© Kovalenko O. Security testing technology based on the provisions of the simulation models scaling theory // Ukrainian Scientific Journal of Information Security, 2018, vol. 24, issue 2, p.110-117.

ЗАХИСТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ОБЛАДНАННЯ / SOFTWARE & HARDWARE ARCHITECTURE SECURITY

DOI: 10.18372/2225-5036.24.13045

ІМІТАЦІЙНА МОДЕЛЬ ТЕХНОЛОГІЇ ТЕСТУВАННЯ БЕЗПЕКИ НА ОСНОВІ ПОЛОЖЕНЬ ТЕОРІЇ МАСШТАБУВАННЯ

Олександр Коваленко

Центральноукраїнський національний технічний університет, Україна

КОВАЛЕНКО Олександр Володимирович, к.т.н.

Рік та місце народження: 1982 рік, м. Кіровоград, Україна.
Освіта: Кіровоградський національний технічний університет, 2004 рік.
Посада: доцент кафедри кібербезпеки та програмного забезпечення.
Наукові інтереси: управління ризиками розробки програмного забезпечення
Публікації: понад 140 публікацій, серед яких статті у фахових виданнях, монографії, патенти, навчальні посібників з грифом МОН України.
E-mail: clashav@gmail.com

Анотація. *У роботі вдосконалена імітаційна модель технологій тестування безпеки Web-додатків. В основу розробки покладено основні положення теорії масштабування імітаційних моделей в рамках алгоритмічного спрощення на основі оцінки транзитивної залежності по управлінню і даним. Відмінною особливістю розробленої імітаційної моделі є адаптація вибору вхідних операторів управління і даних до підвищення вимог оперативності розробки та реалізації моделі, виражена в реалізації процедури взаємодії з реальним браузером з використанням засобів автоматизації браузера і формуванні даних для атаки на декількох діалектах. В імітаційній моделі технології тестування вразливостей пропонується не тільки просте абстрагування від несуттєвого оператора, а і його заміна на більш ефективний, з точки зору точності кінцевих результатів перевірки, оператор. Для зниження витрат часу на імітаційне моделювання було вирішено провести масштабування шляхом заміни процедур інформаційного обміну з сервером за допомогою HTTP клієнта на процедури взаємодії з реальним браузером з використанням засобів автоматизації браузера. Крім основної мети масштабування ця заміна дозволить підвищити достовірність результату тестування атаки з ін'єкцією JavaScript коду. При цьому як засіб автоматизації браузера було вибрано бібліотеку Selenium Webdriver. Однією зі складностей тестування уразливості до SQL ін'єкцій є велика кількість діалектів SQL в залежності від використовуваної системи управління базами даних. Цей факт змушує формувати дані для атаки на декількох діалектах для максимального покриття векторів атаки. З одного боку це збільшує кількість вхідних даних імітаційної моделі, підвищує складність проекту і негативно впливає на загальні часові характеристики реалізації та проведення тестування уразливості. З іншого боку, використовуючи запропонований підхід масштабування можна істотно знизити складність проекту, не погіршуючи якісних характеристик. В основу запропонованого підходу алгоритмічного спрощення імітаційного моделювання прокладені вдосконалені процедури оцінки транзитивної залежності по управлінню і даним. Визначено допустимість і доцільність використання оцінки транзитивної залежності, що знизить обчислювальну складність реалізованих алгоритмів у порівнянні з алгоритмами оцінки прямої залежності до 1,5 раз.*

Ключові слова: *тестування безпеки, масштабування, імітаційна модель, розробка програмного забезпечення, уразливості безпеки.*

Постановка проблеми

Відомим фактом, підтвердженим безліччю актуальних результатів дослідження, є доцільність і актуальність проведення імітаційного тестування з використанням комп'ютерних і телекомунікаційних засобів. Для оцінки результатів математичного моделювання технологій тестування безпеки Web-додатків розробимо і вдосконалимо імітаційну модель. Проведені дослідження показали, що одним з характерних факторів, що впливають на ефективність розроблюваної імітаційної моделі, є істотна

залежність від часу реалізації та експерименту. У той же час більшість існуючих імітаційних моделей мають ряд недоліків, пов'язаних з зайвими витратами обчислювальних ресурсів і часу. У разі моделювання в реальному часі це призводить до зниження точності результатів. Тому виникає необхідність масштабування імітаційної моделі, яка дозволила б знизити обчислювальну, алгоритмічну, технологічну або інші види складності її аналізу без втрати точності моделювання поведінки на заданому рівні абстракції.

Аналіз літератури і постановка задачі

В даний час існує декілька типових ситуацій, коли в процесі імітаційного моделювання можливе використання операцій масштабування. Наприклад, при перевірці властивостей, описаних на більш високому рівні абстракції, ніж сама модель або при перевірці локалізованих властивостей (наприклад, властивостей одного з компонентів великої моделі) [1, 2, 9].

Більшість авторів [1-6] в першому випадку, як правило, модель перебудовують вручну на необхідному (більш високому) рівні абстракції. У другому випадку детальні моделі компонентів, перевірка властивостей яких не передбачається або вважається надмірною, замінюються на «нульові компоненти». Надалі для валідації таких спрощених компонент використовуються знання експертів. Однією з основних проблем такого підходу є оцінка ступеня адекватності і можливості таких спрощень.

У той же час, як показали дослідження, динамічний вибір достатнього рівня моделювання (ступеня масштабування) безпосередньо в ході експерименту може усунути цей недолік.

Тому для аргументованого вибору і розробки способу масштабування проведемо аналіз існуючих підходів і алгоритмів.

Проведені дослідження показали, що в даний час існують різні види залежності між операторами імітаційної моделі. Це транзитивні залежності по даним і управлінню.

Дані види залежності описані в літературі [1-2]. У роботі пропонується використовувати канонічне визначення залежності по даним, що зв'язує два оператори послідовного процесу з уточнення відмінності по значенню змінної. Так само для врахування залежності по управлінню пропонується використовувати результати роботи [4] з уточненням, що враховує залежності, що виникають при зациклені ділянки програми.

Дещо відокремлено в списку залежностей стоять залежності за часом виконання, якщо модельний час виконання одного з них залежить від модельного часу виконання іншого. При цьому потрібно враховувати, що не всі оператори імітаційної моделі просувають модельний час [1]. Цим видом залежності було вирішено знехтувати, в зв'язку з відсутністю технологічної необхідності.

Пропонується розробити імітаційну модель, яка буде відрізнятися від відомих адаптацією вибору вхідних операторів управління і даних до підвищення вимог оперативності розробки та реалізації моделі, що виражається в реалізації процедури взаємодії з реальним браузером з використанням засобів автоматизації браузера і формуванні даних для атаки на декількох діалектах.

Для зниження витрат часу на імітаційне моделювання було вирішено провести масштабування шляхом заміни процедур інформаційного обміну з сервером за допомогою HTTP клієнта на процедури взаємодії з реальним браузером з використанням засобів автоматизації браузера.

Метою роботи є розробка імітаційної моделі технології тестування безпеки на основі положень теорії масштабування імітаційних моделей.

Основна частина дослідження

Основні визначення:

– Вершина n є батьком деякої вершини m (нащадка) в графі $G=(N,E,n_0)$, якщо $(n,m) \in G.E$. Множину всіх нащадків вершини n в графі G будемо позначати як *desc(n,G)*.

– Шляхом «*way*» з $n_i \in G.N$ в $n_k \in G.N$ називається послідовність вершин $n_i, n_{i+1}, \ldots, n_k$, така, що будь-які дві сусідні вершини в ній пов'язані дугою в графі: $(n_j, n_{j+1}) \in G.E$, $j = i, k-1$. Запис $n \in$ «*way*» означає, що вершина n зустрічається на шляху «*way*».

– Шлях «*way*» називається простим, якщо він складається з однієї вершини [1].

– Максимальним називається шлях, який є нескінченним або який закінчується у вершині, що не має нащадків [1].

В основі масштабування по управлінню лежать постулати, що характеризують чутливість операторів до нескінченного зациклення, через поняття максимального шляху [2] в термінах послідовних процесів логічної схеми імітаційної моделі.

У графі управління G послідовного процесу p оператор $n_j \in G.N$ прямо залежить по управлінню чутливо до зациклення від оператора $n_i \in G.N$ тоді і тільки тоді, коли у n_i є два нащадки, n_k та n_z, такі, що:

– у всіх максимальних шляхах, що починаються з n_k, зустрічається n_j, і або $n_i = n_j$, або n_j строго передує будь-якому входженню n_i;

– існує максимальних шлях, що починається з n_z, який, що або в ньому не зустрічається n_j, або n_i строго передує будь-якому входженню n_j.

В роботі [4] наводиться узагальнений алгоритм побудови графа залежностей по управлінню. Однак, проведені дослідження показали, що для даної задачі імітаційного моделювання технології пошуку вразливостей не потрібно знаходження прямої залежності по управлінню. Для коректного моделювання даного процесу досить використовувати більш слабке поняття транзитивної залежності по управлінню. Це істотно знизить обчислювальну складність алгоритму масштабування.

Проведені розрахунки показали, що загальна складність алгоритму дорівнює O (D 2 × | N | 2). Порівняльна оцінка запропонованого алгоритму з відомим алгоритмом, описаним в [2] показала зменшення складності за рахунок заміни циклу по всім керуючим вершинам з множиною символів розміру D × | N |.

Як вже було зазначено вище, залежність по даним описується в роботах [1-3]. У роботі, використ-

товуючи формалізацію відомих визначень [1], уточнимо її вказанням змінної, по значенню якої виникає залежність.

У графі управління G послідовного процесу p оператор $n_j \in G.N$ залежить по даним від оператора $n_i \in G.N$ по змінній v тоді, коли існує змінна $v \in p.V_{ar}$, така, що:

1. існує непростий шлях «*way*» з n_i у n_j, такий, що для будь-якого $n_k \in$ «*way*» $-\{ n_i, n_j \}$[1].

2. $v \in p.def(n_i) \cap p.ref(n_j)$ [2].

Оцінка і розрахунки обчислювальної складності представленого алгоритму показали, що загальна складність алгоритму складе O (| V | × | N | 2).

Скористаємося даними алгоритмами для удосконалення імітаційної моделі технології тестування вразливостей.

Підхід до масштабування імітаційної моделі, що пропонується в роботі, ґрунтується на положеннях відомих теорій масштабування [2-3] в яких зафіксовані основні етапи. Зокрема це виявлення операторів, які не впливають на генерацію і хід подій, спостереження яких потрібно для перевірки заданих властивостей поведінки системи (незначущих операторів) і абстрагування опису імітаційної моделі від таких операторів.

В імітаційній моделі технології тестування вразливостей, що розробляється, пропонується не тільки просте абстрагування від несуттєвого оператора, а і його заміна на більш ефективний, з точки зору точності кінцевих результатів перевірки, оператор.

Для імітаційної моделі технології тестування вразливостей Web-додатків розроблено програмний додаток, що проводить атаки, на які буде вказувати надане користувачем URL посилання, за наведеними алгоритмами, і аналізує результат атак на наявність певної уразливості у Web-додатку.

Як було зазначено вище, для зниження витрат часу на імітаційне моделювання було вирішено провести масштабування шляхом заміни процедур інформаційного обміну з сервером за допомогою HTTP клієнта на процедури взаємодії з реальним браузером з використанням засобів автоматизації браузера. Крім основної мети масштабування ця заміна дозволить підвищити достовірність результату тестування атаки з ін'єкцією JavaScript коду. При цьому як засіб автоматизації браузера було вибрано бібліотеку Selenium Webdriver.

Аналіз літератури показав, що Selenium Webdriver являє собою інструмент автоматизації браузера, який дозволяє розробляти програмні продукти, що мають можливість керувати поведінкою браузера [1-4]. Для роботи з Selenium Webdriver, необхідні наступні програмні компоненти:

– Web-браузер, який буде автоматизовано;

– драйвер для встановленого браузера, який дозволяє автоматизувати поведінку браузера;

– безпосередньо програмний продукт, який складається з набору команд певної мови програмування для драйвера браузера, що надаються спеціальними бібліотеками для багатьох мов програмування. При реалізації імітаційної моделі в якості браузера був обраний Web-браузер Google Chrome. Цей вибір обумовлений тим, що компанією Google в стратегії розвитку архітектури визначено факт, що сьогодні більшість веб-сайтів є не просто вебсторінками, але й веб-додатками. В якості драйвера браузера використаний ChromeDriver, який є імплементацією Webdriver для браузера Google Chrome.

Для розробки програмного продукту, який буде керувати браузером і виконувати атаки на Webдодатки, необхідно вибрати мову програмування. Для реалізації необхідного функціоналу до мови програмування висунуто такі вимоги:

– наявність бібліотеки команд Selenium Webdriver для даної мови програмування;

– можливість роботи з HTTP за допомогою нативних або сторонніх HTTP клієнтів;

– наявність нативної або сторонньої системи журналювання з висновком логу в консоль та зі збереженням логу у текстовому файлі.

Також до мови програмування висунуто такі вимоги для підвищення якості результуючого продукту і підвищення ефективності процесу програмування:

– незалежність мови програмування від оточення, де буде виконуватися програмний продукт;

– наявність сторонніх бібліотек загального призначення;

– наявність інструменту для автоматизованого збору програми і управління зовнішніми залежностями.

Існує декілька мов програмування, що задовольняють цим вимогам, наприклад, Python, Ruby, Java. Однак, для програмної реалізації методу тестування вразливостей Web-додатків обрано мову програмування Java.

Аналіз літератури показав, що мова програмування Java задовольняє вимогам щодо незалежності від оточення за допомогою компіляції вихідного Java коду в байт-код, який є спрощеними машинними командами. Потім програму можна виконати на будь-якій платформі, що має встановлену віртуальну машину Java, яка інтерпретує байт-код в код, пристосований до специфіки конкретної операційної системи і процесора [7]. Зараз віртуальні машини Java існують для більшості процесорів і операційних систем.

Для збору програм, розроблених з використанням мови програмування Java, існує три найбільш поширені інструменти автоматизованого збору - Apache Ant, Apache Maven і Gradle. Для реалізації даного програмного продукту обраний інструмент Apache Maven.

Apache Maven - це засіб автоматизації роботи з програмними проектами, який використовується для Java проектів для управління і збору програм. Для опису програмного проекту який потрібно побудувати (build), Maven використовує конструкцію відому як Project Object Model (POM), залежності від зовнішніх модулів, компонентів і порядку побудови. Maven базується на плагін-архітектурі, що дозволяє зробити використання будь-якої програми контрольованим через стандартний вхід. Даний інструмент має не тільки функціонал зі збором програм, а й по управлінню зовнішніми залежностями і розбиття

програмного продукту на окремі незалежні модулі з подальшим збором в єдиний модуль [8].

Стандартна бібліотека Java надає можливість роботи з мережами, зокрема, протоколом HTTP [7]. Також стандартна бібліотека Java містить класи для роботи текстом, URL, колекціями даних та іншими функціями загального призначення. Для мови програмування Java існує бібліотека команд Selenium Webdriver [1]. Найбільш актуальну версію можна отримати з репозиторія Maven як окрему бібліотеку або використовувати її як зовнішню залежність в Java проекті.

Мова програмування Java має велику кількість бібліотек для журналу з можливостями розширеної конфігурації. Для використання в програмній реалізації методу тестування вразливостей Web-додатків вибрано бібліотеку журналу Apache Log4j2. Бібліотека Apache Log4j2 має велику кількість функцій і можливостей конфігурації журналу, а саме можливість переадресації журналів виконання програми на будь-який потік виведення, наприклад, в текстовий файл на екран, в мережевий сокет та інші [8].

Таким чином, мова програмування Java задовольняє всім вимогам, які були висунуті до мов програмування для можливості реалізації методу тестування вразливостей Web-додатків. Слід зазначити, що для розробки програми буде використано останню версію Java, а саме Java 8.

Для розробки структури програмної реалізації методу тестування вразливостей Web-додатків потрібно визначити множину функціональних вимог, які повинен реалізовувати додаток:

– отримання вхідних параметрів, а саме типу вразливості для тестування і URL веб-сторінки для тестування, в якості параметрів командного рядка;

– для типу вразливості до DOM XSS, програмний продукт повинен отримати код окремої вебсторінки і код всіх зовнішніх JavaScript файлів, які використовуються. Після цього, провести статичний аналіз всього присутнього JavaScript-коду на наявність певних маркерів, які можуть призводити до вразливості;

– для кожного GET-параметра в URL сторінки потрібно визначити тип рефлексії параметра на сторінці і провести XSS атаку з відповідними для типу рефлексії даними і перевірити її результат;

– для типу вразливості SQL Injection, програмний продукт повинен для кожного GET-параметра в URL сторінки провести Boolean blind based SQL ін'єкцію і визначити її результат на основі аналізу тексту веб-сторінок, який повертає сервер;

– детальне логування ходу роботи програмного продукту в консоль і у файл.

Враховуючи дані вимоги до додатка, загальна структура програмної реалізації буде виглядати як багаторівневий проект з наступними модулями:

– модуль інтерфейсу користувача;

– модуль аналізу вразливостей до DOM XSS;

– модуль аналізу вразливості до SQL ін'єкції;

– модуль функцій загального призначення.

Відомо, що Java-програма являє собою набір скомпільованих Java класів, що зібрані у виконуваний архів типу .jar. Проведений аналіз літератури показав, що для збірки програмного продукту в .jar файл існує декілька способів. В даному проекті, в якості інструменту збору проекту був обраний Apache Maven [7].

Дослідження показали, що життєвий цикл збору Maven проекту містить фіксований набір фаз, що виконуються одна за одною. Життєвий цикл за замовчуванням складається з наступних фаз:

– validate (виконується перевірка коректності проекту з точки зору Maven);

– compile (виконується компіляція файлів вихідного коду);

– test (здійснюється тестування скомпільованої коду за допомогою наданого набору модульних тестів);

– package (реалізується упаковка скомпільованої коду в певну форму дистрибуції, наприклад в JAR архів);

– install (переміщення результату упаковки у локальний репозиторія для використання у локальних проектах);

– deploy (переміщення результату упаковки у віддалений репозиторія для використання іншими розробниками).

Apache Maven використовує один або кілька файлів *pom.xml* для подання в форматі XML структури Java проекту. Project Object Model (POM) і містить інформацію про структуру проекту, його внутрішніх і зовнішніх залежностях, інформацію про процес збору проекту і плагінів, які будуть використані при певних фазах збору проекту [6].

Maven підтримує структуру багатомодульних проектів за допомогою наслідування і агрегації проектів. Це досягається введенням додаткових артефактів зі спеціальним маркером в файлах *pom.xml*. При використанні такої конфігурації, загальні характеристики і налаштування властивості можна виносити в базовий файл *pom.xml*, який буде успадкований проектами-нащадками.

Мінімальними необхідними елементами файлу *pom.xml* є координати проекту, що містять наступні елементи:

– groupId – унікальний ідентифікатор проекту або організації, яка розробляє проект. Використовується як аналог пакету в мові Java.

– artifactId – унікальний ідентифікатор артефакту в проекті. Артефактом може бути як сам проект, так і модуль багатомодульного проекта.

– version – версія артефакту. Використовується для версіонування артефактів в репозиторії.

– packaging – тип артефакту проекту Maven. У загальному випадку має значення *jar* для звичайних проектів або окремих модулів, і *pom* для багатомодульного проекту (агрегатора).

Для програмної реалізації методу тестування вразливостей Web-додатків була обрана багатомодульна структура, отже, Maven проект також буде розроблений як багатомодульний проект з декількома модулями. Таким чином, враховуючи загальну структуру програми, Maven проект буде складатися з кореневого проекту, проекту модулю інтерфейсу, проекту модулю функцій загального призначення, кореневого проекту модулів аналізу вразливостей і

безпосередньо проектів модулів аналізу вразливостей. Запропонована структура Maven проекту є розширюваною, так як наявність кореневих проектів дозволяє легко під'єднати додаткові модулі аналізу вразливостей.

Кореневий Maven проект (*dxss-sqli-framework*) визначає ідентифікатор групи для всіх дочірніх проектів як *com.kntu.mtf.koval*. Проект є агрегатором, тому параметр *packaging* має значення *pom*.

```
<groupId> com.kntu.mtf.koval. </groupId>
<artifactId>dxss-sqli-framework</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>pom</packaging>
```

Кореневий проект є агрегатором модулів, тобто проект містить посилання на всі дочірні проекти, що використовуються в додатку.

```
<modules>
 <module>framework-core</module>
 <module>attack-modules</module>
 <module>core-utils</module>
 </modules>
```

Проект також містить залежності, які використовуються усіма дочірніми проектами, а також вказівку Maven використовувати версію 8 мови програмування Java при компіляції вихідних файлів.

```
<build>
<plugins>
 <plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>2.1</version>
 <configuration>
 <source>1.8</source>
target>1.8</target>
 </configuration>
 </plugin>
</plugins>
</build>
```

Проект модуля функцій загального призначення називається *core-utils*. Проект має тип упаковки *jar* і є дочірнім до проекту *dxss-sqli-framework*.

```
<artifactId>core-utils</artifactId>
<packaging>jar</packaging>
<parent>
<artifactId>dxss-sqli-framework</artifactId>
<groupId>com.kntu.mtf.smirn.</groupId>
<version>1.0-SNAPSHOT</version>
</parent>
```

Кореневий проект модулів аналізу вразливостей називається *attack-modules*. Проект є агрегатором, тому параметр *packaging* має значення *pom*. Даний проект є дочірнім для *dxss-sqli-framework* і містить посилання на проект *core-utils* як залежність.

```
<artifactId>attack-modules</artifactId>
<packaging>pom</packaging>
<parent>
<artifactId>dxss-sqli-framework</artifactId>
<groupId> com.kntu.mtf.koval. </groupId>
<version>1.0-SNAPSHOT</version>
</parent>
<modules>
<module>sqli-module</module>
<module>dxss-module</module>
 </modules>
```

```
<dependencies>
<dependency>
<groupId> com.kntu.mtf.koval. </groupId>
 <artifactId>core-utils</artifactId>
<version>1.0-SNAPSHOT</version>
</dependency>
</dependencies>
```

Проект модуля аналізу вразливості до DOM XSS називається *dxss-module*. Проект має тип упаковки *jar* і є дочірнім до проекту *attack-modules*.

Проект модуля аналізу вразливості до SQL ін'єкції називається *sqli-module*. Проект має тип упаковки *jar* і є дочірнім до проекту *attack-modules*.

Проект модуля інтерфейсу називається *framework-core*. Проект має тип упаковки *jar*, є дочірнім для *dxss-sqli-framework* і містить посилання на проекти *sqli-module* і *dxss-module* як залежності.

У розробленій імітаційній моделі технології тестування вразливостей, модуль інтерфейсу відповідає за запуск програми, обробку параметрів командного рядка, і виконання тесту Web-додатка на наявність певної вразливості в залежності від переданих параметрів командного рядка.

Для обробки параметрів командного рядка з метою максимальної масштабованості використаний шаблон проектування «Команда». «Команда» – це шаблон проектування, який відноситься до класу шаблонів поведінки [9]. Він інкапсулює запит в формі об'єкта, що, в свою чергу, дозволяє використовувати єдиний інтерфейс виконання різних дій. Даний шаблон проектування доцільно використовувати при роботі з командним рядком, коли існує більш ніж один варіант роботи програми. UML діаграма шаблону проектування «Команда» приведена на рис. 1.

Структурні елементи шаблону виконують наступні функції:

– Command – об'являє інтерфейс виконання операції;

– ConcreteCommand – реалізує конкретну команду, викликаючи певні методи об'єкту Receiver;

– Client – створює об'єкти ConcreteCommand і встановлює її отримувачів;

– Invoker – звертається до команд з ціллю виконання.

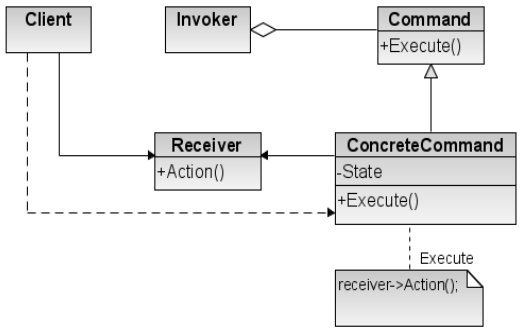

Рис. 1. UML діаграма шаблону проектування «Команда»

У реалізації шаблону «Команда» присутній інтерфейс Command, його імплементації SqlInjectionAttack і DomXssAttack, а також контейнер команд CommandHolder.

В якості структурного елементу Invoker шаблону проектування «Команда» виступає клас Shell, який приймає параметри командного рядка від класу Application і використовує їх для отримання потрібного об'єкта – команди і його виконання.

Як було зазначено вище, модуль аналізу вразливостей до DOM XSS призначений для аналізу окремої Web-сторінки на наявність вразливості до DOM XSS. Відповідно до алгоритму тестування вразливості до DOM XSS, модуль має класи для отримання JavaScript коду Web-сторінки, класи для аналізу JavaScript коду і клас для проведення атак.

Головним класом модуля є клас *DomXssAnalyzer*, який відповідає за безпосередньо аналіз Web-сторінки, на яку вказує переданий URL на наявність вразливості до DOM XSS. Даний клас компонує всі класи модуля для виконання аналізу і є відповідальним за визначення типу рефлексії GET параметра URL, а також безпосередньо за проведення атаки.

Клас DomScanner використовується для аналізу вихідного HTML коду сторінки і отримання списку всіх тегів *script* з їх змістом. Клас містить потоковий HTML «парсер» у вигляді класу *ResponseHtmlHandler*, який формує список тегів *script*.

Для виконання аналізу JavaScript коду сторінки на наявність маркерів вразливості до DOM XSS атак, використовуються класи JavaScript і JavaScriptScanner. Клас JavaScript використовується для опису характеристик одиниці JavaScript коду Web-сторінки, а саме його тип (зовнішній або внутрішній), посилання на зовнішній файл, безпосередньо JavaScript код і список маркерів вразливості, який заповнюється після аналізу. Клас JavaScriptScanner використовується для отримання змісту зовнішніх JavaScript файлів і для аналізу JavaScript коду на наявність маркерів вразливості у вигляді стоків і витоків.

Клас PayloadGenerator використовується для генерації даних для атаки в залежності від типу рефлексії параметра GET запиту URL.

Модуль аналізу уразливості до SQL ін'єкції призначений для аналізу окремої Web-сторінки на наявність вразливості до SQL ін'єкції через GET параметр URL Web-сторінки.

Головним класом модуля є клас SqlInjectionAnalyzer, який відповідає за проведення Boolean blind SQL ін'єкції і аналізу результату атаки з метою визначення наявності потенційної вразливості.

Однією зі складностей тестування уразливості до SQL ін'єкцій є велика кількість діалектів SQL в залежності від використовуваної системи управління базами даних. Цей факт змушує формувати дані для атаки на декількох діалектах для максимального покриття векторів атаки. З одного боку це збільшує кількість вхідних даних імітаційної моделі, підвищує складність проекту і негативно впливає на загальні часові характеристики реалізації та проведення тестування уразливості. З іншого боку, використовуючи запропонований підхід масштабування можна істотно знизити складність проекту, не погіршуючи якісних характеристик.

Враховуючи це та з метою усунення дублювання коду, в додатку реалізований інтерфейс DBMSSpecificPayload, який описує метод, який повертає список даних для проведення атаки з урахуванням конкретної системи управління базами даних. Класи, що реалізують даний інтерфейс, є Common, Oracle та MySql для загальних даних атаки, даних, специфічних для систем управління базами даних Oracle і MySQL відповідно.

Дані для атаки зберігаються в додатку у вигляді класу BooleanBlindPayload, який включає в себе необхідні для проведення атаки дані.

Модуль функцій загального призначення надає набір функцій для роботи з URL посиланнями і їх параметрами, а також для взаємодії з сервером безпосередньо за допомогою протоколу HTTP.

Для перетворення URL і маніпуляції параметрами URL використовується клас UrlUtils. Оскільки функції перетворення URL і маніпуляції їх параметрами не мають стану, вони реалізовані як статичні методи.

Для взаємодії з сервером по протоколу HTTP використовується клас HttpUtils. Даний клас реалізує шаблон проектування «Будівельник» для будівництва внутрішнього класу HttpGetRequest. «Будівельник» – шаблон проектування, дозволяє відокремити конструювання комплексних об'єктів від їх реалізації [1]. В даному випадку, шаблон використаний для конструювання складного об'єкта класу HttpGetRequest, який інкапсулює результат виконання GET запиту протоколу HTTP.

Імітаційна модель технології тестування вразливостей Web-додатків є комплексним програмним забезпеченням з великою кількістю модулів і класів, що в свою чергу, створює проблему компонування класів між собою. Цієї проблеми можна уникнути зменшенням кількості класів, але це призведе до збільшення кількості сильно пов'язаних між собою класів, що в свою чергу ускладнить розробку, підтримку, читабельність і масштабованість програмного забезпечення. Для вирішення задачі компонування класів використано механізми «впровадження залежностей» [2, 3].

З літератури [1-3, 9] відомо, що «впровадження залежностей» – це шаблон проектування, в якому залежності (або сервіси) впроваджуються або передаються по посиланню в залежний об'єкт (клієнт) і стають частиною клієнтського стану. Шаблон відокремлює створення залежностей клієнта від власної логіки клієнта, дозволяє компонентам бути слабо пов'язаними і дотримуватися принципів інверсії залежностей і єдиної відповідальності. Існує три основних типи «впровадження залежностей» в клас:

– впровадження в конструктор;
– впровадження у властивість;
– впровадження в метод.

У загальному випадку шаблон «впровадження залежностей» не прив'язаний до програмної реалізації і не вимагає окремих бібліотек або фреймворків для реалізації, однак, використання контейнерів для «впровадження залежностей» значно спрощує процес розробки.

Одним з найпопулярніших контейнерів для «впровадження залежностей» в екосистемі Java є Spring Framework. Spring Framework – фреймворк

для додатків, розроблених на мові програмування Java, який надає контейнер для впровадження залежностей, підтримку аспектно-орієнтованого програмування і багато інших функцій [10].

Проведені дослідження показали, що актуальною на момент розробки є версія 4.3.8 від 18.04.2017, що свідчить про постійний розвиток фреймворку.

Контейнер Spring дозволяє централізувати створення об'єктів програми та автоматично вирішити всі залежності створених компонентів. При цьому, контейнер бере на себе завдання по створенню компонентів, управлінню їх життєвими циклами і знищенню компонентів при знищенні контейнера. Spring також підтримує ще один вид «впровадження залежностей», а саме ін'єкцію в приватне поле.

Для вирішення залежностей Spring використовує зовнішній конфігураційний файл у вигляді XML файлу або Java класу. Використання контейнера Spring має наступні етапи: 1. Конфігурація контейнера у зовнішньому файлі. 2. Створення контексту програми з посиланням на файл конфігурації. 3. Отримання необхідних компонентів з контексту.

Для розробки імітаційної моделі технології тестування Web-додатків було вибрано конфігурацію контейнера для «впровадження залежностей» у вигляді Java класу. Кожен модуль багатомодульного проекту містить свій клас з конфігурацією, який відповідає за створення компонентів тільки свого модуля, після чого всі конфігураційні файли будуть використані при створенні контексту програми, що, в свою чергу, призведе до створення контейнера для «впровадження залежностей» Spring і створення всіх необхідних компонентів програми.

Клас конфігурації компонентів Spring має певну структуру. Мінімальною вимогою до класу є анотування класу анотацією @Configuration, що дозволяє використовувати клас для конфігурації контейнера для впровадження залежностей під час запуску програми. Клас конфігурації повинен містити методи, що створюють компоненти програми. Дані методи повинні бути анотовані анотацією @Bean. При використанні Spring, для повного впровадження всіх залежностей всі компоненти повинні бути створені в конфігураційних класах. У разі використання декількох конфігураційних класів, повинен існувати головний конфігураційний клас, анотований крім анотації @Configuration також анотацією @Import, що містить посилання на всі інші конфігураційні класи. Опишемо процедуру тестування імітаційної моделі на прикладі аналізу вразливостей до DOM XSS. Для тестування імітаційної моделі у відповідному режимі потрібно запустити зібраний проект з параметрами командного рядка *dxss* [URL]. Для тестування було обрано Web-додаток *https://xssgame-.appspot.com/level1*, який надано фірмою Google для тренування навичок інженерів з інформаційної безпеки. Проведений аналіз показав, що даний Web-додаток створено з відомою вразливістю до DOM XSS атак. Розроблений додаток в режимі тестування має наступні параметри командного рядка: *dxsshttps://xss-game.appspot.com/level1/frame?query=123.*

Після запуску програми відкривається вікно браузера з відповідним URL.

Після відкриття Web-сторінки виконується аналіз JavaScript коду на сторінці. При аналізі в журналі додатка виводиться інформація про знайдені JavaScript файли і їх типи.

Після аналізу Javascript коду на наявність маркерів вразливості виконується визначення типу рефлексії параметра URL. Для цього генерується унікальне значення і вставляється як значення параметра URL, після чого виконується перехід по посиланню.

Потім, виходячи з типу рефлексії параметра, визначається набір даних для атаки і проводиться атака з кожним набором даних. Для простоти тестування в якості маркера було використано діалогове вікно Alert. У разі успішної атаки набір даних записується (логується) в текстовий файл.

Висновки

В роботі удосконалена модель технології тестування безпеки на основі положень теорії масштабування імітаційних моделей. Відмінною особливістю розробленої імітаційної моделі є адаптація вибору вхідних операторів управління і даних до підвищення вимог оперативності розробки та реалізації моделі, виражена в реалізації процедури взаємодії з реальним браузером з використанням засобів автоматизації браузера і формуванні даних для атаки на декількох діалектах.

В основу запропонованого підходу алгоритмічного спрощення імітаційного моделювання прокладені вдосконалені процедури оцінки транзитивної залежності по управлінню і даним. Визначено допустимість і доцільність використання оцінки транзитивної залежності, що знизить обчислювальну складність реалізованих алгоритмів у порівнянні з алгоритмами оцінки прямої залежності до 1,5 раз.

Подальші дослідження направлені на визначення ефективності запропонованої технології безпеки додатків та оцінки достовірності отриманих результатів математичного моделювання.

Література

[1] Ranganath V., Amtoft T., Banerjee A., Dwyer M., Hatcliff J. A new foundation for controldependence and slicing for modern program structures. Technical report 8. Santos lab. Kansas State University. 2004. P. 428–434.

[2] Савенков К. О. Использование зависимостей при масштабировании имитационных моделей. *Методы и средства обработки информации: труды 2 Всероссийской научной конф.* Москва: МГУ им. М.В. Ломоносова. Москва, 2005. С. 28-37.

[3] Семенов С.Г., Швачич Г.Г., Карпова Т.П., Волнянський В.В. Застосування багатопроцесорних систем для удосконалення технологічних процесів. *Системи обробки інформації*. 2016. №3(140). С.221-226.

[4] Коваленко А.В., Смирнов А.А., Якименко Н.Н., Доренский А.П. Проблемы анализа и оценки рисков информационной деятельности. *Системи обробки інформації*. 2016. № 3(140). С. 40-42.

[5] Коваленко А., Смирнов А., Якименко Н., Доренский А.П. Метод качественного анализа рисков разработки программного обеспечения. *Наука і техніка Повітряних Сил Збройних Сил України*. 2016. № 2(23). С. 150-158.

[6] Коваленко А.В. Метод управления рисками разработки программного обеспечения. *Системи управління, навігації та зв'язку*. 2016. №2 (38). С. 93-100.

[7] Maven – Introduction: URL: https://maven.apache.org/what-is-maven.html.

[8] Maven – POM Reference. URL: https://maven.apache.org/pom.html.

[9] Fowler M. Inversion of Control Containers and the Dependency Injection pattern: URL: https://martinfowler.com/articles/injection.html.

[10] Spring Framework. URL: http://projects.spring.io/spring-framework/.

УДК 004.41:004.056 (045)

Коваленко А.В. Технология тестирования безопасности на основе положений теории масштабирования имитационных моделей

Аннотация. *В работе усовершенствована имитационная модель технологий тестирования безопасности Web-приложений. В основу разработки положены основные положения теории масштабирования имитационных моделей в рамках алгоритмического упрощения на основе оценки транзитивной зависимости по управлению и данным. Отличительной особенностью разработанной имитационной модели является адаптация выбора входных операторов управления и данных к повышению требований оперативности разработки и реализации модели, выраженная в реализации процедуры взаимодействия с реальным браузером с использованием средств автоматизации браузера и формировании данных для атаки на нескольких диалектах. В имитационной модели технологии тестирования уязвимостей предлагается не только простое абстрагирование от несущественного оператора, но и его замена на более эффективный, с точки зрения точности конечных результатов проверки, оператор. Для снижения затрат времени на имитационное моделирование было решено провести масштабирование путем замены процедур информационного обмена с сервером с помощью HTTP клиента на процедуры взаимодействия с реальным браузером с использованием средств автоматизации браузера. Помимо основной цели масштабирования эта замена позволит повысить достоверность результата тестирования атаки с инъекцией JavaScript кода. При этом в качестве средства автоматизации браузера было выбрано библиотеку Selenium Webdriver. Одной из сложностей тестирования уязвимости к SQL инъекций является большое количество диалектов SQL в зависимости от используемой системы управления базами данных. Этот факт заставляет формировать данные для атаки на нескольких диалектах для максимального покрытия векторов атаки. С одной стороны это увеличивает количество входных данных имитационной модели, повышает сложность проекта и негативно влияет на общие временные характеристики реализации и проведения тестирования уязвимости. С другой стороны, используя предложенный подход масштабирования можно существенно снизить сложность проекта, не ухудшая качественных характеристик. В основу предлагаемого подхода алгоритмического упрощения имитационного моделирования проложены усовершенствованные процедуры оценки транзитивной зависимости по управлению и данным. Определены допустимость и целесообразность использования оценки транзитивной зависимости, снизит вычислительную сложность реализуемых алгоритмов по сравнению с алгоритмами оценки прямой зависимости до 1,5 раз.*

Ключевые слова: *тестирование безопасности, масштабирование, имитационная модель, разработка программного обеспечения, уязвимости безопасности.*

Kovalenko O. Security testing technology based on the provisions of the simulation models scaling theory

Abstract. *The simulation model of web- application security testing technologies is improved in the work. The basis of the development is the basic provisions of the theory of scaling imitation models in the framework of algorithmic simplification based on the evaluation of transitive dependence on management and data. A distinctive feature of the developed simulation model is the adaptation of the choice of input control-flow statements and data to an increase in the requirements for the speed of development and implementation of the model, which resulted in the implementation of the procedure for interacting with a real browser using browser automation tools and generating attack data in several dialects. In order to reduce the time spent on simulation modeling, it was decided to conduct scaling by replacing the information exchange procedures with the server using an HTTP client with the procedures of interacting with a real browser using browser automation tools. In addition to the main purpose of scaling, this replacement will increase the reliability of the result of testing the attack with the injection of JavaScript code. In this case, the Selenium Webdriver library was chosen as a mean of browser automation. One of the difficulties of testing vulnerability to SQL injections is a large number of SQL dialects depending on the database management system used. This fact forces the generation of data for an attack in several dialects to maximize coverage of attack vectors. On the one hand, this increases the amount of input data of the simulation model, increases the complexity of the project and negatively affects the overall time characteristics of implementation and testing of the vulnerability. On the other hand, using the proposed scaling approach, the complexity of the project can be significantly reduced without degrading the qualitative characteristics. The proposed approach of algorithmic simplification of simulation modeling is based on advanced procedures for estimating transitive control and data dependencies. The admissibility and expediency of using the estimation of the transitive dependency has been determined, which will reduce the computational complexity of the implemented algorithms in comparison with the algorithms for estimating the direct dependency up to 1,5 times.*

Key words: *security testing, scaling, simulation model, software development, security vulnerabilities.*

Отримано 25 *червня 2018 року*, затверджено редколегією 12 *липня 2018 року*