коли критична ділянка коду переміщається в захищений модуль (наприклад, конфіденційну інформацію, яка є інтелектуальною власністю), за рахунок чого ступінь захисту буде вище, ніж вихідний код програми.

Однією з популярних реалізацій даного підходу є винесення критичних ділянок коду на захищений сервер.

На сервер додатка перекладається робота по виконанню критичної ділянки (секретного алгоритму). Клієнтська частина (до якої має доступ зловмисник) може спілкуватися з сервером за допомогою безлічі технологій і протоколів, таких як RMI, EJB, RPC, XML-RPC, SOAP, CORBA і т.д.

Основна мета RMI – дати можливість програмістам розробляти розподілені

програми Java, використовуючи той самий синтаксис та семантику, що і при розробці звичайних нерозподілених програм. Для цього необхідно взяти одну модель роботи класів та об'єктів, які знаходяться в одній віртуальній машині JVM і перетворити в іншу модель в розподілену обчислювальне середовище (множинний JVM).

2.4.1 Архітектура Java RMI

Метою архітектури RMI є втілення такої розподіленої компонентної об'єктної моделі, яка б входила в мову програмування Java, та так само в локальну об'єктну модель.

Архітектура RMI дотримується такого принципу, в якому поведінка і реалізація є різними позначеннями. RMI дозволяє розділити і виконати на різних JVM код, який визначає та реалізує цю поведінку [7].

Це підходить для розподілених систем, де клієнти знають визначення служб, а сервери надають ці послуги.

Саме в RMI така служба кодується в класі завдяки інтерфейсу Java (див. рис. 2.1).

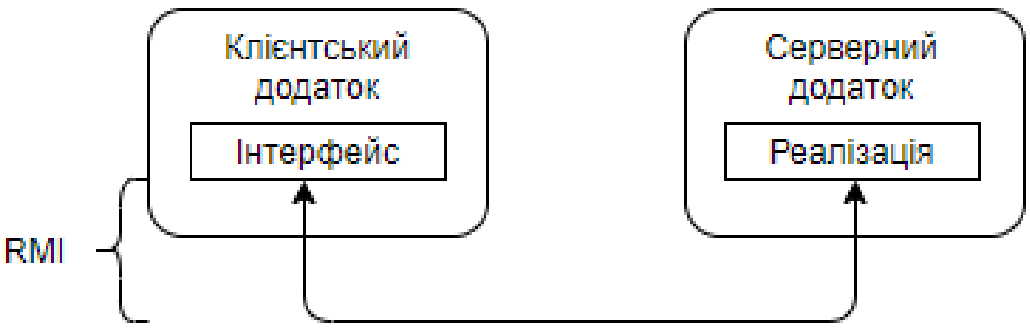

Рисунок 2.1 – Архітектура RMI

Реалізація RMI містить три різних абстрактних рівнів.. Перший – це заглушка та каркасний рівень, розташовані безпосередньо перед забудовником. Цей рівень перехоплює виклики методу, зроблені клієнтом, використовуючи посилальну змінну інтерфейсу, і пересилає їх до віддаленої служби RMI.

Наступним рівнем є рівень віддаленого посилання. Цей рівень розуміє, як інтерпретування і управління посиланнями на віддаленому рівні сервісними об'єктами. Це з'єднання є одностороннім посиланням (односпрямоване посилання).

Транспортний рівень забезпечує такі вбудовані сполуки як TCP та IP, а так само вбудовані стратегії захисту від несанкціонованого доступу (див. рис. 2.2).

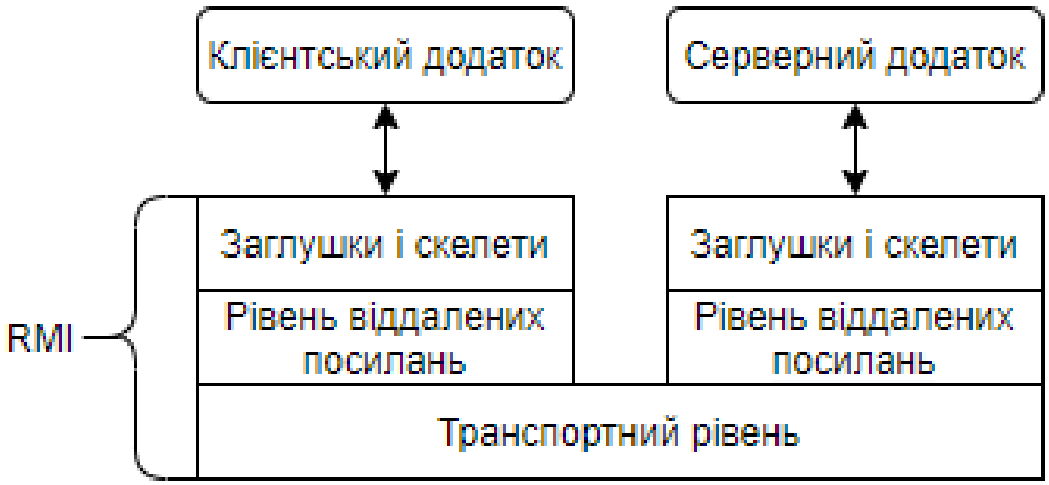

Рисунок 2.2 – Архітектура RMI

Даний підхід повністю вирішує проблему зворотної інженерії, розкриття секретного алгоритму. Недоліком є низька продуктивність, пов'язана з пропускною спроможністю каналів зв'язку, залежність клієнта від каналу, стану сервера.

2.5 Використання Ahead-Of-Time компіляції

Ahead-of-Time (AOT) – це різновид компіляції, в результаті якого

отримується виконуваний файл готовий до виконання в будь-який час.

AOT компілюють Java код в машинний, що в свою чергу серйозно ускладнює завдання декомпіляцію.

Недоліками цього підходу можна виділити:

– залежність від платформи, не для всіх платформ існують AOT компілятори;

– збільшення розміру виконуваної програми (в порівнянні з байткод);

– неможливість AOT компіляції для динамічних додатків, які завантажують класи в run-time (плагіни і т.п.), маніпулюють байткод в run-time для створення proxy класів;

– відсутність специфічної для платформи оптимізації, які застосовуються в JIT компіляторах, наприклад використання Intel MMX / SSE / SSE2 розширень, тому що необхідно виробляти код для широкого кола платформ.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ ЗАХИСТУ JAVA ДОДАТКІВ

3.1 Декомпіляція і шифрування байткоду

Розглянемо клас edu.nure.javaprotect.bytecode.algorithm.SecretAlgorithmImpl.java:

```
public class SecretAlgorithmImpl {
    private static final int CONST = 1;

    public int perform(int i) {
        return i + CONST;
    }
     }
```

Скомпілюємо клас в байткод:

javac SecretAlgorithmImpl.java

Скористаємося утилітою javap зі стандартної поставки JDK, яка дозволяє аналізувати код байткод:

javap -verbose -c -p -s SecretAlgorithmImpl > javap.txt

Результат виконання утиліти javap наведено нижче.

```
Compiled from "SecretAlgorithmImpl.java"
public class edu.nure.javaprotect.bytecode.SecretAlgorithmImpl extends
java.lang.Object
  SourceFile: "SecretAlgorithmImpl.java"
  minor version: 0
  major version: 49
  Constant pool:
const #1 = Method     #3.#16;    //  java/lang/Object."<init>":()V
const #2 = class#17; //  edu/nure/javaprotect/bytecode/SecretAlgorithmImpl
const #3 = class#18; //  java/lang/Object
const #4 = AscizCONST;
const #5 = AscizI;
const #6 = AscizConstantValue;
const #7 = int  1;
const #8 = Asciz<init>;
const #9 = Asciz()V;
const #10 = Asciz     Code;
const #11 = Asciz     LineNumberTable;
const #12 = Asciz     perform;
const #13 = Asciz     (I)I;
const #14 = Asciz     SourceFile;
const #15 = Asciz     SecretAlgorithmImpl.java;
const #16 = NameAndType     #8:#9;//  "<init>":()V
const #17 = Asciz     edu/nure/javaprotect/bytecode/SecretAlgorithmImpl;
const #18 = Asciz     java/lang/Object;
```

```
{
private static final int CONST;
  Signature: I
  Constant value: int 1

public edu.nure.javaprotect.bytecode.SecretAlgorithmImpl();
  Signature: ()V
  Code:
   Stack=1, Locals=1, Args_size=1
   0: aload_0
   1: invokespecial    #1; //Method java/lang/Object."<init>":()V
   4: return
  LineNumberTable:
   line 6: 0

public int perform(int);
  Signature: (I)I
  Code:
   Stack=2, Locals=2, Args_size=2
   0: iload_1
   1: iconst_1
   2: iadd
   3: ireturn
  LineNumberTable:
   line 10: 0

      }
```

Ми отримали набір інструкція байткоду. Байткод містить інформацію про:

– пул констант (імена, значення);

– імена, модифікатори доступів, успадкування класів та інтерфейсів;

– опис полів, методів, атрибутів;

– таблиці локальних змінних.

Як, видно, з прикладу, даної інформації досить щоб відновити вихідний код. Існує велика кількість реалізацій декомпілятори, які дозволяють отримати вихідний код. Найбільш популярною реалізацією є JAD [8].

Коли JVM завантажує class файл, вона отримує потік байткода для кожного методу. Байткод може бути інтерпретований віртуальною машиною, скомпільовано в машинний код на льоту (just-in-time) або виконаний іншим способом (залежить від реалізації JVM).

При завантаженні класу JVM використовуються так звані classloader. Classloader в свою чергу мають ієрархічну структуру. Дочірній classloader має доступ до всіх класів, які завантажив його предок, а при завантаженні нового класу

він делегує завантаження свого батька. У разі якщо батько не може завантажити клас, то дочірній classloader намагається виконати цю операцію самостійно.

– bootstrap classloader (вершина ієрархії, завантажує бібліотеки core java (rt.jar));

– extension classloader (наслідує bootstrap, завантажує бібліотеки з lib / ext);

– system classloader (успадковує extension, завантажує класи з classpath);

Для запобігання несанкціонованому використанню програм деякі джерела, наприклад, рекомендують таку техніку:

– зашифрувати байткод;

– створити реалізацію класу java.lang.ClassLoader, який зможе розшифрувати байт код і тим самим завантажити class в пам'ять VM. Додати ключ для розшифровки як вхідний параметр.

Розглянемо наступну реалізацію утиліти шифрування байткоду, яка використовує DES шифрування:

```
public class Encrypter {

    private static final String ALGORITHM = "DES";
    public static final String ENCRYPTED_FILE_EXT = "enc";

    public static SecretKey generateKey() throws Exception {
        return KeyGenerator.getInstance(ALGORITHM).generateKey();
    }

    public static byte[] encrypt(byte[] inputBytes, SecretKey key) throws
InvalidKeyException,    BadPaddingException,    IllegalBlockSizeException,
NoSuchAlgorithmException, NoSuchPaddingException {
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.ENCRYPT_MODE, key);
        return cipher.doFinal(inputBytes);
    }

    public static void encryptFile(File file, SecretKey key) throws
InvalidKeyException,    BadPaddingException,    IllegalBlockSizeException,
NoSuchAlgorithmException, NoSuchPaddingException, IOException {
```

```
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] encryptedBytes = cipher.doFinal(Utils.fileToBytes(file));
        Utils.bytesToFile(encryptedBytes, file + "." + ENCRYPTED_FILE_EXT);
    }

    public static void encryptFilesInPath(File path, SecretKey key) throws
IllegalBlockSizeException,        IOException,        InvalidKeyException,
NoSuchAlgorithmException, NoSuchPaddingException, BadPaddingException {
        File[] files = path.listFiles();
        for (File file : files) {
            if (file.isDirectory()){
                encryptFilesInPath(file, key);
            } else if (file.getName().endsWith(".class")) {      // TODO
                encryptFile(file, key);
                file.delete();
            }
        }
    }

    public static byte[] decrypt(byte[] encryptionBytes, SecretKey key)
throws InvalidKeyException, BadPaddingException, IllegalBlockSizeException,
NoSuchAlgorithmException, NoSuchPaddingException {
        Cipher cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(Cipher.DECRYPT_MODE, key);
        return cipher.doFinal(encryptionBytes);
    }

    public static void writeKey(SecretKey key, File file) throws IOException,
NoSuchAlgorithmException, InvalidKeySpecException {
        SecretKeyFactory                    keyfactory                    =
SecretKeyFactory.getInstance(ALGORITHM);
        DESKeySpec   keyspec   =   (DESKeySpec)   keyfactory.getKeySpec(key,
DESKeySpec.class);
        byte[] rawkey = keyspec.getKey();
        FileOutputStream out = new FileOutputStream(file);
        out.write(rawkey);
        out.close();
```

```
    }

    public static SecretKey readKey(File file) throws IOException,
NoSuchAlgorithmException, InvalidKeyException, InvalidKeySpecException {
        DataInputStream in = new DataInputStream(new FileInputStream(file));
        byte[] rawkey = new byte[(int) file.length()];
        in.readFully(rawkey);
        in.close();

        DESKeySpec keyspec = new DESKeySpec(rawkey);
        SecretKeyFactory keyfactory =
SecretKeyFactory.getInstance(ALGORITHM);
        return keyfactory.generateSecret(keyspec);
    }
     }
```

`public static SecretKey generateKey()` – дозволяє згенерувати ключ для DES.

`public static void writeKey(SecretKey key, File file)` – записує DES ключ в файл.

`public static void writeKey(SecretKey key, File file)` – зашифровує всі файли рекурсивно по заданому шляху .

Скористаємося цими методами для шифрування класів проєкту по шляху зазначеному в змінній path:

```
        SecretKey key = Encrypter.generateKey();
        File keyFile = new File(KEY_FILE_NAME);
        Encrypter.writeKey(key, keyFile);

        String path =
"D:\\Projects\\java\\diploma\\out\\production\\diploma\\edu\\nure\\javaprot
ect\\bytecode\\algorithm\\";

            Encrypter.encryptFilesInPath(new File(path), key);
```

В результаті отримуємо зашифрований клас

D:\Projects\java\diploma\out\production\diploma\edu\nure\javaprotect\bytecode\algorithm\SecretAlgorithmImpl.class.enc

Перейменуємо цей файл в SecretAlgorithmImpl.class

При спробі скористатися утилітою

javap -verbose -c -p -s SecretAlgorithmImpl > javap.txt, отримаємо наступну помилку:

```
D:\...ion\diploma\edu\nure\javaprotect\bytecode\algorithm>javap  -verbose -
c -p -s SecretAlgorithmImpl > javap.txt
java.lang.ClassFormatError: wrong magic: 0x5C0721F6, expected 0xCAFEBABE
        at sun.tools.javap.ClassData.read(ClassData.java:71)
        at sun.tools.javap.ClassData.<init>(ClassData.java:52)
        at sun.tools.javap.JavapPrinter.<init>(JavapPrinter.java:28)
        at sun.tools.javap.Main.displayResults(Main.java:201)
        at sun.tools.javap.Main.perform(Main.java:61)
        at sun.tools.javap.Main.entry(Main.java:49)
        at sun.tools.javap.Main.main(Main.java:34)
ERROR:fatal error
Exception in thread "main" java.lang.NullPointerException
        at sun.tools.javap.ClassData.getName(ClassData.java:570)
        at sun.tools.javap.ClassData.getSourceName(ClassData.java:597)
        at
sun.tools.javap.JavapPrinter.printclassHeader(JavapPrinter.java:47)
        at sun.tools.javap.JavapPrinter.print(JavapPrinter.java:36)
        at sun.tools.javap.Main.displayResults(Main.java:202)
        at sun.tools.javap.Main.perform(Main.java:61)
        at sun.tools.javap.Main.entry(Main.java:49)
        at sun.tools.javap.Main.main(Main.java:34)
```

Таким чином, зашифровані класи можна завантажити за допомогою стандартного інтерпретатора байткоду, можна декомпілювати і неможливо виконати без ключа для розшифровки.

Тепер розглянемо реалізацію класу java.lang.ClassLoader, який зможе розшифрувати байткод і тим самим завантажити клас в пам'ять VM. Ключ розшифровки, як вхідний параметр, буде використовуватися раніше згенерований файл key:

```
public class DecryptClassLoader extends ClassLoader {
    private SecretKey secretKey;

    public DecryptClassLoader(ClassLoader parent, String secretKeyFileName)
{
        super(parent);
        try {
            this.secretKey              =              Encrypter.readKey(new
File(secretKeyFileName));
        } catch (Exception e) {
```

```
            throw new DecryptClassLoaderException("Can't read key from file
" + secretKeyFileName, e);
        }
    }

    @Override
    protected Class<?> findClass(String className) throws
ClassNotFoundException {
        URL encryptedURL = getResource(className.replaceAll("\\.", "/") +
".class." + Encrypter.ENCRYPTED_FILE_EXT); // find resource in classpath
        byte[] encryptedBytes;
        File file = new File(encryptedURL.getFile());
        try {
            encryptedBytes = Utils.fileToBytes(file);
        } catch (IOException e) {
            throw new DecryptClassLoaderException("Can't convert given file
" + file + " to byte[]", e);
        }
        byte[] classBytes = decrypt(encryptedBytes);
        return defineClass(className, classBytes, 0, classBytes.length);
    }

    private byte[] decrypt(byte[] encryptedBytes) {
        try {
            return Encrypter.decrypt(encryptedBytes, secretKey);
        } catch (Exception e) {
            throw new DecryptClassLoaderException("Error occurred during
byte code decryption", e);
        }
    }
     }
```

Завантажимо клас edu.nure.javaprotect.bytecode.algorithm.SecretAlgorithmImpl використовуючи класс DecryptClassLoader:

```
        DecryptClassLoader loader = new
DecryptClassLoader(ClassLoader.getSystemClassLoader(), KEY_FILE_NAME);

        Class<?> aClass =
loader.loadClass("edu.nure.javaprotect.bytecode.algorithm.SecretAlgorithmIm
```

```
pl");
```

Виконаємо метод perform().

```
        Class<?>                              aClass                              =
loader.loadClass("edu.nure.javaprotect.bytecode.algorithm.SecretAlgorithmIm
pl");
        Method performMethod = aClass.getMethod("perform", int.class);

        Object instance = aClass.newInstance();

        System.out.println(performMethod.invoke(instance, 5));
```

DecryptClassLoader успішно розшифрував байткод.

3.2 Уразливості шифрування байткоду

Уразливість в підході з шифруванням байткоду полягає в тому, що все classloader доставляють масиви байт класів в метод `java.lang.ClassLoader.defineClass()`.

У свою чергу цей метод викликає native код JVM використовуючи JNI. Таким чином, щоб перехопити масив байт вмісту класу, досить поставити крапку зупинки відладчика в цьому методі або підкласти свою реалізацію класу `java.lang.ClassLoader` в rt.jar, яка б залогувала вміст масиву (див. рис. 3.1).

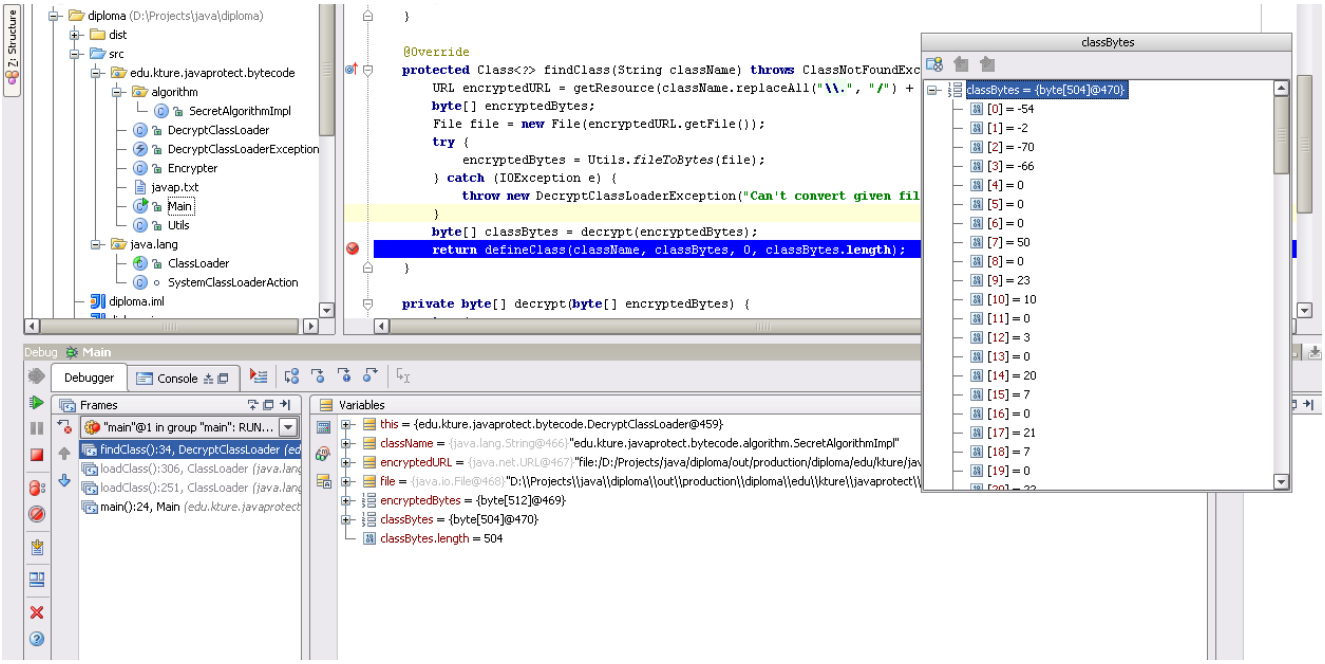

Рисунок 3.1 – Точка зупинки DecryptClassLoader. findClass()

Клас java.lang.ClassLoader міститься в архіві JAVA_HOME / jre / lib / rt.jar. Тексти програм знаходяться в JAVA_HOME / src.zip.

Знайдемо вихідний код класу і виправимо метод:

```
protected final Class<?> defineClass(String name, byte[] b, int off, int
len)
        throws ClassFormatError
```

Таким чином:

```
protected final Class<?> defineClass(String name, byte[] b, int off, int
len)
        throws ClassFormatError
    {
        System.out.println(name);
        System.out.println(Arrays.toString(b));
        return defineClass(name, b, off, len, null);
        }
```

Скомпілюємо нашу версію класу ClassLoader і підкладемо її замість оригінальної.

Тепер при запуску нашої програми отримуємо наступний висновок:

```
edu.nure.javaprotect.bytecode.algorithm.SecretAlgorithmImpl
[-54, -2, -70, -66, 0, 0, 0, 49, 0, 23, 10, 0, 3, 0, 20, 7, 0, 21, 7, 0,
22, 1, 0, 5, 67, 79, 78, 83, 84, 1, 0, 1, 73, 1, 0, 13, 67, 111, 110, 115,
```

```
116, 97, 110, 116, 86, 97, 108, 117, 101, 3, 0, 0, 0, 1, 1, 0, 6, 60, 105,
110, 105, 116, 62, 1, 0, 3, 40, 41, 86, 1, 0, 4, 67, 111, 100, 101, 1, 0,
15, 76, 105, 110, 101, 78, 117, 109, 98, 101, 114, 84, 97, 98, 108, 101, 1,
0, 18, 76, 111, 99, 97, 108, 86, 97, 114, 105, 97, 98, 108, 101, 84, 97,
98, 108, 101, 1, 0, 4, 116, 104, 105, 115, 1, 0, 62, 76, 101, 100, 117, 47,
107, 116, 117, 114, 101, 47, 106, 97, 118, 97, 112, 114, 111, 116, 101, 99,
116, 47, 98, 121, 116, 101, 99, 111, 100, 101, 47, 97, 108, 103, 111, 114,
105, 116, 104, 109, 47, 83, 101, 99, 114, 101, 116, 65, 108, 103, 111, 114,
105, 116, 104, 109, 73, 109, 112, 108, 59, 1, 0, 7, 112, 101, 114, 102,
111, 114, 109, 1, 0, 4, 40, 73, 41, 73, 1, 0, 1, 105, 1, 0, 10, 83, 111,
117, 114, 99, 101, 70, 105, 108, 101, 1, 0, 24, 83, 101, 99, 114, 101, 116,
65, 108, 103, 111, 114, 105, 116, 104, 109, 73, 109, 112, 108, 46, 106, 97,
118, 97, 12, 0, 8, 0, 9, 1, 0, 60, 101, 100, 117, 47, 107, 116, 117, 114,
101, 47, 106, 97, 118, 97, 112, 114, 111, 116, 101, 99, 116, 47, 98, 121,
116, 101, 99, 111, 100, 101, 47, 97, 108, 103, 111, 114, 105, 116, 104,
109, 47, 83, 101, 99, 114, 101, 116, 65, 108, 103, 111, 114, 105, 116, 104,
109, 73, 109, 112, 108, 1, 0, 16, 106, 97, 118, 97, 47, 108, 97, 110, 103,
47, 79, 98, 106, 101, 99, 116, 0, 33, 0, 2, 0, 3, 0, 0, 0, 1, 0, 26, 0, 4,
0, 5, 0, 1, 0, 6, 0, 0, 0, 2, 0, 7, 0, 2, 0, 1, 0, 8, 0, 9, 0, 1, 0, 10, 0,
0, 0, 47, 0, 1, 0, 1, 0, 0, 0, 5, 42, -73, 0, 1, -79, 0, 0, 0, 2, 0, 11, 0,
0, 0, 6, 0, 1, 0, 0, 0, 6, 0, 12, 0, 0, 0, 12, 0, 1, 0, 0, 0, 5, 0, 13, 0,
14, 0, 0, 0, 1, 0, 15, 0, 16, 0, 1, 0, 10, 0, 0, 0, 56, 0, 2, 0, 2, 0, 0,
0, 4, 27, 4, 96, -84, 0, 0, 0, 2, 0, 11, 0, 0, 0, 6, 0, 1, 0, 0, 0, 10, 0,
12, 0, 0, 0, 22, 0, 2, 0, 0, 0, 4, 0, 13, 0, 14, 0, 0, 0, 0, 0, 4, 0, 17,
0, 5, 0, 1, 0, 1, 0, 18, 0, 0, 0, 2, 0, 19]
```

Другий спосіб злому полягає в використанні Java Virtual Machine Profile Interface (JVMPI) (див. рис. 3.2).

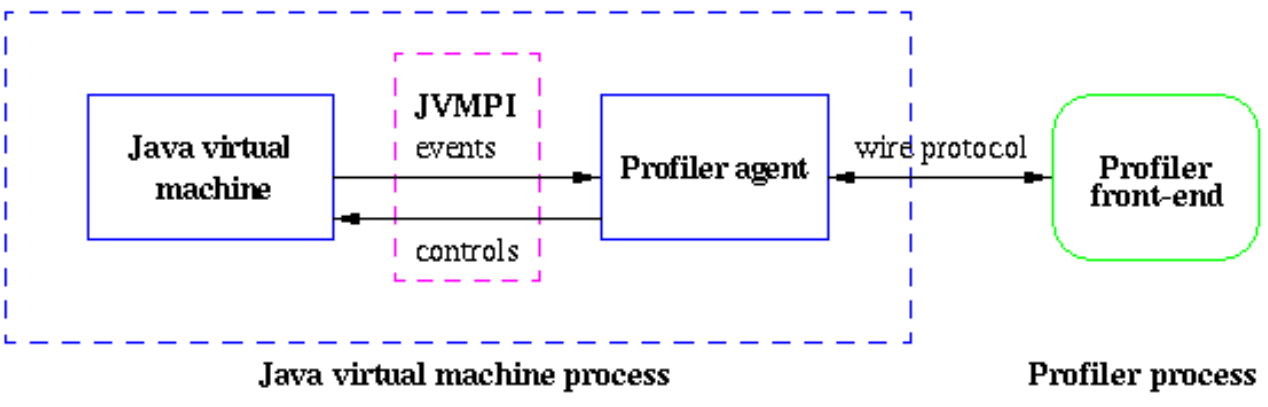

Рисунок 3.2 – Архітектура Java Virtual Machine Profile Interface

Java Virtual Machine Profile Interface – це інтерфейс для відстеження та моніторингу стану віртуальної машини, виділення пам'яті, роботи потоків, завантаження класів і т.д. Для отримання інформації про завантаження класі необхідно реалізувати агента JVMPI, який обробляє подія JVMPI_EVENT_CLASS_LOAD_HOOK [9].

3.3 Винесення критичного програмного коду в окремий захищений модуль

Розглянемо реалізацію підходу винесення програмного коду в захищений модуль використовуючи описану раніше технологію Java RMI.

```
public interface SecretAlgorythm extends Remote {
    public int perform(int i) throws RemoteException;
      }
```

Серверна реалізація алгоритму:

```
public class SecretAlgorithmImpl extends UnicastRemoteObject implements
SecretAlgorythm {

    private static final int CONST = 1;

    public int perform(int i) {
        return i + CONST;
    }
    SecretAlgorithmImpl() throws RemoteException {
    }
}
```

Сервер реєструє реалізацію алгоритму в RMI-реєстрі:

```
public class ServerMain {

     public static void main(String args[]) throws RemoteException,
MalformedURLException {
        System.setSecurityManager(new RMISecurityManager());
```

```
        //create a local instance of the object
        SecretAlgorithmImpl server = new SecretAlgorithmImpl();

        //put the local instance in the registry
        Naming.rebind("SAMPLE-SERVER", server);

        System.out.println("server waiting.....");
    }
}
```

Клієнтська частина, отримує посилання на віддалений об'єкт і виконує алгоритм:

```
public class Client {

    public static void main(String[] args) throws RemoteException,
MalformedURLException, NotBoundException {
        System.setSecurityManager(new RMISecurityManager());

        String url = "//127.0.0.1/SAMPLE-SERVER";
        SecretAlgorythm remoteObject = (SecretAlgorythm) Naming.lookup(url);

        System.out.println(remoteObject.perform(5));
    }
}
```

В даному розділі було проведено реалізація методів захисту Java додатків.

4 ДОСЛІДЖЕННЯ МЕТОДІВ ЗАХИСТУ ВЕБ ДОДАТКІВ

Як показує практичний досвід компаній у проведенні тестування на проникнення та аудиту інформаційної безпеки, уразливості вебдодатків залишаються однією з найпоширеніших вад інформаційної безпеки. Інші поширені проблеми – низька обізнаність працівників з питань інформаційної безпеки, слабка політика щодо паролів або широке її недотримання, недосконалість в процесах управління оновленнями ПЗ, обрані небезпечні конфігурацій і наскільки це не парадоксально, що контроль доступу брандмауера неефективний.

Незважаючи на те, що уразливості вебдодатків неодноразово описувались у "науково-популярній" та спеціалізованій літературі, досить рідко можна знайти превентивні захисні механізми, які зменшують ризик використання в них різних уразливостей [10].

Проблема безпеки вебдодатків ускладняється ще й тим, що при розробці вебдодатків питання, пов’язані із захистом цих систем від різних видів загроз (внутрішніх та зовнішніх), часто не беруться до уваги, і цьому процесу приділяється мало уваги. Це, в свою чергу, створює ситуацію, коли проблеми інформаційної безпеки потрапляють у поле зору власника системи після завершення проєкту. А усунення уразливостей у вже створеному вебдодатку є більш дорогим бюджетом, ніж розробка та впровадження.

Недооцінка серйозності ризику, реалізації загроз інформаційної безпеки за допомогою вебдодатків, доступних з Інтернету, є, мабуть, головним фактором нинішнього низького рівня безпеки для більшості з них.

4.1 Уразливості вебдодатків

Нижче наведено огляд уразливостей вебдодатків, отриманих із двох джерел:

– під час тестування на уразливість, ступінь безпеки та інших проведених досліджень, виконаних фахівцями компанії Positive Technologies в 2020 році;

– перевірка безпеки сайту, яка здійснюється на базі системи MaxPatrol (модуль Pentest) компанією Positive Technologies.

Загалом статистика включає дані про 10457 вебдодатків. Зібрана інформація базуються на 16123 автоматизованому скануванні, детальному розборі 6 вебдодатків, включаючи аналіз вихідного коду понад 10 з них.

Залежно від типу виконуваної роботи були задіяні різні методи проведення опитування вебдодатків, від автоматизованого інструментального опитування методом чорної скриньки, сліпий метод із використанням сканерів безпеки XSpider та MaxPatrol, до всіх ручних перевірок за допомогою "білого ящика" метод. (білий ящик), включаючи частковий та повний аналіз вихідного коду. Статистика включала дані лише про зовнішні вебпрограми, доступні у глобальній мережі Інтернет.

Виявлені уразливості були класифіковані згідно з Класифікацією загроз веббезпеки Консорціуму вебпрограм безпеки (WASC WSTCv2). Ця класифікація є спробою об'єднати загрози для безпеки вебдодатків. Консорціумом з безпеки вебдодатків було створено проєкт, який повинен буде розробляти стандарти термінологій для опису проблем безпеки у вебдодатках. Наявність цього документа дозволяє розробникам додатків, професіоналам безпеки, виробникам програмного забезпечення та аудиторам використовувати єдину мову для взаємодії.

Загальні уразливості вебдодатків представлені в упорядкованому списку з такими класами (WSTCv2):

– аутентифікація;

– авторизація;

– атаки на стороні клієнта;

– виконання коду;

– розкриття інформації;

– логічні недоліки;

– чи небезпечні конфігурації;

– недоліки зловживання протоколами;

Для кожного з класів надається детальний опис включених типів атак. Описи містять приклади уразливостей, які можуть призвести до атаки, а також посилання на додаткові матеріали.

Ця статистика визначає уразливості вебдодатків, а такі проблеми як недолік управління оновленнями ПЗ не йдуть в рахунок. Серйозність уразливості оцінюється відповідно до CVSSv2 (Загальна система оцінки уразливості, версія 2.0) [11].

Розподіл додатків, що вивчаються методом "білого ящика", залежно від сфери діяльності власника, наведено на рис. 4.1.

Рисунок 4.1 – Секторна діаграма в % співвідношенні розподілу власників по галузі

Загалом представлена статистика включала дані про 10458 вебдодатків, 7862 з яких містив одну або кілька уразливостей. Загалом у всіх додатках було знайдено 33932 помилок різних типів ризику.

Таким чином, ймовірність виявлення уразливостей в одному вебдодатки (тобто ефективність оцінки безпеки) при докладному аналізі на 27% перевищує цей

показник під час автоматичного сканування (див. рис. 4.2).

Рисунок 4.2 – Секторна діаграма в % співвідношенні ймовірності виявлення уразливості різними методами їх пошуку

Щоб домогтися кращого результату, потрібно відмовитися від автоматизованого сканування на користь розбору вихідного коду і виконання ручної перевірки. Крім цього, в процесі роботи з вивчення ручної перевірки додатків застосовувалися аналізу, які створені на основі журналів подій і вихідних кодів, завдяки чому обсяг і результат API дозволяє отримати більш точну оцінку досліджених методів безпеки систем. Автоматизоване сканування не налаштовувало профілі сканування для певної вебпрограми, і сканування виконувалось методом "чорного ящика".

Секторна діаграма в процентному відношенні виявлених уразливостей за різними типами за допомогою автоматизованих інструментів показано на рисунку 4.3.

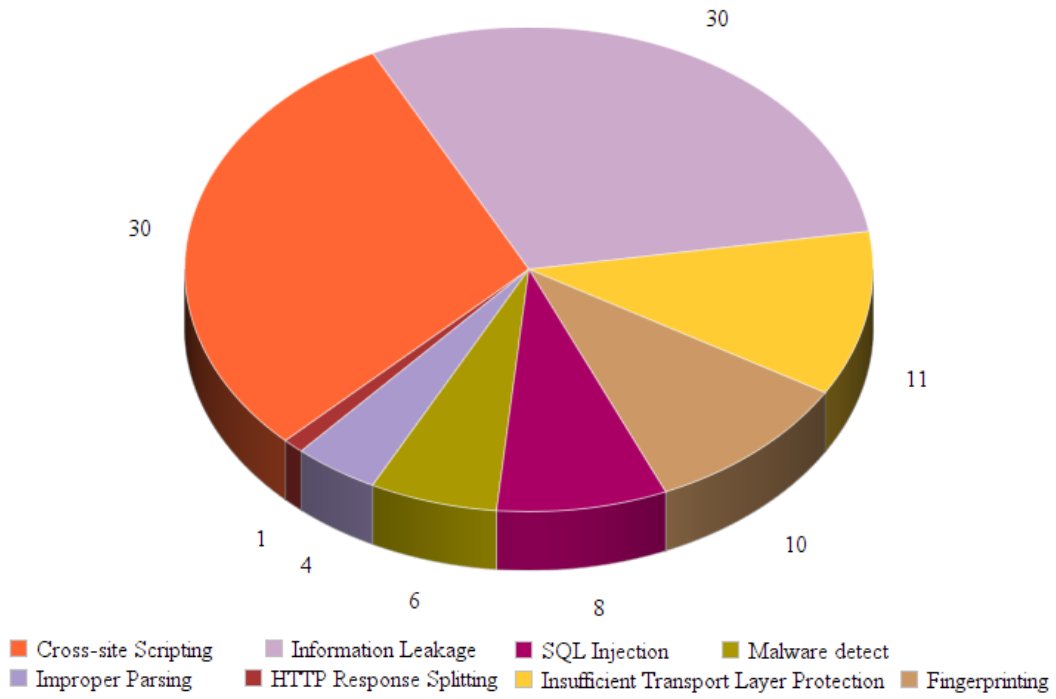

Рисунок 4.3 – Секторна діаграма в % співвідношенні уразливостей вебдодатків (автоматичне сканування)

Дані представлені за системами в яких було знайдено починаючи з однієї і до більшого числа уразливостей. При розрахунку відсотка уразливих вебсайтів не йдуть в рахунок вебдодатки, в яких число уразливостей дорівнювало нулю.

Варто зазначити, що статистика не включає загальну уразливість вебдодатків – підробку міжсайтових запитів (так званий CSRF). Ця помилка знаходилася в тій чи іншій формі у всіх аналізованих додатках.

На міжсайтовий сценарій (XSS) припадає приблизно 25% усіх помилок. Ця уразливість мала місце у 50% усіх аналізованих веб-сайтів. Виходячи з цього, кожен другий вебсайт містить таку ж уразливість.

Інша поширена уразливість, яка наблизилася до "міжсайтових сценаріїв", пов'язана з різними типами витоку інформації. Цей тип уразливості містить такі типові помилки, доступ до вихідного коду сценаріїв сервера, розкриття адреси в каталог вебсервера, отримання різної секретної інформації тощо. Помилки, пов'язані з цією уразливістю, зустрічалися майже на кожному досліджуваному сайті.

Отже, найвищу позицію в рейтингу по частоті виявлення уразливостей в вебдодатках під час автоматичного аналізу займає уразливість крадіжки інформації. Слід зазначити, що ступінь можливого ризику цієї уразливості може коливатися від низького до критичного. Помилки в поділі доступу до ресурсів вебсайтів, зберігання особистих даних для загального доступу, "прихованих" папок, а також зберігання резервних копій сценаріїв говорить про типових прикладах недоліків. У деяких випадках аудитори могли отримати доступ до важливої системної чи ділової інформації (наприклад, до баз даних облікових записів, журналів транзакцій), використовуючи лише механізм примусового перегляду, тобто імена файлів грубої сили, доступні з Інтернету.

Цікаву позицію в статистиці займає уразливість "Виявлення зловмисного програмного забезпечення", на яку припадає близько 7% усіх виявлених уразливостей під час автоматичного сканування. Наявність цієї уразливості свідчить про те, що вебпрограма містить заражений код (Trojan-Spy backdoor, Code.JS, Code.I та ін.), внаслідок чого шкідливе ПЗ може бути встановлене на комп'ютерах відвідувачів такого вебсайту. Статистика уразливостей високої серйозності, виявлених на сайтах, що містять заражений код, показує, що найбільш вірогідними шляхами розповсюдження зараженого коду в цих програмах є використання таких уразливостей:

– введення операторів SQL;

– виконання команд ОС;

– реалізація розширень сервера.

Статистика критичних уразливостей, що містять заражений код, показана на рисунку 4.4.

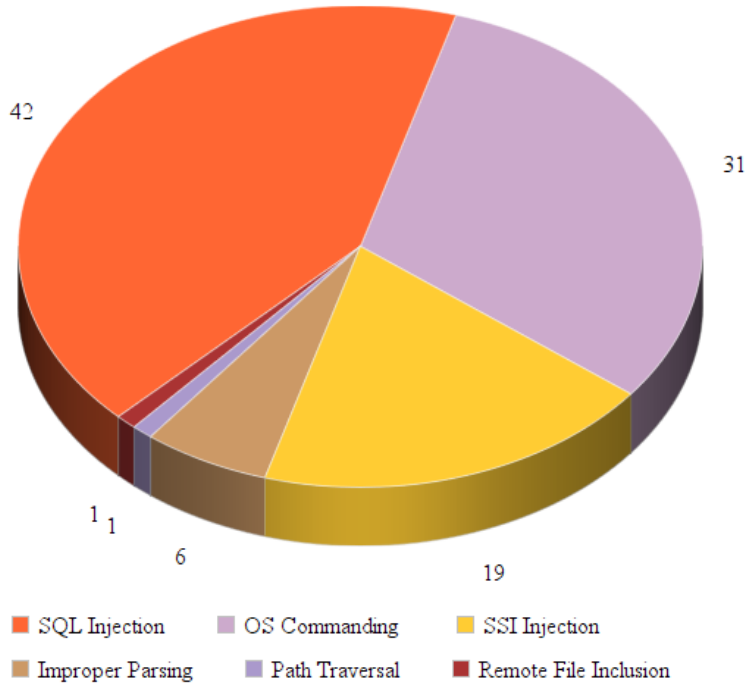

Рисунок 4.4 – Секторна діаграма в % співвідношенні критичних уразливостей, що містять інфікований код

Співвідношення в процентах виявлених уразливостей за різними типами (детальний аналіз вебдодатків), показано на рисунку 4.5.

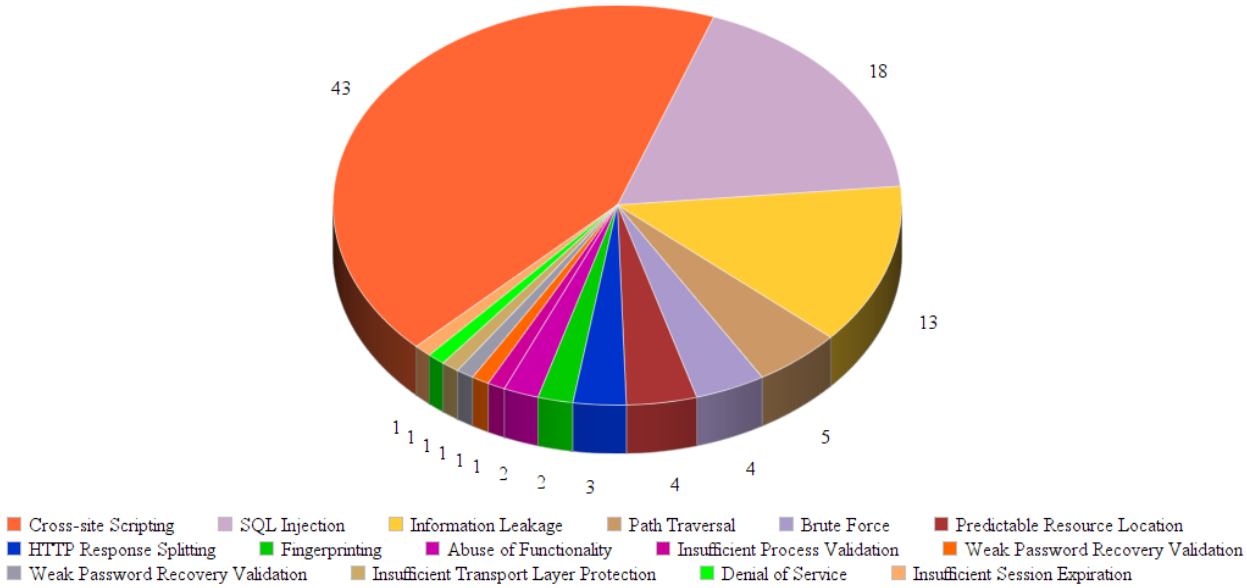

Рисунок 4.5 – Секторна діаграма в % співвідношенні уразливостей вебдодатків (детальний аналіз)

Як і при автоматизованому скануванні вебдодатків, при проведенні детального розбіру найпоширенішою уразливістю як і раніше є міжсайтовий сценарій (XSS), на який припадає приблизно 43% усіх помилок. Ця уразливість була виявлена в 61% усіх аналізованих додатків.

На другому місці, при детальному аналізі безпеки вебдодатків, виявилася уразливість SQL Injection. З цією уразливістю стикалися у 15% випадків, приблизно 67% усіх вивчених програм [12].

Таким чином, провідні позиції щодо ймовірності виявлення уразливості у вебпрограмі під час її детального аналізу посідають уразливість на стороні сервера вебсервера (SQL Injection) та уразливість, що використовується на стороні клієнта – "Міжсайтовий сценарій" (XSS).

Маленький відсоток розповсюдження уразливості витоку інформації порівняно з автоматичним скануванням пояснюється методом, що використовується при детальному аналізі безпеки вебдодатків. Отже, багато вад, які сканер безпеки класифікує як цей тип уразливості та узагальнює їх, під час експертного аналізу були згруповані в одну уразливість або виявилися ознаками інших, більш серйозних проблем (наприклад, помилки контролю доступу) [13].

На основі отриманих даних можна отримати такі висновки:

– найпоширенішими уразливими місцями є «Міжсайтовий сценарій», «Введення SQL», різні типи витоку інформації, «Читання довільних файлів» та вгадування пароля;

– ймовірність виявлення критичної помилки в динамічному вебдодатку становить близько 18% при виконанні автоматичного сканування методом «чорного ящика» та 82% при проведенні всебічного експертного аналізу за методом «білого ящика»;

– можливість автоматизованого інфікування сторінок уразливого вебсистеми становить приблизно 10-15%.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ ЗАХИСТУ ВЕБДОДАТКІВ

5.1 Міжсайтовий скриптинг

XSS (Cross-Site Scripting) – це тип уразливості у вебсистеми, особливість якого полягає в тому, що замість прямої атаки на сервер використовується уразливий сервер зловмисника як засіб для атаки на клієнта.

Зазвичай XSS атака являє собою небезпечне впровадження клієнтського введення в HTML сторінку. Будь-який вебдодаток, що виводить будь-яку інформацію, отриману від користувача, може бути піддано атакам цього класу. Можливість успішного проведення такої атаки виникає в результаті відсутності фільтрації значень змінних, отриманих від користувача, на наявність скриптів (JavaScript) або небезпечних HTML-тегів.

Розглянемо приклад. Припустимо, що існує JSP сторінка index.jsp, яка приймає HTTP параметр "name" і як відповідь генерує HTML сторінку в такий спосіб.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head><title>Simple jsp page</title></head>
  <body>
    Hello <%=request.getParameter("name") %>
  </body>
</html>
```

Таким чином, якщо ми звернемося до цієї сторінки index.jsp?Name=World, то в результаті отримаємо наступний HTML:

```
<html>
  <head><title>Simple jsp page</title></head>
  <body>
    Hello World
  </body>
</html>
```

У разі звернення до цієї сторінки з параметром – index.jsp?name=<script type="text/javascript">alert("XSS")</script>, згенерується сторінка

```
<html>
  <head><title>Simple JSP Page</title></head>
  <body>
    Hello <script type="text/javascript">alert("XSS")</script>
  </body>
</html>
```

Внаслідок чого в браузері виконається javascript код <script type="text/javascript">alert("XSS")</script> (див. рис. 5.1)

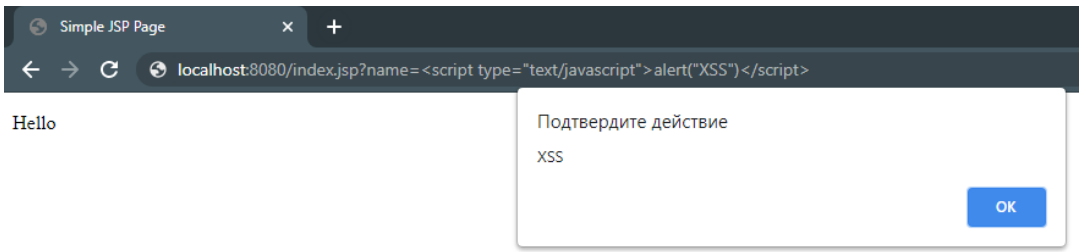

Рисунок 5.1 – Результат виконання XSS атаки

Змінюючи значення параметра "name" можна виконати будь-який javascript код на стороні клієнта в браузері. Як зловмисник може цим використовуватися? В першу чергу, йому необхідно змусити жертву перейти по необхідної йому посиланням (в яку вбудований javascript). На практиці це здійснюється дуже просто, наприклад розміщення посилання в блозі, форумі і т.д.

Що ж дозволяють здійснити XSS-уразливості? Крадіжку конфіденційної інформації користувача. В першу чергу сюди варто віднести крадіжку cookies (document.cookie) як найважливіший атрибут безпеки користувача.

Cookie – це інформація, яку сервер надсилає клієнту. Браузер зберігає cookie та відправляє її як фрагмент заголовка HTTP на сторонній сервер з кожним запитом. Деякі файли cookie зберігаються лише протягом одного сеансу, вони видаляються після закриття браузера. Інші, встановлені на певний проміжок часу, записуються у файл. Якщо необхідно авторизувати або автентифікувати користувачів, тоді файли cookie часто зберігають ідентифікатор користувача та хеш пароля.

Одним з основних методів захисту сайтів від уразливостей XSS є використання різних фільтрів для символів, введених користувачем.

В ході роботи був розроблений фільтр дозволяє блокувати XSS атаки.

API фільтра визначається такими інтерфейсами в пакеті javax.servlet:

– Filter;

– FilterChain;

– FilterConfig.

Фільтр визначається реалізацією інтерфейсу фільтра, головним методом є метод doFilter, який передає запити, відповіді та об'єкти ланцюга фільтрування. Цей метод може зробити наступне:

– перевіряти заголовки запиту;

– змінювати об'єкт запиту, якщо необхідно змінити заголовки або дані запиту;

– змінювати об'єкт відповіді, якщо необхідно змінити заголовки або дані відповіді;

– викликати наступний об'єкт в ланцюжку фільтрів. Якщо поточний фільтр є – останнім фільтром в ланцюжку, що закінчується цільовим вебкомпонентом або статичним ресурсом, наступним об'єктом є останній ресурс в ланцюжку; в іншому випадку, таким об'єктом є наступний фільтр, конфігурований в WAR. Наступний об'єкт викликається за допомогою методу doFilter з об'єктом ланцюжка (передаючи запит і відповідь, з якими він був викликаний, або змінені версії, які він міг створити). В якості альтернативи він може заблокувати запит, не виконуючи завдання в наступному об'єкта. В останньому випадку фільтр відповідальний за заповнення відповіді;

– перевіряти заголовки відповіді після виклику наступного фільтра в ланцюжку;

– згенерувати виняткову ситуацію для індикації помилки при обробці.

Реалізація XSSFilter виглядає наступним чином:

```
public class XssFilter extends BaseFilter {
    public static final String ERROR_PAGE = "/paramError.jsp";
```

```
    private ServletContext servletContext;
    public void init(FilterConfig config) throws ServletException {
        super.init(config);
        servletContext = config.getServletContext();
    }

    protected void subDoFilter(HttpServletRequest request,
HttpServletResponse response,
                               FilterChain chain) throws IOException,
ServletException {
        boolean success = filter(servletContext, request, response);
        if (success) {
            HttpServletRequest requestWrapper = new
CrlfRequestWrapper(request);
            chain.doFilter(requestWrapper, response);
        }
    }

    public static boolean filter(ServletContext servletContext,
HttpServletRequest request,
                                 HttpServletResponse response) throws
IOException, ServletException {
        Map faultyParameters = checkRequestParameters(request);
        if (faultyParameters.size() > 0) {
            logPossibleAttack(request);
            RequestDispatcher dispatcher =
servletContext.getRequestDispatcher(ERROR_PAGE);
            dispatcher.forward(request, response);
            return false;
        }
        return true;
    }

    private static Map<String, String>
checkRequestParameters(HttpServletRequest request) {
        Map<String, String[]> parameterMap = request.getParameterMap();
        Map<String, String> sussParams = new HashMap<String, String>();
        for (Map.Entry<String, String[]> entry : parameterMap.entrySet()) {
```

```
            String[] values = entry.getValue();
            for (String param : values) {
                boolean isSafe = SecurityUtil.isParameterSafe(param);
                if (!isSafe) {
                    sussParams.put(entry.getKey(), param);
                }
            }
        }
        return sussParams;
    }

    private static void logPossibleAttack(HttpServletRequest request) {
        StringBuffer description = new StringBuffer();

        description.append("Suspicious XSS attack strings");
        description.append("           from           IP           address
").append(request.getRemoteAddr());
        description.append(" URL: ");
        description.append(SecurityUtil.reconstructQuery(request));

        System.out.print(description.toString());
    }
}
```

Так як остання версія реалізації J2EE не дозволяє використовувати регулярні вирази для конфігурації URL шаблонів для сервлетів і фільтрів, то був розроблений абстрактний базовий клас BaseFilter, який дозволяє обійти це обмеження.

Налаштування XSSFilter для перевірки всіх запитів крім запитів до ресурсів gif, bmp, jpg, png, css, js, doc:

```
    <filter>
        <filter-name>XssFilter</filter-name>
        <filter-class>edu.nure.security.XssFilter</filter-class>
        <init-param>
            <param-name>include-regex</param-name>
            <param-value>.*</param-value>
        </init-param>
        <init-param>
            <param-name>exclude-regex</param-name>
```

```
            <param-value>
(?:^.*\.gif$)|(?:^.*\.bmp$)|(?:^.*\.jpg$)|(?:^.*\.png$)|(?:^.*\.css$)|(?:^.
*\.js$)|(?:^.*\.doc$)
            </param-value>
        </init-param>
        <init-param>
            <param-name>case-sensitive</param-name>
            <param-value>false</param-value>
        </init-param>
    </filter>
```

Основну функціональність перевірки параметрів виконує клас edu.nure.security. SecurityUtil.

Для того щоб перевірити працездатність цього класу, були розроблені ряд функціональних і юніт тестів edu.nure.security. SecurityUtilTest на базі фреймворка JUnit [14].

Функціональні тести використовують базу шаблонів XSS атак (https://owasp.org/www-community/xss-filter-evasion-cheatsheet).

Відповідь сервера та логування на XSS атаку з включеним XSSFilter наведено на рисунках 5.2-5.3.

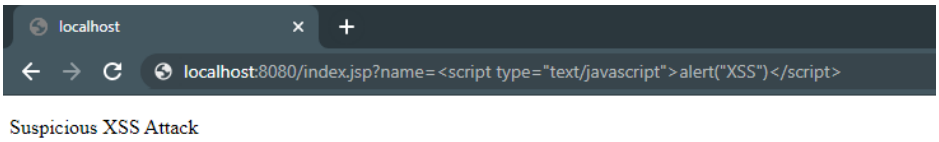

Рисунок 5.2 – Відповідь сервера на XSS атаку

Рисунок 5.3 – Логування XSS атаки

Кожен шаблон був поміщений в окремий файл. Шаблони можна знайти в директорії diploma\web_security\test\resources\xss\. Зараз база шаблонів налічує 100 різних підходів.

5.2 Фіксація сесії

Більшість вебдодатків використовують сесії (сеанси) для створення зручного для користувача середовища. Як уже згадувалося, HTTP протокол не підтримує механізм сесії. Тому, для того щоб штучно підтримувати стан сервера між послідовністю запитів користувача – вебсервери вдаються до різних способів. Основна ідея полягає в тому, що сервер генерує ідентифікатор сесії (ID, в стандарті Java Enterprise Edition – JSESSIONID) на самому початку взаємодії з користувачем, відправляє цей ідентифікатор браузеру користувача і гарантує, що цей же JSESSIONID відправлятиметься назад на сервер при кожному наступному запиті. Таким чином, на сервері JSESSIONID ідентифікує користувача.

У Servlet API (частина стандарту Java Enterprise Edition) за контроль і управління сесії відповідає інтерфейс javax.servlet.http.HttpSession.

Існує три широко поширених способу підтримки сесій в вебдодатках: URL аргументи, приховані поля форми та cookies. У той час як кожен з них має свої переваги і недоліки, cookies довели що є найбільш зручним і безлопатева способом. З точки зору безпеки, більшість, якщо не всі відомих уразливостей в cookie орієнтованому підході підтримки сесії можуть бути застосовані до підходів з URL аргументами і прихованими полями форм, але не навпаки [15].

Дуже часто сесійні ID є не тільки ідентифікаторів користувача, але гарантують, що користувач пройшов аутентифікацію. Це означає, що під час входу в систему, користувач проходить аутентифікацію, засвідчуючи свою особистість, наприклад на підставі імені та пароля або цифрового сертифікату, при цьому на сервері створюється сесія, прив'язана до даного користувачеві, яка гарантує, що

користувач пройшов аутентифікацію.

Фіксація сесії (Session Fixation) є методом вторгнення, який примусово встановлює ідентифікатор сеансу (session ID) в конкретне значення. Залежно від функціональності програми-мішені, щоб "зафіксувати" ідентифікатор сесії може бути застосовано безліч методів. Різноманітність цих методів – від застосування міжсайтового скриптинга до закидання додатки попередньо сформованими HTTP-запитами. Потім ідентифікатор користувача сесії фіксується і зловмисник очікує тих, хто буде входити в систему. Як тільки користувач зробить це, зловмисник використовує зумовлене значення ідентифікатора сесії, щоб отримати його онлайн ідентифікацію з усіма витікаючими наслідками.

Існує два типи систем управління сеансами, залежно від того, як генеруються значення ідентифікатора. Перший тип системи – «вільний», що дозволяє веббраузерам вказувати будь-який ідентифікатор. Другий тип систем – "строгий", який приймає лише значення, що генеруються на сервері. У вільних системах довільний ідентифікатор сеансу обробляється взагалі без будь-якого виклику вебдодатку. Серйозні системи вимагають, щоб зловмисник підтримував "пастку сеансу" з періодичними викликами вебдодатку, щоб запобігти часу очікування бездіяльності.

Без активного захисту від фіксації сеансу атаку можна розпочати проти будь-якої вебпрограми, яка використовує сеанси для автентифікації користувачів. Вебпрограми використовують ідентифікатори сеансів, як правило, файли, також використовуються URL-адреси та поля прихованих форм. На жаль, сеанси на основі файлів cookie легше атакувати. Більшість ідентифікованих в даний час методів атак спрямовані на виправлення файлів cookie.

На відміну від викрадення ідентифікаторів користувацьких сеансів після їх входу у вебпрограму, фіксація сеансу забезпечує набагато ширший спектр атак. Активна частина атаки здійснюється до входу користувачів до системи (див. рис. 5.4).

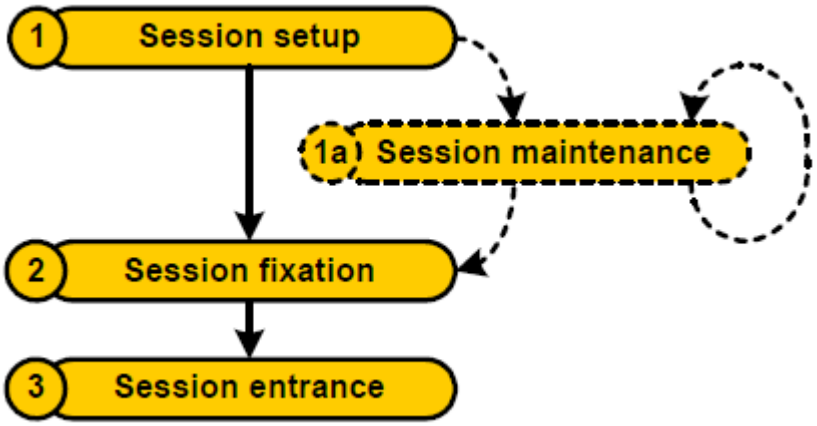

Рисунок 5.4 – Життєвий цикл сесії

Як правило, атака фіксації сеансу складається з трьох етапів:

– налаштування сеансу (зловмисник встановлює "сеанс пастки" на цільовому сайті та отримує цей ідентифікатор сеансу, або зловмисник може вибрати випадковий ідентифікатор сеансу для атаки, але в цьому випадку значення, встановлене сеансом підключення, повинно підтримуватися (бути "активним") із повторним підключенням до вебпрограми);

– фіксація сеансу (зловмисник вводить значення сеансу «перехоплення» в браузер користувача та фіксує ідентифікатор сеансу користувача);

– входження до сеансу (зловмисник чекає, поки користувач увійде на цільовий сайт, коли користувач зробить це, використовується значення фіксованого ідентифікатора сеансу, і зловмисник може "взяти на себе" (див. рис. 5.5).

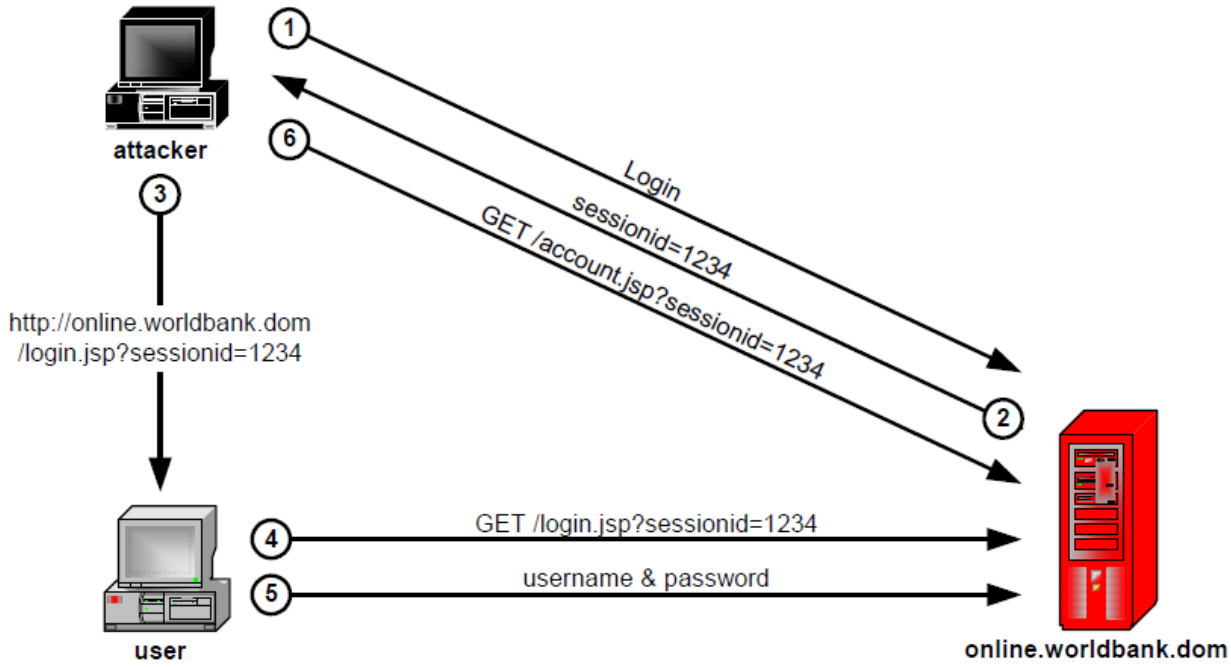

Рисунок 5.5 – Фіксація сесії

Фіксація ідентифікатора користувача сеансу може бути здійснено одним із таких методів:

– видача cookie, використовуючи META тег. Цей метод простіше попереднього, але так само ефективний, особливо коли блокатор XSS запобігає ін'єкцію тегів HTML скриптів, але не META тегів. Фрагмент коду:

http://example/<meta%20http-equiv=SetCookie%20content="jsessionid=1234;%20domain=.example.dom">.idc;

– видача cookie, використовуючи HTTP-заголовок відповіді. Зловмисник викликає або на цільовому вебсайті, або на будь-якому з його піддоменів, видачу cookie з ідентифікатором сесії, що може бути виконано різними способами;

– видача cookie, використовуючи клієнтський скрипт. Видача нового значення ідентифікатора сесії в cookie, використовуючи клієнтський XSS скрипт в слабких місцях в системі захисту на будь-якому піддомені вебсайту може бути використано для зміни поточного значення cookie. Фрагмент коду:

http://example/<script>document.cookie="jsessionid=1234;%20domain=.example.dom";</script>.idc.

Можливі такі способи видачі-файли:

– взлом вебсервера в домені (наприклад, погано підтримуваний WAP сервер);

– зараження користувальницького DNS сервера, практично додавши вебсервер зловмисника в атакується домен;

– установка зловмисного вебсервера, атакується домен (наприклад, на робочій станції в Windows2000 домен, все робочі станції також знаходяться в DNS домені);

– використання атаки розбиття HTTP-відповіді (див. рис. 5.6).

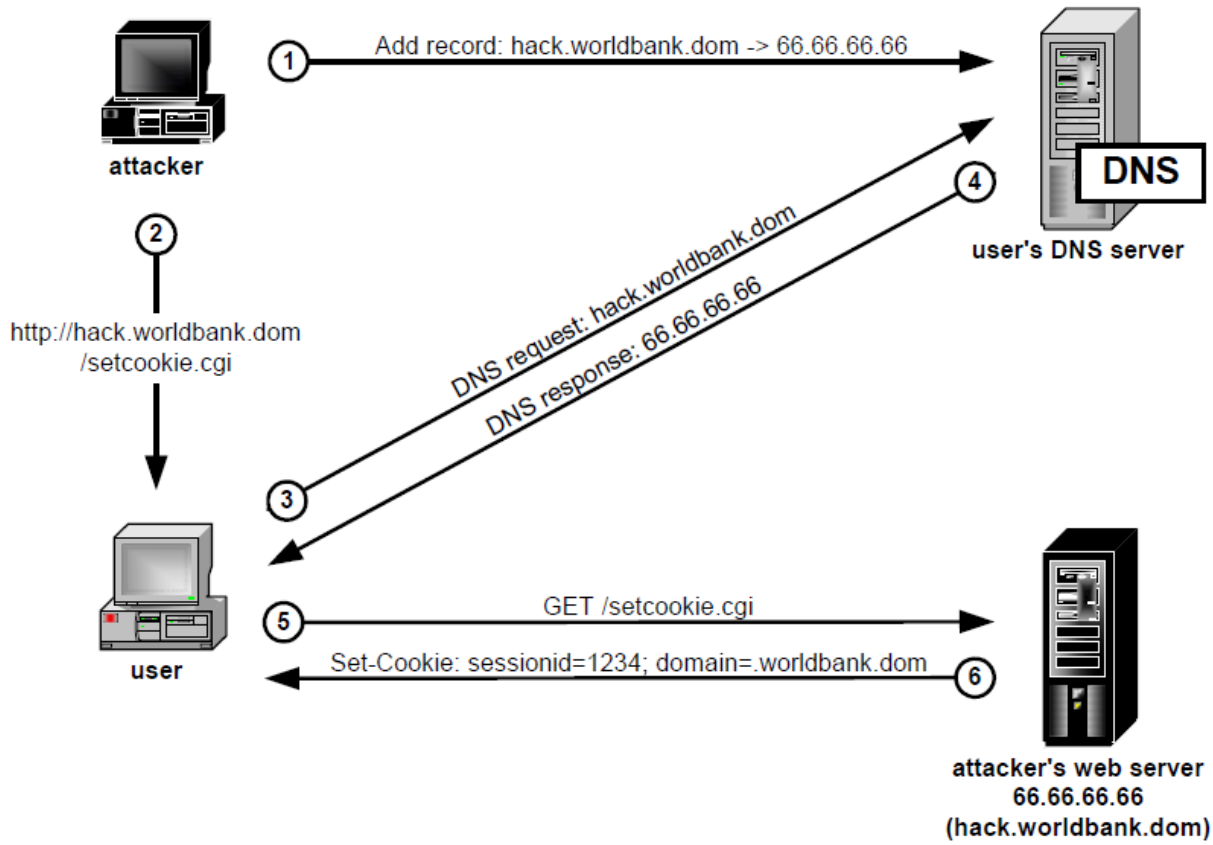

Рисунок 5.6 – Фіксація сеансу, використовуючи атаку на DNS сервер

Довгострокова атака фіксації сесії може бути виконана, видачею постійних cookie, які будуть тримати фіксацію сеансу навіть після перезавантаження користувачем комп'ютера. Фрагмент коду:

http://example/<script>document.cookie=”osessionid=1234;%20Expires=Friday, %201-Jan2017%2000:00:00%20GMT”;</script>.idc

Розглянемо типову реалізацію аутентифікації користувача:

```
public class LoginServlet extends HttpServlet {

    public static final String USER_SESSION_ATTRIBUTE = "USER_SESSION";

    private static final String VALID_USER_NAME = "user1";

    private static final String VALID_USER_PASSWORD = "password1";

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doPost(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String name = request.getParameter("name");
        String password = request.getParameter("password");

        boolean result = false;
        if (VALID_USER_NAME.equals(name) &&
VALID_USER_PASSWORD.equals(password)){
            HttpSession session = request.getSession(true);
            session.setAttribute(USER_SESSION_ATTRIBUTE, new
UserSession(name));
            result = true;
        }

        request.setAttribute("result", result);

getServletContext().getRequestDispatcher("/login.jsp").forward(request,resp
onse);
    }
}
```

Реалізація фільтра авторизації користувача:

```
public class CheckPermissionFilter implements Filter {

    /**
     * Place this filter into service.
     * @param filterConfig The filter configuration object
     */
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    /**
     * Checks user permissions. If user isn't logged in, forwards request
to fail page.
     *
     * @param servletRequest  The servlet request we are processing
     * @param servletResponse The servlet response we are creating
     * @param filterChain     The filter chain we are processing
     * @throws IOException      if an input/output error occurs
     * @throws ServletException if a servlet error occurs
     */
    public void doFilter(ServletRequest servletRequest,
                         ServletResponse servletResponse,
                         FilterChain filterChain)
            throws IOException, ServletException {

        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpSession session = request.getSession(false);
        if (session == null ||
session.getAttribute(LoginServlet.USER_SESSION_ATTRIBUTE) == null) {
            request.setAttribute("reason", "You are not logged in.");
            RequestDispatcher dispatcher =
request.getRequestDispatcher("/error.jsp");
            dispatcher.forward(servletRequest, servletResponse);
        } else {
            filterChain.doFilter(servletRequest, servletResponse);
        }
    }
```

```
    /**
     * Take this filter out of service.
     */
    public void destroy() {
    }
}
```

Розглянемо етапи атаки. Установка зловмисником "сесії-пастки" на сайт-мішень і отримує цей ідентифікатор сесії (див. рис. 5.7).

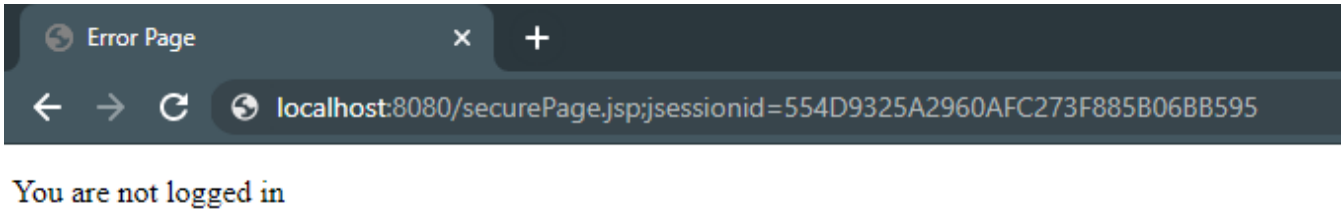

Рисунок 5.7 – Фіксація сесії зловмисником

На етапі фіксації сесії зловмисник поміщає значення сесії пастки в браузер користувача і фіксує ідентифікатор користувача сесії. В даному прикладі ідентифікатор сесії для наочності використовується у вигляді URL параметра [16].

Зловмисник зафіксував ідентифікатор користувача сесії рівній 8D9E9FB6E272137C8F94CE22E88B3F81.

На етапі входження сесії зловмисник очікує поки користувач не увійде в систему сайту-мішені (див. рис. 5.8).

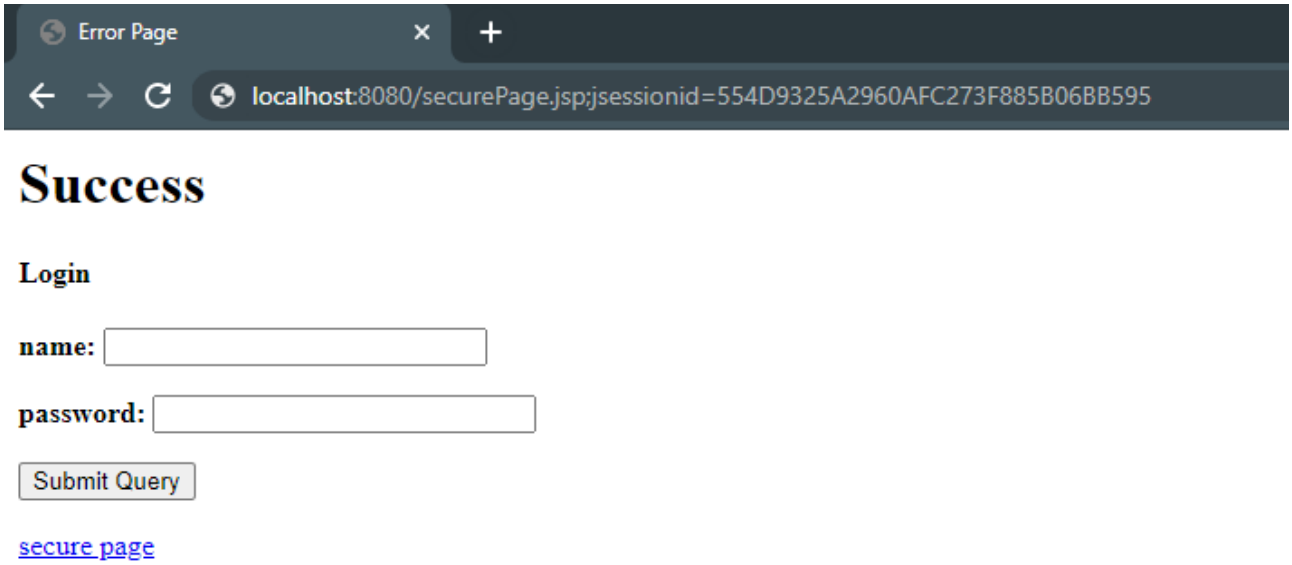

Рисунок 5.8 – Входження сесії

Після входження сесії користувачем, зловмисник може використовувати цей ідентифікатор для несанкціонованого доступу.

Тепер розглянемо підхід до модифікації модулів авторизації і аутентифікації який би дозволив реалізувати захист від фіксації сесії. Ідея полягає в тому, щоб в кожному HTTP запиті передавати додатковий унікальний ідентифікатор [17]. У даній реалізації він називається USER_TOKEN. На стороні сервера даний параметр приєднаний до сесії.

```
public class LoginServlet extends HttpServlet {

    public static final String USER_SESSION_ATTRIBUTE = "USER_SESSION";

    public static final String USER_TOKEN = "USER_TOKEN";

    private static final String VALID_USER_NAME = "user1";

    private static final String VALID_USER_PASSWORD = "password1";

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doPost(request, response);
    }
```

```
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String name = request.getParameter("name");
        String password = request.getParameter("password");

        boolean result = false;
        if (VALID_USER_NAME.equals(name) &&
VALID_USER_PASSWORD.equals(password)){
            HttpSession session = request.getSession(true);
            session.setAttribute(USER_SESSION_ATTRIBUTE, new
UserSession(name));
            result = true;
        }

        request.setAttribute("result", result);

        Cookie userTokenCookie = createUserCookie(request, false);
        response.addCookie(userTokenCookie);
        request.getSession().setAttribute(USER_TOKEN,
userTokenCookie.getValue());


getServletContext().getRequestDispatcher("/login.jsp").forward(request,resp
onse);
    }

    private Cookie createUserCookie(HttpServletRequest request, boolean
makeSecure) {
        String name = USER_TOKEN;
        String value = RandomUtil.getLoginCookieValue(request);
        Cookie cookie = new Cookie(name, value);
        cookie.setSecure(makeSecure);
        return cookie;
    }
     }
```

Змінимо CheckPermissionFilter, додавши валідацію USER_TOKEN

```
     /**
```

```
     * @param servletRequest  The servlet request we are processing
     * @param servletResponse The servlet response we are creating
     * @param filterChain     The filter chain we are processing
     * @throws IOException      if an input/output error occurs
     * @throws ServletException if a servlet error occurs
     */
    public void doFilter(ServletRequest servletRequest,
                         ServletResponse servletResponse,
                         FilterChain filterChain)
            throws IOException, ServletException {

        HttpServletRequest request = (HttpServletRequest) servletRequest;
        HttpSession session = request.getSession(false);
        if (session == null
                ||
session.getAttribute(LoginServlet.USER_SESSION_ATTRIBUTE) == null
                || !validateUserToken(request)) {

            request.setAttribute("reason", "You are not logged in.");
            RequestDispatcher dispatcher =
request.getRequestDispatcher("/error.jsp");
            dispatcher.forward(servletRequest, servletResponse);
        } else {
            filterChain.doFilter(servletRequest, servletResponse);
        }
    }

    private boolean validateUserToken(HttpServletRequest request) {
        HttpSession httpSession = request.getSession();
        String userToken = (String)
httpSession.getAttribute(LoginServlet.USER_TOKEN);
        if (userToken == null) {
            return false;
        }

        Cookie[] cookies = request.getCookies();
        Cookie userTokenCookie = null;
        for (Cookie cooky : cookies) {
```

```
            if (cooky.getName().equals(LoginServlet.USER_TOKEN)) {
                userTokenCookie = cooky;
                break;
            }
        }

        if (userTokenCookie == null) {
            return false;
        }

        return userToken.equals(userTokenCookie.getValue());
    }
```

Переконаємося, що при аутентифікації користувача генерується cookie USER_TOKEN (див. рис. 5.9).

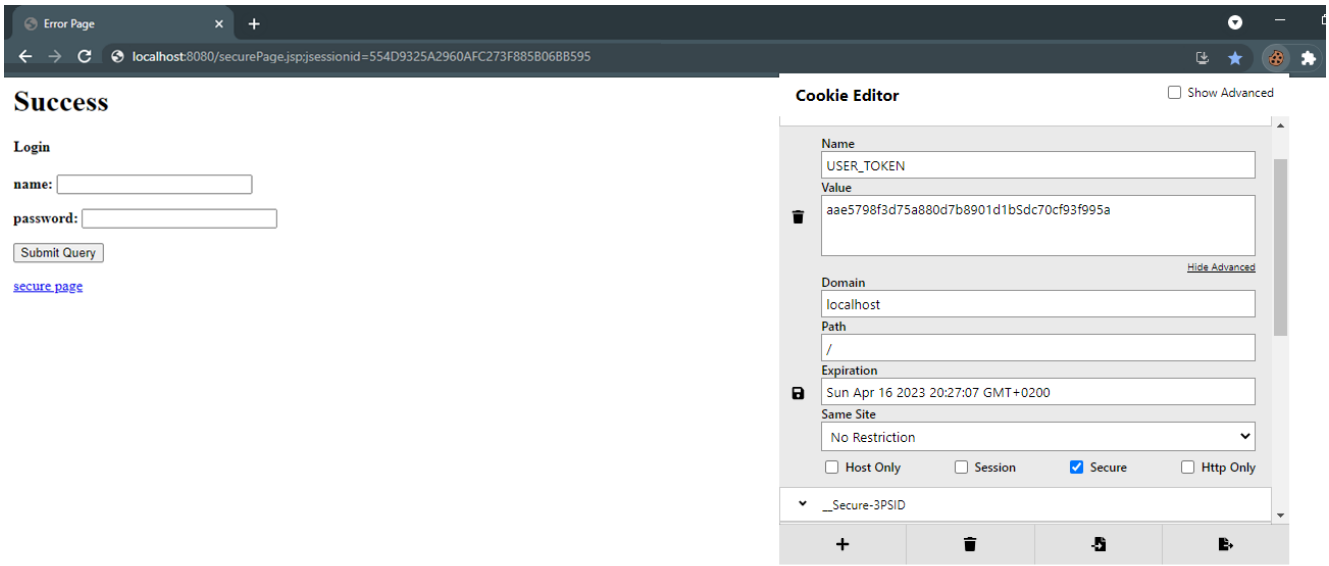

Рисунок 5.9 – Згенерований USER_TOKEN

При цьому підході реалізації модулів аутентифікації і авторизації, зловмисник все також має можливість зафіксувати ідентифікатор JSESSIONID, але для повторної авторизації використовуючи зафіксований JSESSIONID, відсутня інформація про раніше створеному USER_TOKEN. Одночасна фіксація JSESSIONID і USER_TOKEN також не приведе до результата, так як при авторизації користувача для ідентифікатора USER_TOKEN згенерує нового значення.

ВИСНОВКИ

У процесі даної дослідницької роботи були розглянуті теоретичні питання, пов'язані з різними методами захисту Java додатків, а саме: шифрування програмного коду, обфускація, винесення критичного програмного коду в окремий захищений модуль, використання Ahead-Of-Time компіляції. Для кожного методу були проаналізовані переваги і недоліки.

Дослідження показало, що загальний рівень безпеки веб-додатків продовжує знижуватися. Розробники прагнуть забезпечити максимальну функціональність системи і не завжди виділяють свій час для повної безпеки коду. Важливо відзначити широко поширені недоліки, пов'язані з адміністративними помилками. Їх небезпека зазвичай невелика, але в разі експлуатації деяких таких вразливостей порушник може не тільки отримати конфіденційну інформацію (наприклад, в результаті її розкриття на сторінках додатків), але і отримати несанкціонований доступ до системи.

Більш того, додатки, які вже використовуються, не менше потребують ефективному захисту від атак. Більша частина таких ресурсів були схильні до критично небезпечним уразливостям (62%). Ці недоліки можуть привести не тільки до розкриття важливих даних, але і до повної компрометації системи або її виходу з ладу. Тому для захисту від атак рекомендується використовувати брандмауери на рівні додатків.

Комплексне виконання всіх проаналізованих методів в кваліфікаційній роботі дозволяє значно підвищити рівень безпеки веб-ресурсів і знизити ризик компрометації систем до прийнятного рівня.

Результатом даної роботи є дослідження методів захисту додатків розроблений з використанням технології Java, їх програмна реалізація, а також рекомендації щодо їх ефективного використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кей С. Хорстманн, Гарі Корнелл. Java. Бібліотека професіонала. Том 2. Розширені засоби програмування. М: Вільямс – 864 с.

2. JVM Tutorial – Java Virtual Machine Architecture Explained for Beginners / URL: https://www.freecodecamp.org/news/jvm-tutorial-java-virtual-machine-architecture-explained-for-beginners/ (дата звернення: 15.04.2021).

3. O. G. Kachko, D. Televnyi (2019). The kupyna hash function cryptanalysis with the merkle trees signature schemes. Telecommunications and Radio Engineering. Volume 78, Issue 8, DOI: 10.1615/TelecomRadEng.v78.i8.40, С. 683-689 (дата звернення: 28.03.2021).

4. Обфускація і захист програмних продуктів / URL: http://www.citforum.ru/security/articles/obfus/ (дата звернення: 13.03.2021).

5. Guide to Java Reflection / URL: https://www.baeldung.com/java-reflection (дата звернення: 17.04.2021).

6. Gorbenko, I.D., Kachko, O.G., Gorbenko, Yu.I., ...Kandy, S.O., Yesina, M.V. Methods of building general parameters and keys for NTRU Prime Ukraine of 5th – 7th levels of stability. Telecommunications and Radio Engineering (English translation of Elektrosvyaz and Radiotekhnika), 78(7), 2019, С. 579-594 (дата звернення: 02.04.2021).

7. Основи RMI / URL: https://docs.oracle.com/javase/tutorial/rmi/index.html (дата звернення: 13.03.2021).

8. Home Page of Jad – the fast Java Decompiler / URL: http://www.kpdus.com/jad.html (дата звернення: 14.04.2021).

9. JavaTM Virtual Machine Profiler Interface (JVMPI) / URL: https://nick-lab.gs.washington.edu/java/jdk1.4.1/guide/jvmpi/jvmpi.html (дата звернення: 16.04.2021).

10. Уразливості і погрози вебдодатків в 2020 році / URL: https://www.ptsecurity.com/upload/corporate/ru-ru/analytics/web-vulnerabilities-2020-

rus.pdf(дата звернення: 14.03.2021).

11. Common Vulnerability Scoring System / URL: http://www.first.org/cvss/ (дата звернення: 16.03.2021).

12. Gorbenko, I.D., Kachko, O.G., Yesina, M.V. Analysis of asymmetric NTRU prime IIT Ukraine encryption algorithm with regards to known attacks. Telecommunications and Radio Engineering (English translation of Elektrosvyaz and Radiotekhnika), 77(9), 2018, C. 799-816 (дата звернення: 08.04.2021).

13. XSS Filter Evasion Cheat Sheet / URL: https://owasp.org/www-community/xss-filter-evasion-cheatsheet (дата звернення: 21.03.2021).

14. JUnit / URL: http://www.junit.org/ (дата звернення: 17.03.2021).

15. Kachko O., N. Bilous, Semerkov V. Research on methods for secure web applications development Information Technologies in Innovation Business (ITIB), 7-9 October, 2015, Kharkiv, Ukraine Proceedings of ITIB, ISBN 978-966-659-214-2, P. 26-27, (дата звернення: 11.04.2021).

16. Web Application Security Consortium / URL: http://webappsec.org/projects/threat/ (дата звернення: 15.03.2021).

17. I.D. Gorbenko, O.G.Kachko, A.N. Aleksiychuk, O.O. Kuznetsov, Yu.I. Gorbenko, V.V. Onoprienko, M.V. Yesina Algorithms of asymmetric encryption and encapsulation of keys of post-quantum period of 5 -7 stability stability levels and their applications / URL: https://www.researchgate.net/publication/337691068_Algorithms_of_asymmetric_encryption_and_encapsulation_of_keys_of_post-quantum_period_of_5_-7_stability_stability_levels_and_their_applications (дата звернення: 24.03.2021).