

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

«На правах рукопису»

УДК 004.056

«До захисту допущено»

В.о. завідувача кафедри

_____ М.В.Грайворонський

“ ____ ” _____ 2019 р.

Магістерська дисертація
на здобуття ступеня магістра

зі спеціальності: 125 Кібербезпека

на тему: Виявлення шкідливих входних параметрів у веб-застосунках

Виконав: студент 2-го курсу, групи ФБ-81мп
(шифр групи)

_____ Корженевський Олександр Сергійович

(прізвище, ім'я, по батькові)

_____ (підпис)

Науковий керівник В.о. завідувача кафедри ІБ, к.ф.-м.н., доцент,
Грайворонський М.В.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Рецензент _____

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

_____ (підпис)

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць інших
авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2019 року

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

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність (освітньо-професійна програма) – 125 Кібербезпека («Системи, технології та математичні методи кібербезпеки»)

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)

«___» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Корженевському Олександрю Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації: Виявлення шкідливих вхідних параметрів у веб-застосунках
науковий керівник дисертації: Грайворонський Микола Владленович, к.ф.-м.н., доцент,
затверджені наказом по університету від «___» _____ 2019 р. № _____
2. Термін подання студентом дисертації 10.12.2019 р.
3. Об'єкт дослідження: перевірка вхідних параметрів на предмет наявності шкідливих конструкцій
4. Вихідні дані: правила для виявлення шкідливих конструкцій та модель для їх побудови, удосконалений метод токенизації вхідних параметрів, імплементація рішення у вигляді програмного модуля.
5. Перелік завдань, які потрібно розробити: дослідити вразливості веб-застосунків та існуючі атаки, виявити недоліки механізмів захисту, запропонувати рішення для виявлення атак та вирішення проблеми обходу сучасних механізмів захисту
6. Орієнтовний перелік ілюстративного матеріалу _____

7. Орієнтовний перелік публікацій: стаття «Model of rules for malicious input parameters detection» у журналі «Theoretical and applied cybersecurity».

8. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Обрання теми та складання плану роботи	26.10.2018	
2	Огляд атак на веб-застосунки	22.02.2019	
3	Дослідження існуючих проблем захисту	12.04.2019	
4	Опис концепції рішення	14.06.2019	
5	Узгодження остаточної теми роботи	13.09.2019	
6	Розробка рішення	25.10.2019	
7	Проведення тестування	15.11.2019	
8	Подача статті до журналу	26.11.2019	
9	Оформлення та подання роботи	10.12.2019	

Студент

(підпис)

(ініціали, прізвище)

Науковий керівник дисертації

(підпис)

(ініціали, прізвище)

РЕФЕРАТ

Обсяг роботи 119 сторінок, 20 ілюстрацій, 37 таблиць, 2 додатки, 18 джерел літератури.

Метою даної дипломної роботи є вирішення проблеми захисту веб-застосунків від атак типу включення шкідливих параметрів та передових методів обходу існуючих механізмів захисту.

Об'єктом дослідження є перевірка вхідних параметрів веб-застосунків на наявність шкідливих конструкцій.

Предметом дослідження є метод токенизації, удосконалений за допомогою спеціальної моделі побудови правил для виявлення шкідливих конструкцій.

Результати роботи викладені у вигляді правил для синтаксичного та лексичного аналізу у розширеній формі Бекуса-Наура, а також програмній реалізації на мові програмування Python3 з використанням ANTLR4. Також наведені результати тестування отриманого рішення та порівняння його з існуючим.

Отримані результати можуть бути використані при розробці програмного забезпечення для захисту веб-застосунків від атак, таких як Web Application Firewall, Intrusion Prevention/Detection System та інших подібних продуктів.

ВЕБ-ЗАСТОСУНКИ, ВИЯВЛЕННЯ АТАК, СИНТАКСИЧНИЙ ТА ЛЕКСИЧНИЙ АНАЛІЗ, ФОРМА БЕКУСА-НАУРА.

ABSTRACT

The work includes 119 pages, 20 pictures, 37 tables, 2 appendixes, 18 literary references.

The purpose of this work is to solve the problem of protecting web applications from attacks such as the inclusion of malicious parameters and advanced methods of bypassing existing security mechanisms.

The object of the study is checking the input parameters of web applications for malicious constructions.

The subject of the study is the method of tokenization, improved with the help of a special model of building rules for the detection of malevolent structures.

The results of the work are presented in the form of rules for syntactic and lexical analysis in the extended Backus-Naur form, as well as software implementation in Python3 programming language using ANTLR4. The results of testing for the obtained solution and comparing it with the existing one are also given.

The obtained results can be used in the development of software for protection of web applications from attacks such as Web Application Firewall, Intrusion Prevention / Detection System, and other similar products.

WEB APPLICATIONS, ATTACKS DETECTION, SYNTATIC AND LEXICAL ANALYSIS, BACKUS-NAUR FORM.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів ..	8
Вступ.....	10
1 Огляд вразливостей та існуючих методів захисту від атак, спрямованих на дані вразливості.....	12
1.1 Найпопулярніші атаки та вразливості, які їх спричиняють	12
1.1.1 SQL-ін'єкції	12
1.1.2 Міжсайтовий скриптинг (XSS).....	15
1.1.5 XXE ін'єкції.....	19
1.1.4 HTML-ін'єкції	20
1.2 Існуючі рішення	21
1.2.1 Продуктові рішення.....	21
1.2.2 Основні підходи до виявлення шкідливого навантаження ...	23
1.3 Існуючі проблеми та методи проходження WAF.....	25
Висновки до розділу 1	26
2 Розробка рішення для виявлення шкідливих конструкцій	28
2.1 Синтаксичний та лексичний аналіз. Форма Бекуса-Наура.....	33
2.2 Вибір парсера	37
2.3 Опис правил для різних шкідливих конструкцій	39
2.3.1 Базові правила для аналізу будь-яких конструкцій.....	42
2.3.2 Нотації для виявлення SQL-ін'єкцій.....	47
2.3.3 Нотації для виявлення XSS	56
2.3.4 Нотації для виявлення HTML ін'єкцій	59
2.3.5 Нотації для XXE ін'єкцій	59

Висновки до розділу 2	60
3 Оцінка розробленого рішення	62
3.1 Створення тестового середовища	62
3.2 Проведення тестів	65
3.3 Оцінка результатів	68
Висновки до розділу 3	71
4 Розробка стартап-проекту	73
4.1 Опис ідеї проекту	73
4.2 Технологічний аудит ідеї проекту.....	75
4.3 Аналіз ринкових можливостей запуску стартап-проекту.....	76
4.4 Розроблення ринкової стратегії проекту	84
4.5 Розроблення маркетингової програми стартапу.....	87
Висновки до розділу 4	90
Висновки	92
Список джерел посилання.....	94
Додаток А.....	98
Додаток Б	114

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних.

БНФ (Форма Бекуса-Наура) – спеціальна нотація для опису синтаксису.

ДБНФ – доповнена форма Бекуса-Наура.

ПЗ – програмне забезпечення.

РБНФ – розширена форма Бекуса-Наура.

СУБД – система управління базами даних.

ALL-парсер – адаптивний LL-парсер.

ANTLR4 – генератор синтаксичних та лексичних аналізаторів.

ASCII – American Standard Code for Information Interchange – американський стандартний код для обміну інформацією.

CSS – Cascading Style Sheets – каскадна таблиця стилів.

CUTTABLE – спеціальна назва для токена, який може бути вирізаний.

DOM – Document Object Model– об’єктна модель документа.

HTML – HyperText Markup Language – гіпертекстова мова розмітки.

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту.

IDS – Intrusion detection system – система виявлення вторгнень.

IPS – Intrusion prevention system – система запобігання вторгненням.

LL-парсер – один із представників передбачаючих парсерів.

OWASP – Open Web Application Security Project.

SQL – structured query language – мова структурованих запитів, використовується для маніпуляцій з даними у СУБД.

SQL ін'єкція – включення коду SQL у якості вхідних параметрів у веб-застосунок.

Unicode – один із стандартів кодування символів.

URL – Uniform Resource Locator – уніфікований вказівник ресурсу.

WAF – Web Application Firewall – ПЗ для захисту веб-застосунків.

XML – eXtensible Markup Language – розширювана мова розмітки.

XSS – Cross-Site Scripting – атака на веб-застосунок, за якої у браузері жертви виконується сторонній код.

ВСТУП

Інтернет-технології давно відійшли від парадигми статичних сторінок. Усі сучасні веб-застосунки містять у собі елементи, які передбачають інтерактивність у діях користувача. Як мінімум у їх ролі виступають пошукові форми. Також можливі форми для відправки повідомлень, завантаження файлів тощо. Такі інтерфейси є одними із найвразливіших місць програми.

Звичайно, основні вразливості інтернет-застосунків давно відомі та досить детально досліджені. Цей факт є як позитивним, так і негативним, тому що створюються керівництва, які дозволяють програмістам розробляти безпечні програми, а адміністраторам правильно налаштовувати інформаційні системи, однак вони відомі й зловмисникам. На практиці часто правила захисту та попередження атак не дотримуються, але і їх виконання не гарантує абсолютної безпеки. Сучасний світ є своєрідним цифровим полем бою між тими, хто атакує, та тими, хто захищається. Зазвичай перші вигадують нові способи обходу існуючих засобів захисту, а другі змушені змиритися з роллю переслідувачів та відповідати на нові виклики.

Поява нових методів експлуатації веб-застосунків є причиною **актуальності** даної роботи.

За мету ставиться розробка рішення для проблеми виявлення шкідливих конструкцій, а також спроб обходу сучасних механізмів захисту.

Для досягнення поставленої мети необхідно вирішити такі завдання: дослідити популярні атаки на веб-застосунки, виявити недоліки сучасних механізмів захисту; запропонувати рішення для проблеми виявлення атак та маскуванню шкідливого навантаження; реалізувати рішення та провести тестування на звичайних та просунутих атаках.

Об'єктом дослідження є перевірка вхідних параметрів веб-застосунків на наявність шкідливих конструкцій.

Предметом дослідження є метод токенизації, удосконалений за допомогою спеціальної моделі побудови правил для виявлення шкідливих конструкцій.

До методів дослідження відносяться такі загальнонаукові методи, як аналіз (атак, механізмів захисту тощо), спостереження, формалізація (правил, моделі їх побудови), порівняння (засобів, отриманих результатів тощо) та експеримент (демонстрація роботи рішення).

Наукова новизна одержаних результатів: удосконалений сигнатурний підхід для виявлення шкідливих вхідних параметрів, а саме метод токенизації, за допомогою спеціальної моделі побудови правил, яка охоплює детектування факту використання маскування навантаження, а саме кодування та включення конструкцій, які у подальшому будуть видалені (відфільтровані).

Практичне значення одержаних результатів: запропоноване рішення може бути використаним при розробці продуктів для захисту веб-застосунків та виявлення атак типу WAF, IPS, IDS тощо.

Публікації: стаття з результатами дослідження «Model of rules for malicious input parameters detection» подана до публікації у найближчому номері журналу «Theoretical and applied cybersecurity».

1 ОГЛЯД ВРАЗЛИВОСТЕЙ ТА ІСНУЮЧИХ МЕТОДІВ ЗАХИСТУ ВІД АТАК, СПРЯМОВАНИХ НА ДАНІ ВРАЗЛИВОСТІ

1.1 Найпопулярніші атаки та вразливості, які їх спричиняють

1.1.1 SQL-ін'єкції

Переважна більшість сучасних веб-ресурсів мають так звану «трирівневу архітектуру». З назви очевидно, що виокремлюють 3 рівні функціонування веб-застосунку: рівень клієнта, рівень логіки (сервера застосунків) та рівень даних (сервера бази даних). Якщо база даних є реляційною, то вона використовує той чи інший діалект мови SQL.

SQL (англ. *structured query language* – «мова структурованих запитів») – мова, яка використовується для створення, модифікування та керування даними у реляційній базі даних, яка керується відповідною СУБД. Дана мова є стандартизованою Міжнародною організацією зі стандартизації. Поточною версією стандарту на момент написання даної роботи є SQL:2016 (ISO/IEC 9075:2016). Некоректне використання SQL у програмі дає зловмисникам додатковий вектор для атаки сервісу.

«**Ін'єкція SQL** – це атака, за якої код SQL вставляється або додається в кінець параметрів застосунку або вводу користувача, які пізніше передаються на сторону SQL серверу для парсингу та обробки. Будь-яка процедура, яка збирає запити SQL може бути потенційно вразливою, так як різноманітна природа SQL та методи доступні для конструювання його надають безліч варіантів коду» [1, с. 22]. Зазвичай SQL ін'єкції виникають внаслідок безпосередньої конкатенації уражених параметрів з командами та їх подальшим виконанням. Менш очевидний варіант даної атаки полягає у збереженні рядків зі шкідливим навантаженням у сховищі. Далі ці збережені рядки можуть бути використані у динамічних SQL запитах. Таким чином код нападника буде виконаний.

Розглянемо наступний приклад:

Запит до веб-сайту:

`http://testsite.com/finduser?id=2748` (1.5)

Припустимо, що 2748 – це ідентифікатор поточного користувача. Після запиту (1.5) на стороні сервера бази даних виконується наступний запит (1.6).

Запит до БД:

`SELECT username FROM users WHERE id = 2748;` (1.6)

Як результат користувач отримує у відповідь свій логін, зображений на екрані. Якщо модифікувати запит (1.5) наступним чином (1.7), можна отримати у відповідь username для усіх користувачів даного сервісу. Адже умова пошуку (1.8) завжди буде рівною true.

Запит до веб-сайту:

`http://testsite.com/finduser?id=2748 or 1=1` (1.7)

Запит до БД:

`SELECT username FROM users WHERE id = 2748 or 1=1;` (1.8)

SQL ін'єкції можуть бути як дуже простими, так і містити складні та заплутані конструкції. Однак зазвичай їх усіх поділяють на чотири типи:

- **SQL ін'єкції на основі помилок** – даний тип атак використовує помилки, які видає сервер при введенні некоректних параметрів, для отримання інформації про структуру бази даних;
- **SQL ін'єкції на основі UNION** – даний тип атак використовує ключове слово UNION для того, щоб додати до існуючого запиту ще один від користувача та отримати спільні результати. Виконання таких атак потребує однакової кількості полів, які повертаються для обох запитів. Приклад запиту представлений у (1.9);

- **сліпі SQL ін'єкції на основі контенту** (або булевих значень) – даний тип атак полягає у відправці запиту до бази даних, який змушує застосунок повертати різний результат в залежності від того, чи запит повертає значення TRUE (правильне) або FALSE (хибне). За допомогою такої ін'єкції зловмисник може перерахувати усі об'єкти бази даних. Проте це потребуватиме багато часу. Приклад запиту представлений у (1.12);
- **сліпі SQL ін'єкції на основі часу** – даний тип атак полягає у відправці запитів, які провокують затримку часу (зазвичай за допомогою стандартних функцій СУБД) при коректному надісланому параметрі та її відсутність при хибному. Таким чином нападник може перераховувати об'єкти у базі даних, однак це також потребуватиме значної кількості часу. Приклад запиту представлений у (1.14).

SQL ін'єкція на основі UNION:

```
SELECT id_news, header, body, author FROM news WHERE id_news =-1
UNION SELECT 1,username,password,1 FROM admin;
```

(1.9)

Даний запит окрім новин повертає логін та пароль адміністратора.

Наступний приклад. Нехай маємо наступний HTTP-запит (1.10):

```
http://testsite.com/items.php?id=2
```

(1.10)

Модифікуємо дане посилання, передавши йому наступні параметри (1.11). Запит до серверу застосунку:

```
http://testsite.com/items.php?id=2 and 1=2
```

(1.11)

SQL запит:

```
SELECT * FROM items WHERE id = 2 and 1=2;
```

(1.12)

Якщо запит (1.11) повертає відповідь, відмінну від результату для (1.10), то це означає, що виконалася команда (1.12), а, отже, можлива сліпа ін'єкція

на основі булевих значень. Виконуючи запити такого виду, зловмисник і шукає вразливі до даної атаки конструкції.

Модифікувавши запит (1.11) наступним чином (1.13) можна отримати приклад сліпої ін'єкції на основі часу (1.14):

Запит до серверу застосунку:

`http://testsite.com/items.php?id=2 and SLEEP(5)` (1.13)

SQL команда:

`SELECT * FROM items WHERE id = 2 and SLEEP(5);` (1.14)

Захистом від SQL ін'єкцій можуть слугувати наступні дії:

- фільтрація рядкових параметрів;
- фільтрація цілочисельних параметрів;
- усікання вхідних параметрів;
- використання параметризованих запитів та збережених процедур.

1.1.2 Міжсайтовий скриптинг (XSS)

Даний тип атак відрізняється від попередніх розглянутих тим, що його шкідливі дії направлені не на сервер застосунків або бази даних, а на інших клієнтів. Проте передумова для появи вразливості такого роду та сама – відсутність належної обробки вхідних параметрів.

«**XSS** – це техніка атаки, яка змушує веб-сайт відображувати шкідливий код, який потім виконується у браузері клієнта» [2]. Так як сторону клієнта зазвичай утворюють HTML, CSS та JavaScript, саме функції та оператори останньої мови найчастіше використовуються для експлуатації такого роду вразливостей.

Розрізняють такі види XSS:

- **Відображені (непостійні) XSS** – це тип міжсайтового скриптингу, який у формуванні спеціального посилання-запиту, яке потім надсилається жертві поштою або іншими засобами комунікації, таким чином, що воно містить у собі шкідливі параметри. При переході користувачем за даним посиланням шкідливе навантаження повертається назад у якості частини опису помилки (якщо запит некоректний), як результати пошуку і так далі, та виконується у браузері. Приклад такого посилання надано нижче (1.15).

`http://test.com/search.php?q=<script>DoSomeBadThings();</script>` (1.15)

- **Збережені (постійні) XSS** – цей тип атак, якщо нападник має змогу впровадити свій код у сховище веб-сайту. Тоді при кожному звертанні до відповідної області пам'яті користувач отримуватиме у відповідь від серверу шкідливе навантаження, яке буде виконуватися його браузером. Згаданий термін зазвичай використовується для експлуатації вразливостей, пов'язаних з некоректною обробкою тексту, який вводиться у коментарях на форумах, статтях блогу або повідомленнях. Проте у більш широкому сенсі, на мій погляд, його можна застосувати і до XSS через завантаження файлу. У даній атаці ім'я файлу утворюється спеціальним чином так, що при помилці спрацьовує скрипт. Помилку можна викликати, наприклад, вказавши текстовий тип файлу при завантаженні зображення, змінивши запит за допомогою програмного забезпечення як Burp Suite або OWASP Zed Attack Proxy. Приклади коду, який може зберігатися у сховищі (1.16) та шкідливого імені файлу (1.17) подано нижче.

Код, який може зберігатися у сховищі:

```
<script>document.location="http://attackerhost.example/steelcookie?" + document.cookie </script>
```

(1.16)

Приклад імені файлу, яке викликає виконання скрипту:

```
><img src=x onerror=prompt(document.domain)> (1.17)
```

- **«XSS на основі DOM** – це XSS-атака, в якій корисне навантаження нападника виконується в результаті модифікації середовища DOM у браузері жертви, що використовується оригінальним скриптом сторони клієнта, так що код клієнта виконується "несподівано". Тобто сама сторінка (відповідь HTTP) не змінюється, але код клієнта, що міститься на сторінці, виконується інакше через шкідливі модифікації, які відбулися в середовищі DOM». [3] «XSS на основі DOM є формою XSS, де весь зіпсований потік даних від джерела до стоку має місце в браузері, тобто джерелом даних є в DOM, стік також знаходиться в DOM, і потік даних ніколи не залишає браузер.» [4] Інакше кажучи, дана атака відбувається без участі сервера.

XSS на основі DOM гарно ілюструється у книзі XSS Attacks: Cross Site Scripting Exploits and Defense [2]. Припустимо, що вразливий сайт доступний за посиланням (1.18), а скрипт, який обробляє параметри має наступний вигляд (1.19).

```
http://victim/promo?product_id=100&title=Last+Chance! (1.18)
```

```
<script> var url = window.location.href;

var pos = url.indexOf("title=") + 6;

var len = url.length;

var          title_string          =          url.substring(pos,len);
document.write(unescape(title_string)); </script> (1.19)
```

Тоді, подавши на вхід шкідливе значення параметру title, отримаємо реалізацію XSS на основі DOM. Наприклад, для посилання 1.20 функція document.write допише до структури сайту назву Foo та скрипт, який буде виводити виринаюче вікно з текстом «XSS Testing».

`http://victim/promo?product_id=100&title=Foo#<SCRIPT>alert('XSS%20Testing') </SCRIPT>` (1.20)

Захист від атак такого типу полягає у наступних кроках:

- Вихідні дані кодування HTML на стороні сервера.
- Видалення / кодування вхідних даних на стороні сервера.
- Автоматична та ручна оцінка вразливостей.
- Уникнення перезапису, перенаправлення або інших критичних дій з використанням даних клієнта на стороні клієнта. Більшість цих ефектів може бути досягнуто за допомогою динамічних сторінок (на стороні сервера).
- Аналіз і зміцнення коду клієнта (Javascript). Потрібно перевірити посилання на об'єкти DOM, на які може вплинути користувач (зловмисник), включаючи (але не обмежуючись ними):
 - document.URL
 - document.URLUnencoded
 - document.location (and many of its properties)
 - document.referrer
 - window.location (and many of its properties)
- Використання дуже суворої політики IPS (Intrusion Prevencion System), в якій, наприклад, очікується, що сторінка welcome.html отримає лише один параметр з ім'ям «name», зміст якого перевіряється, а будь-яка невідповідність (включаючи надмірні параметри чи відсутність параметрів) призводить до невиконання оригінальної сторінки, так само як і за будь-якого іншого порушення (наприклад, заголовку авторизації або заголовку Referer, що містить проблемні дані), оригінальний вміст не повинен подаватися. [5]
- Використання заголовку CSP (Content Security Policy).
- Коректний менеджмент куків (встановлення параметру httponly тощо).

1.1.5 XHE ін'єкції

XML (Extensible Markup Language) – це розширювана мова розмітки. «Стандартом визначені 2 рівні правильності документа XML:

- **Правильно побудований (well-formed) документ.** Такий документ відповідає загальним правилам синтаксису XML, які можуть застосовуватися до будь-якого XML-документу. І якщо, наприклад, початковий тег не має відповідного йому кінцевого тега, то це неправильно побудований XML.
- **Дійсний (valid) документ.** Дійсний документ додатково відповідає деяким семантичним правилам. Це більш сувора перевірка коректності документа на відповідність наперед визначеним, але вже зовнішнім правилам. Ці правила описують структуру документа: допустимі назви елементів, їх послідовність, назви атрибутів, їх тип і тому подібне. Зазвичай такі правила зберігаються в окремих файлах спеціального формату - схемах.

Основні формати визначення правил валідності XML-документів – це DTD (Document Type Definition) і XML Schema» [6]. Частиною DTD є так звані **сутності** (англ. «entities»). Вони використовуються для того, щоб підключати фрагменти файлів XML, які часто використовуються за їхніми іменами. Сутності можуть бути локальні (1.22) або глобальні (1.23) (зовнішні – англ. «External»). Так, наприклад, глобальна сутність (1.23) звертається до зовнішнього ресурсу, аби отримати поточну дату.

Приклади використання сутностей:

<!ENTITY user “Oleksandr Korzhenevskiy”> (1.22)

<!ENTITY current-date SYSTEM

"http://www.getcurrenttime.com/timestamp.xml"> (1.23)

У подальшому коді до визначених у прикладах змінних можна звертатися &name та ¤t-date.

Далі створений документ перевіряється парсером (обробником). У стандарті прописані наступні слова: «Коли обробник XML розпізнає посилання на аналізовану сутність, щоб перевірити документ, обробник повинен включити текст її заміни. Якщо об'єкт є зовнішнім, і обробник не намагається перевірити документ XML, обробник може, але не потребує, включати текст заміни сутності. Якщо обробник, що не проводить валідацію, не включає текст заміни, він повинен повідомити програму, що він розпізнав, але не прочитав, об'єкт» [7]. На практиці більшість парсерів проводить валідацію, а, отже, робить заміну сутності. Це дає підставу для виникнення вразливості та її експлуатації.

Нехай сутність (1.23) тепер має інший вигляд (1.24). При запиті до парсера, обробці та поверненні результату зловмисник отримає вміст файлу /etc/passwd.

<!ENTITY current-date SYSTEM "file:///etc/passwd"> (1.24)

Захиститися від такого роду атак можна шляхом налаштування обробника на вирішення лише локальних сутностей. На рівні веб-застосунку можна фільтрувати вхідні параметри та забороняти використання сутностей.

1.1.4 HTML-ін'єкції

HTML (Hyper Text Markup Language) – мова гіпертекстової розмітки. Вона використовується на стороні клієнту та інтерпретується браузером, після чого користувачеві відображається готова форматована веб-сторінка.

«Ін'єкція HTML – це тип атаки включення, який виникає, коли користувач може керувати вхідною точкою і може вводити довільний HTML-код у вразливу веб-сторінку. Ця вразливість може мати багато наслідків, таких

як розкриття сесійного cookie користувача, яке може бути використано для виокремлення жертви, або, загалом, це може дозволити зловмиснику змінити вміст сторінки, яку бачать жертви» [8]. Можна сказати, що вразливості даного роду є передумовами для виникнення XSS та CSRF і в певній мірі їх узагальненням з точки зору механізму реалізації. Так само, як і у випадку з XSS, розрізняють збережені та відображені HTML ін'єкції.

Гарний приклад даної атаки запропонований на сайті Open Web Application Security Project [8]. Нехай застосунок містить вразливий код (1.26). Він використовує функцію `innerHTML()`, яка може бути експлуатована за допомогою запиту 1.27, в результаті чого у браузері з'явиться виринаюче віконце. Замість виклику віконця може виконатися будь-який інший код, який буде корисний для нападника.

Фрагмент коду програми:

```
var userposition=location.href.indexOf("user=");
var user=location.href.substring(userposition+5);
document.getElementById("Welcome").innerHTML=" Hello, "+user; (1.26)
```

Запит:

```
http://vulnerable.site/page.html?user=<img%20src='aaa'%20onerror=alert(1)
> (1.27)
```

Захист від атак такого типу полягає у коректній обробці вхідних параметрів, фільтрації, екрануванні спеціальних символів тощо.

1.2 Існуючі рішення

1.2.1 Продуктові рішення

Описані раніше атаки стосуються сьомого (прикладного) рівня моделі OSI. Найпоширенішим протоколом даного рівня, який зазвичай

використовується у комунікації між клієнтською та серверною частинами веб-застосунків, є HTTP. Аналізом HTTP-пакетів переважно займається Web Application Firewall (WAF).

Web Application Firewall – це програмне забезпечення або окремий пристрій з відповідним програмним забезпеченням, який подібно до звичайного фаєрволу проводить моніторинг, фільтрацію та / або блокування пакетів даних, які надходять до або надсилаються з веб-сайту або веб-застосунку. WAF є загальноновживаним засобом забезпечення безпеки, який може протистояти атакам нульового дня, зараженню шкідливим ПЗ, атакам типу SQL-ін'єкцій, XSS, переповнення буфера, викрадення сесій тощо. За своїм фізичним розміщенням, функціями та зонами відповідальності можна виокремити наступні типи фаєрволів:

- Мережеві WAF – встановлюються, як правило, у вигляді окремих фізичних пристроїв, які проводять аналіз трафіку для усіх веб-застосунків, які працюють у корпоративній або приватній мережі;
- WAF хосту – часто може бути впровадженим у програмний код самого веб-застосунку, що надає змогу детальнішого налаштування та є більш дешевим рішенням. Водночас використання цього типу фаєрволу може стати недоліком за умов недостатніх обчислювальних потужностей комп'ютера, на якому він функціонує;
- Хмарний WAF – фаєрвол, розміщений у хмарному середовищі. Основною перевагою даного типу є те, що організації, які надають відповідні послуги, мають велику базу новітніх вразливостей та здатні більш ефективно протистояти загрозам, які є актуальними у теперішній час.

Окрім WAF для виявлення шкідливих конструкцій у певній мірі можуть використовуватися інші рішення, які можуть аналізувати пакети різних рівнів, у тому числі прикладного. Серед них Системи виявлення вторгнень та

запобігання вторгненням, Next generation firewall або Unified threat management. Принцип дії даних пристроїв або програм має певні відмінності, однак підходи до виявлення атак на веб-застосунки є подібними до тих, які складають основу при розробці Web Application Firewall.

1.2.2 Основні підходи до виявлення шкідливого навантаження

Принцип роботи Web Application Firewall є загальновідомим та зазвичай зводиться до наступних кроків, зображених на ілюстрації (рис. 1.1).

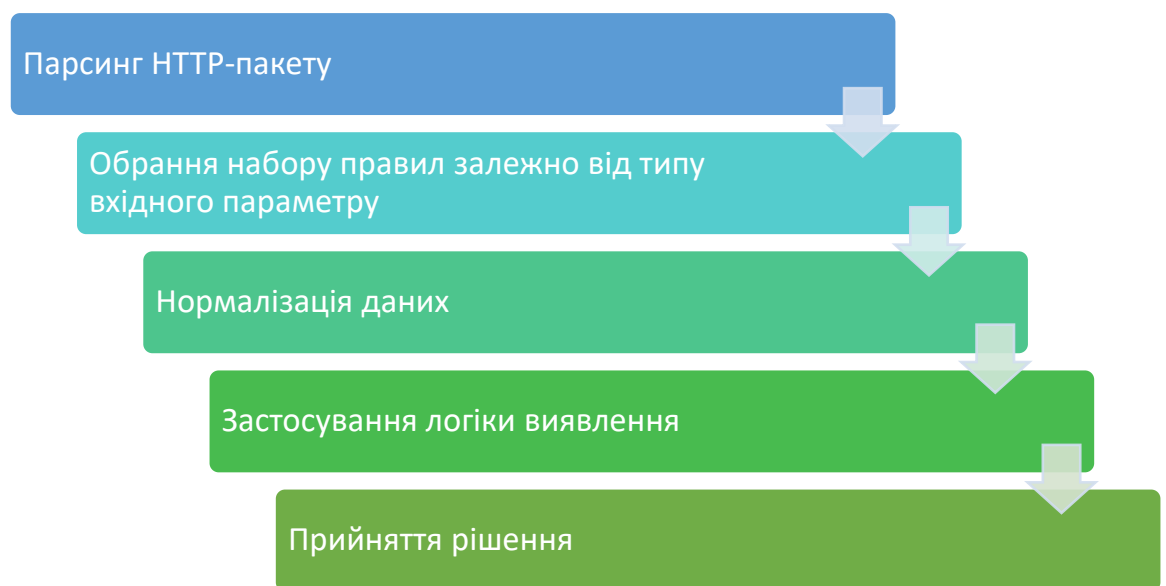


Рисунок 1.1 – Послідовність дій WAF при аналізі пакету

Перший, другий та п'ятий пункти є очевидними та однозначними, щодо них не повинно виникати додаткових запитань. Під нормалізацією даних слід розуміти приведення даних до визначеного вигляду. Прикладом може слугувати декодування, якщо дані були попередньо закодовані. Це може бути URL-декодування (наприклад, «This%20is%20my%project» трансформується у «This is my project»), Base64-декодування тощо.

Найбільш нечітким є четвертий крок роботи фаєрволу. Застосування логіки виявлення є надто загальним поняттям та може ховати у собі різноманітні методи. Володимир Іванов, науковий магістр інформаційної

безпеки та тестувальник на проникнення у веб-затосунки компанії Positive Technologies, у своєму дослідженні, результати якого були представлені на конференції Black Hat USA 2016, виділяє наступні підходи до виявлення шкідливих конструкцій:

- Використання регулярних виразів;
- Виокремлення токенів;
- Утворення рахунку;
- Виявлення аномалій;
- Аналіз репутації. [10]

Більшість існуючих фаєрволів використовують регулярні виразів. Для пошуку використовуються рядки спеціального виду – шаблони, які складаються з символів та метасимволів, які задають правила. Розробник вивчає синтаксичні конструкції, які є найпоширенішими, та створює відповідний регулярний вираз для їхнього детектування. Однак даний підхід має свої недоліки, які пов'язані у першу чергу з тим, що кожен регулярний вираз дозволяє виявляти лише конкретний запит або навіть тільки окремий його параметр. До того ж синтаксис регулярних виразів, використання еквівалентних конструкцій і використання різних представлень символів призводять до труднощів та помилок при формуванні складних правил.

Використання токенайзерів – це інший підхід, який дозволяє швидко і точно виявляти атаки класу SQL-ін'єкцій та XSS. Цей метод зводиться до пошуку сигнатур, представлених у вигляді послідовності токенів. Такі сигнатури утворюють чорний список, за яким потім проводиться аналіз кожного запиту. Недоліком такого методу є можливість впровадження у послідовність спеціальних токенів, які роблять її не розпізнаваною.

Утворення рахунку (score building) – цей метод не бере участі у виявленні шкідливих конструкцій безпосередньо, натомість доповнює інші, роблячи їх більш точними та гнучкими. Даний підхід дозволяє зменшити

помилки першого та другого роду шляхом виставлення балів для правил в залежності від критичності.

Виявлення аномалій є важливим механізмом у роботі WAF. За його допомогою можна визначити не тільки спроби впровадження некоректних вхідних параметрів, а й не типові для звичайного користувача запити та їх ланцюги. Для розв'язання даних задач переважно використовуються методи машинного навчання.

Аналіз репутації у свою чергу полягає у блокуванні трафіку, який надходить від підозрілих адрес. Оновлення баз може відбуватися вручну адміністратором або автоматично у досконаліших моделях фаєрволів.

1.3 Існуючі проблеми та методи проходження WAF

Незважаючи на існування різних методів виявлення шкідливих параметрів, описаних у попередньому підрозділі, системи, які будуються на них не є довершеними. Так існує багато публікацій, у яких дослідники діляться своїм досвідом, як обходити Web Application Firewall. Прикладом таких робіт може бути стаття, викладена на веб-сторінці OWASP, яка має назву, що українською мовою може бути інтерпретована як «SQL-ін'єкції, які обходять WAF» («SQL Injection Bypassing WAF» [11]). У ній описані різні способи видавання шкідливого навантаження за корисне такі, як додавання спеціальних символів, які видаляються фаєрволом, наприклад, коментарів, що робить некоректний запит (1.28) валідним (1.29).

`http://victim.com/news.php?id=1+union+select+1,2,3--` (1.28)

`http://victim.com/news.php?id=1+un/**/ion+se/**/lect+1,2,3--` (1.29)

Замість коментарів можуть використовуватися інші символи такі, як, наприклад, решітки '#' або нульові байти '%00', '0x00' тощо.

Також можливий варіант використання таких технік як зміна регістру літер (1.30) або включення ключових слів усередину інших, що направлене на обхід фаєрволу, налаштованого на їх видалення (1.31).

`http://victim.com/news.php?id=1+UnIoN/**/SeLeCt/**/1,2,3--` (1.30)

`http://victim.com/news.php?id=1+UNunionION+SEselectLECT+1,2,3--` (1.31)

Для XSS та іншого роду впроваджень коду HTML або JavaScript зазвичай використовується метод заміни конструкцій на еквівалентні за функціями та результатом виконання, але відмінними за синтаксисом. Таким чином можна змінити рядок (1.32) на (1.33), який у свою чергу може бути заміненим на (1.34).

`<script>alert("XSS")</script>` (1.32)

`` (1.33)

`` (1.34)

Такі техніки направлені на те, щоб скористатися недоліками механізмів захисту, як спрямованість на виявлення лише конкретного запиту для регулярних виразів або нездатність опрацювати токен-брейкери для токенайзерів. Успішність використання машинного навчання залежить від того, на якій вибірці тренувалася нейронна мережа та яка статистична модель була застосована.

Висновки до розділу 1

Незважаючи на те, що атаки типу ін'єкцій поширені та загально відомі, та існує достатня кількість методів та засобів для протидії ним, уразливості, які є причиною даних атак все ще є поширеними та актуальними проблемами сьогодення. Про це свідчить і статистика, якою володіє Open Web Application Security Project та на основі якої вони складають свій рейтинг OWASP TOP 10 [9], який містить 10 найбільш критичних загроз для веб-застосунків. Станом

на 2017 рік (поточна версія), атаки типу включення займають перше місце, XXE ін'єкції – четверте місце, а міжсайтовий скриптинг – сьоме.

Існуючі рішення спрямовані на виявлення параметрів, про шкоду яких відомо розробникам та адміністраторам. Однак зазвичай більшість методів, які є найбільш уживаними такі, як регулярні вирази, виявляються недостатньо гнучкими та нездатними детектувати подібні атаки та комбінації відомих механізмів впровадження шкідливого навантаження. Це є однією з проблем, вирішення якої дасть змогу спрямувати успішність реалізації загроз, згаданих у цьому розділі, до нуля.

2 РОЗРОБКА РІШЕННЯ ДЛЯ ВИЯВЛЕННЯ ШКІДЛИВИХ КОНСТРУКЦІЙ

Перш, ніж перейти до опису самого рішення, яке пропонується для вирішення поставленої задачі, розглянемо деякі приклади. Перш за все повернемося до запиту (1.28) та спробуємо розбити усю послідовність на окремі частини (рисунок 2.1). Те саме зробимо для виразу (1.29) (рисунок 2.2).

http://victim.com/news.php?
id=1+union+select+1,2,3--

Рисунок 2.1 – Розбиття виразу «1+union+select+1,2,3--» на окремі токени

http://victim.com/news.php?
id=1+un/**/ion+se/**/lect+
1,2,3--

Рисунок 2.2 – Розбиття виразу «1+un/**/ion+se/**/lect+1,2,3--» на окремі токени

Помаранчевим кольором на обох ілюстраціях (рис. 2.1 та 2.2) виділені числа, синім – слова, зеленим – символ «+», який заміняє пробіл, а бузковим – символи пунктуації. На рисунку 2.2 червоними прямокутниками виокремлені коментарі. Як можна побачити, за умов окремого розпізнавання коментарів, слова «union» та «select» не виявляються, хоча саме їх пошук є ціллю аналізу даної послідовності. Тоді виникає ідея: необхідно описати токени «union» та

«select» таким чином, що у будь-якому місці може бути присутня будь-яка кількість певних символів, наприклад, коментарів як у даному випадку. Тоді розпізнавання буде відбуватися, як на рисунку 2.3.

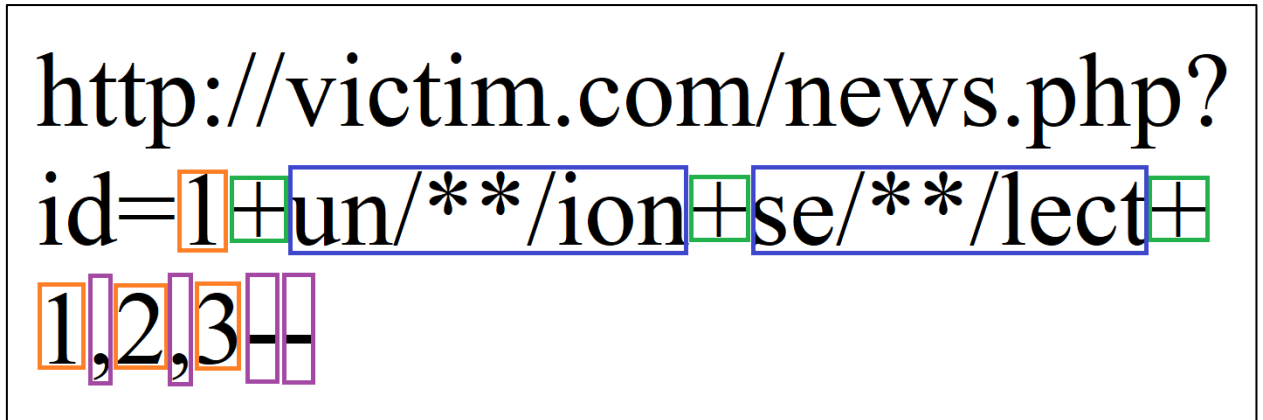


Рисунок 2.3 – Розбиття виразу «1+un/**/ion+se/**/lect+1,2,3--» на окремі токени з урахуванням можливості включення коментарів

Розглянемо також інший приклад. Цього разу розберемо вирази мови JavaScript, яка може бути включена у веб-застосунок та спричинити XSS. Вирази (2.1) та (2.2) можуть здаватися зовсім неподібними один до одного. Проте це лише на перший погляд. Насправді, вони є еквівалентними, але у випадку (2.1) кожен символ представлений його HTML-кодом, як це зображено на рисунку 2.4.

alert('XSS');

(2.1)

alert("XSS")

(2.2)



Рисунок 2.4 – Представлення «alert('XSS')» у закодованому вигляді

Очевидним варіантом вирішення даної проблеми здається декодування виразу перед аналізом. Проте використання різного кодування, а ще, що гірше, різних їх комбінацій, призводить до значних проблем при приведенні послідовності до необхідного вигляду перед дослідженням. Натомість можна змусити аналізатор сприймати певну визначену кодову комбінацію, як відповідний йому символ, без проведення декодування усього виразу. Так, наприклад, визначимо токен «А», як узагальнення певного класу елементів, які мають різні представлення (рис. 2.5).



Рисунок 2.5 – Представлення токена «А» у різному вигляді

Отже, роблячи певні проміжні підсумки, можна стверджувати, що виявлення шкідливих конструкцій має враховувати наступні твердження:

- У будь-якому місці всередині токена або між токенами може бути вставлена певна конструкція, яка буде унеможлилювати розпізнавання шуканих об'єктів, натомість може бути вирізана при подальшій фільтрації, що спричинить виконання атаки. Така конструкція може бути виокремлена у певний спеціальний токен та врахована при укладанні правил пошуку.

- Токен може представляти клас елементів, які мають різні представлення, але відповідають одному й тому ж символу. Як у випадку зображеному на рисунку 2.5. (2.3)

Спираючись на твердження (2.3) можна побудувати модель, на основі якої треба описувати правила для виявлення шкідливих конструкцій. Так як згаданий спеціальний токен позначає конструкцію, яка може бути вирізана, назвемо його CUTTABLE. Модель носитиме ієрархічний характер, і кожне правило матиме наступний вигляд (рис. 2.6):

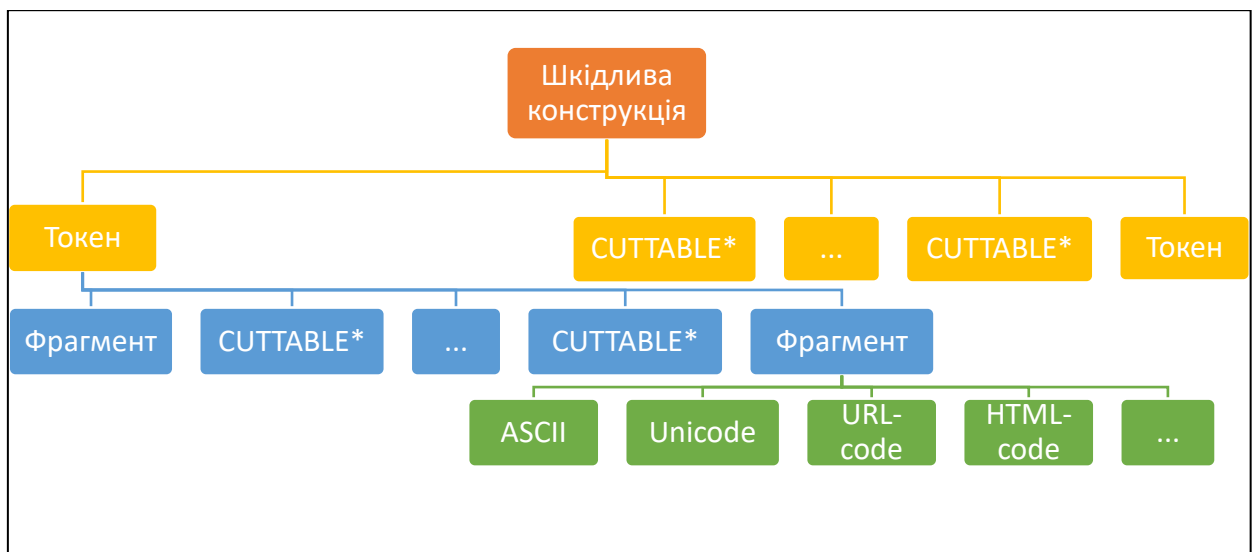


Рисунок 2.6 – Ієрархічна модель представлення шуканої шкідливої конструкції

Символ «*» означає, що цей токен може зустрічатися у даній позиції безліч разів підряд або не зустрічатися взагалі. Насправді кожен фрагмент може складатися з менших фрагментів, кількість рівнів ієрархії не обмежується, так як це показано на рисунку 2.7. По суті різниця між токеном і фрагментом полягає у тому, що перший виступає як самостійна одиниця або може входити до складу іншого, тоді як другий є виключно частиною першого. Варто зазначити, що CUTTABLE також є складеним і може представляти один із варіантів зображених на ілюстрації (рис. 2.8).

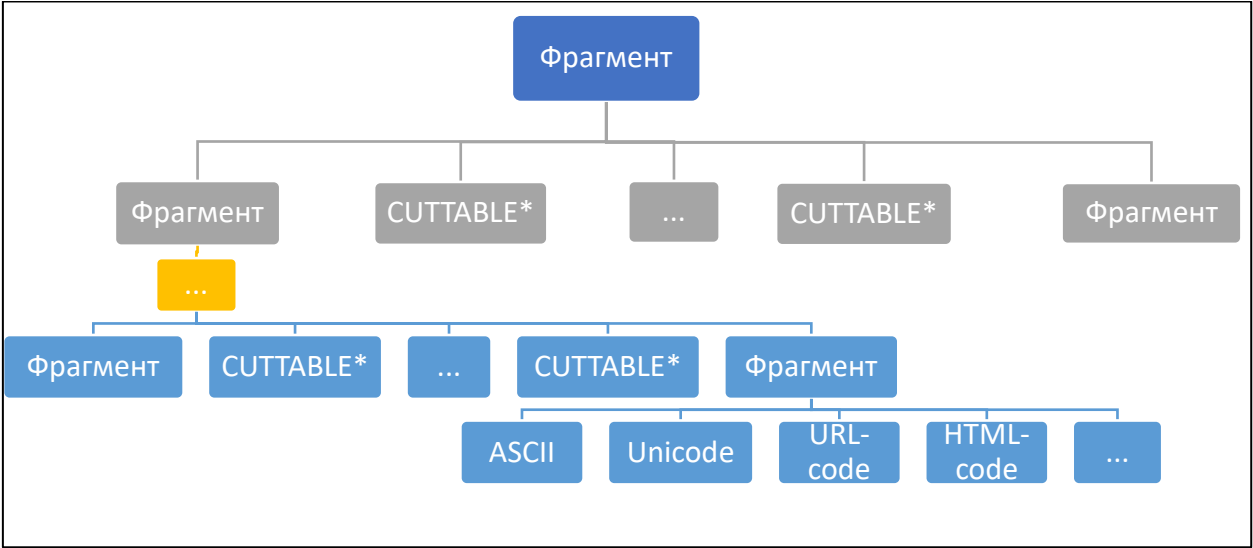


Рисунок 2.7 – Ієрархічна модель побудови фрагмента

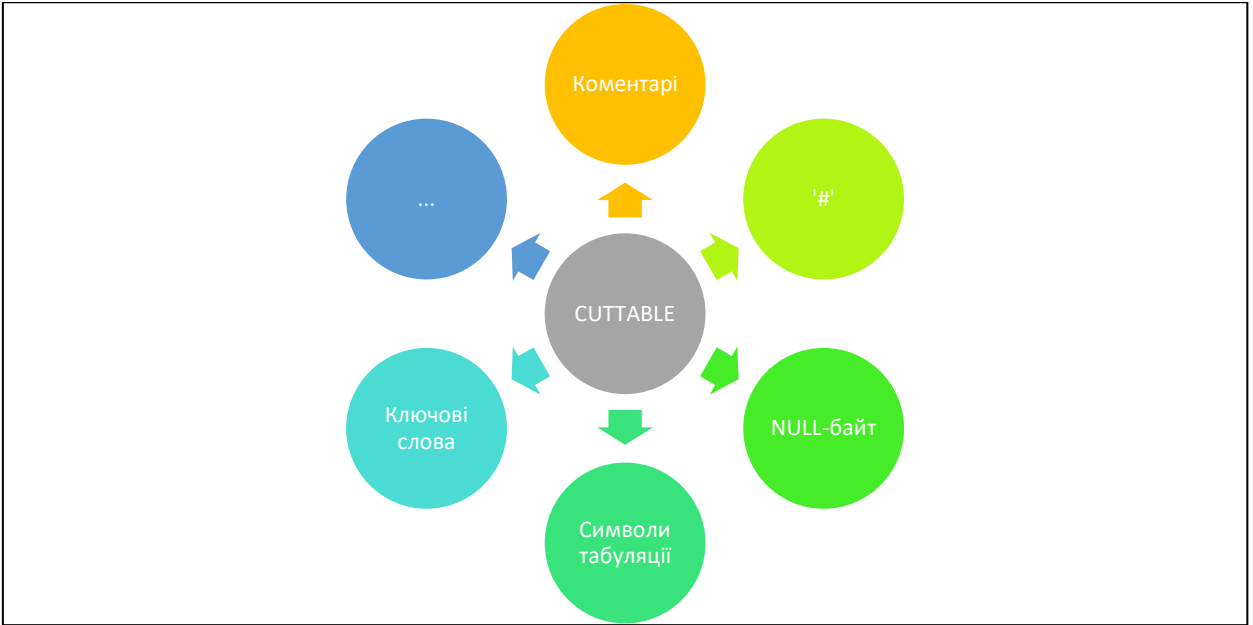


Рисунок 2.8 – Конструкції, які представляє спеціальний токен CUTTABLE

У минулому розділі були описані основні методи виявлення шкідливих вхідних параметрів, які застосовуються у сучасних WAF. Підхід, який буде розглядатися у цьому розділі, є у певній мірі поєднанням поділу послідовності на токени та використання регулярних виразів: шкідлива конструкція розглядається як послідовність токенів, натомість для опису самих токенів може бути використана регулярна мова.

Для розв'язання поставленої задачі буде застосований метод, притаманний для аналізаторів програмного коду, а саме синтаксичний та лексичний аналіз. Описана модель буде реалізована за допомогою форми Бекуса-Наура.

2.1 Синтаксичний та лексичний аналіз. Форма Бекуса-Наура

Будь-який вхідний параметр можна представити як ланцюг символів, який належить до певної мови. Перш ніж ввести поняття мови, необхідно дати визначення деяким суміжним термінам. Насамперед цікавими є поняття алфавіту, ланцюгу та мови.

Алфавітом (Σ) називається будь-яка множина символів. Поняття «символ» будемо вважати атомарним, тобто тим, на якому базуються інші дефініції, та таким, що є загальновідомим та не потребує роз'яснень. Такі поняття, як «знак» або «буква», слід вважати синонімами до слова «символ». Послідовність символів, розташованих один за одним, називається ланцюгом. Тотожними до цього терміну є «слово», «речення» або «рядок». Мовою у алфавіті Σ називається множина ланцюгів у Σ .

Першим кроком дослідження вхідних параметрів слугує лексичний аналіз (або «токенізація»). Перед початком роботи визначається алфавіт, послідовність символів якого і буде аналізуватися. «Робота лексичного аналізатора полягає у тому, щоб згрупувати визначені термінальні символи у єдині синтаксичні об'єкти, які називаються лексемами» [12]. Кожна лексема ставиться у відповідність деякій парі (тип лексеми, деякі дані). Отже, на виході лексичного аналізатора ми отримуємо ланцюг лексем. Це поняття відповідає терміну «токен», яке вводилося раніше.

У попередньому абзаці був вжитий термін «термінальні символи». Під цим поняттям треба розуміти набір усіх атомарних символів, які можуть зустрічатися у вхідному потоці. Також існують нетермінальні символи. Вони

не зустрічаються у вхідному потоці, натомість використовуються у правилах грамматики.

«Синтаксичний аналіз (або «парсинг») – це процес, у якому досліджується послідовність лексем та встановлюється, чи задовольняє вона структурним умовам, явно сформульованим у визначенні синтаксису мови» [12]. При вивченні цієї послідовності парсер працює лише з першими елементами пар – типами лексем. У якості виходу синтаксичного аналізатора виступає абстрактне синтаксичне дерево, яке зображує структуру вихідного ланцюга.

Для опису синтаксису використовуються грамматики. Граматикою називають деяку математичну систему, яка має вигляд четвірки:

$$G = (N, T, P, S) \text{ – граматика, де} \quad (2.4)$$

N – скінчена множина нетермінальних символів;

T – скінчена множина термінальних символів;

P – скінчена підмножина множини $(N \cup T)^* N (N \cup T)^* \times (N \cup T)^*$;

$(\alpha, \beta) \in P$ – правило; записується у вигляді $\alpha \rightarrow \beta$;

S – виокремлений символ з N , який називається початковим або вихідним. [12]

Правила, які задаються граматикою, є рекурсивними. Це означає, що будуються спеціальні ланцюги, які є вивідними для грамматики $G = (N, T, P, S)$:

- 1) S – вивідний ланцюг;
- 2) Якщо $\alpha\beta\gamma$ – вивідний ланцюг та $\beta \rightarrow \delta \in P$, то $\alpha\delta\gamma$ – також вивідний ланцюг.

Рекурсивність – важлива властивість граматик, яка стане у нагоді при пошуку шкідливих конструкцій у вхідних послідовностях.

Великий вклад у розвиток галузей науки, пов'язаних з мовами та граматиками, зробив американський лінгвіст Аврам Ноам Хомський (Чомскі). Він запропонував наступну класифікацію граматик:

Нехай дана граматика $G = (N, T, P, S)$. Тоді:

- 1) Якщо правила граматики не задовольняють жодні обмеження, то її називають граматикою типу 0, або граматикою без обмежень.
- 2) Якщо
 - а. Кожне правило граматики, окрім $S \rightarrow e$, має вигляд $\alpha \rightarrow \beta$, де $|\alpha| \leq |\beta|$, та
 - б. У тому випадку, коли $S \rightarrow e \in P$, символ S не зустрічається у правих частинах правил,

То граматика називають граматикою типу 1, або нескорочуючою або контекстно-залежною (КЗ-граматикою) або контекстно-чутливою (КЧ-граматикою).

- 3) Якщо кожне правило граматики має вигляд $A \rightarrow \beta$, де $A \in N$, $\beta \in (N \cup T)^*$, то її називають граматикою типу 2, або контекстно-вільною (КВ-граматика). [13]
- 4) Граматики типу 3 мають назву регулярних. Можна виокремити
 - а. праволінійну граматика, яка має наступний вигляд:

$$\alpha \rightarrow \beta$$

$$\alpha \rightarrow \omega\beta, \beta \in N, \omega \in T^*$$
 або

$$\alpha \rightarrow \omega, \omega \in T^*$$
 - б. та ліволінійну:

$$\alpha \rightarrow \beta$$

$$\alpha \rightarrow \beta\omega, \beta \in N, \omega \in T^*$$
 або

$$\alpha \rightarrow \omega, \omega \in T^*.$$

Граматика даного типу також називають скінчено-автоматними, тому що мови, які вони породжують, розпізнаються скінченими автоматами. Ці граматика становлять базу для регулярних виразів.

Для описаних вище граматик виконуються такі властивості:

- 1) Кожна граматика 3-го типу є також граматикою 2-го типу.
- 2) Кожна КВ-граматика є також граматикою типу 1.
- 3) Кожна КЗ-граматика є також граматикою нульового типу.

У інформатиці широко застосовуються контекстно-вільні граматика. Так, зокрема, за допомогою них задаються граматичні структури для багатьох мов програмування. Так як у даній дисертації розглядаються включення шкідливих параметрів, які зазвичай містять оператори SQL, JavaScript та інших мов, то саме КВ-граматика мають значення при їх дослідженні та виявленні. Слід зазначити, що правила синтаксичного та лексичного аналізу, які будуть надані пізніше, не утворюють ліволінійну чи праволінійну граматика, а, отже, ми будемо працювати саме з КВ-граматиками та мовами, що вони породжують. (Зауваження: не виключається, що усі зазначені правила можна привести до вигляду, який потребує регулярна граматика, натомість відповідні перетворення не проводилися, тому твердження про використання саме регулярних граматик є необґрунтованими.)

Найпоширенішим способом задання правил для них є форма Бекуса-Наура (БНФ).

Форма Бекуса-Наура – це компактна формальна метамова для КВ-граматик, яка часто використовується для опису синтаксису мов програмування, формату документів (наприклад, XML-документів) та іншого. У даній нотації одні синтаксичні категорії послідовно визначаються через інші. БНФ була винайдена Джоном Бекусом, який займався проектуванням мов програмування у компанії IBM. Він розробив її для опису синтаксису ALGOL 58, а вперше вона була опублікована у його доповіді щодо ALGOL 60.

Трохи пізніше інший розробник даної мови програмування Пітер Наур вніс певні правки до нотації, спростивши її.

Окрім традиційної БНФ також існують розширена (РБНФ) та доповнена (ДБНФ) нотації. Їхній синтаксис містить деякі відмінності та робить традиційну форму більш потужною та компактною у написанні. До того ж різні парсери можуть використовувати власні модифіковані версії правил. Натомість основні ідеї, які закладалися Бекусом та Науром, залишаються незмінними. Поданий нижче приклад описує правила для парсингу цілих невід’ємних чисел у десятковому представленні та є ілюстрацією синтаксису БНФ-конструкцій (2.2).

$$\begin{aligned} \langle \text{digit} \rangle &::= '0' \mid '1' \mid '2' \mid '3' \mid '4' \mid '5' \mid '6' \mid '7' \mid '8' \mid '9' ; \\ \langle \text{number} \rangle &::= \langle \text{digit} \rangle \langle \text{number} \rangle \mid \langle \text{digit} \rangle ; \end{aligned} \quad (2.5)$$

2.2 Вибір парсера

Контекстно-вільні граматики, а також відповідні аналізатори можна певним чином класифікувати. Наприклад, за критерій для поділу можна взяти тип алгоритму парсера.

Типи алгоритмів:

- Нисхідний парсер – продукції граматики розкриваються, починаючи з початкового символу до отримання потрібної послідовності токенів. Ці аналізатори створюють ланцюги лівостороннього виводу. Древа виводу за їхнього використання будуються згори донизу. Зазвичай при роботі таких парсерів використовується метод рекурсивного спуску. Із самої назви є очевидним, що в його основі лежать рекурсивні виклики функцій, кожна з яких ставиться у відповідність до певного правила граматики або БНФ. Існує 2 варіанти реалізації:

- Передбачаючий парсер – таким, наприклад, є LL-парсер. Він здатний коректно передбачати один з варіантів розкриття кожного нетерміналу в залежності від поточного токена або їх послідовності. Часова складність роботи LL(k) алгоритму є лінійною.
- Парсер з поверненням – у даному випадку перебираються усі альтернативи, доки чергова спроба не буде успішною. Час роботи цього алгоритму є експоненціальним. Його недоліком є можливість виникнення лівої рекурсії.
- Висхідний парсер – продукції відновлюються у зворотньому порядку: обробка починається з токенів та закінчується стартовим символом. Відповідно аналізатори даного типу будують ланцюги правостороннього виводу та дерево виводу знизу угору. Зазвичай, коли говорять про парсери такого типу, мають на увазі LR(k)- або GLR(k)-аналізатори.

«k» у даному контексті означає кількість наступних символів після поточного, які треба подивитися, щоб обрати коректне правило для аналізу. Тобто, якщо вказаний LL(1) або LR(1), то для правильного парсингу треба знати 2 символи: поточний та наступний.

Для виконання поставленої задачі був обраний генератор парсерів та лексерів ANTLR4. Цей парсер типу ALL(*), що є різновидом LL-аналізаторів. «*» у дужках означає, що даний аналізатор не обмежує значення k наступних токенів, які треба переглянути, а також може розпізнавати, чи належать ці токени певній регулярній мові. «A» означає адаптивність. Як зазначають автори статті, у якій презентуються результати дослідження якості функціонування ANTLR4: «ALL(*) стратегія парсингу поєднує простоту, ефективність та передбачуваність звичайних нисхідних LL(k) аналізаторів з потужністю механізмів, подібних до GLR, у прийнятті рішень щодо розпізнавання». [14]

Нижче розглянуті переваги ANTLR4, які стали причинами вибору саме цього продукту, у якості генератора синтаксичних та лексичних аналізаторів:

- а) Швидкодія: у вищезгаданій статті [14] стверджується, що теоретично виведена часова складність сягає $O(n^4)$, що не є добре та є гіршим показником, аніж для GLR, у яких дана оцінка дорівнює $O(n^3)$. Натомість результати послідовних випробувань з граматиками, які зазвичай використовуються на практиці, вказують на лінійний час виконання операцій щодо розпізнавання послідовностей токенів та віднесення їх до того чи іншого правила.
- б) Підтримка мов програмування: у порівнянні зі своїм попередником ANTLR4 підтримує меншу кількість мов, для яких генерується код синтаксичних та лексичних аналізаторів. З іншого боку у переліку присутні C++ та Go, які зазвичай використовуються у різного роду мережевих пристроях та серверах, тому можуть бути корисними для створення мережевих або хмарних Web Application Firewall. Також ANTLR4 підтримує такі мови, як C#, Java, JavaScript та Python, що є найпоширенішими у індустрії створення веб-застосунків. У цьому випадку можуть бути згенеровані парсери та лексери, які будуть корисними для WAF хосту. Доповнює перелік згаданих мов Swift, популярний серед розробників програмного забезпечення для продуктів компанії Apple.

Таким чином, обравши даний генератор аналізаторів, ми отримуємо швидке рішення для великої частини сучасного світу інтернет.

2.3 Опис правил для різних шкідливих конструкцій

Синтаксис, який запропонували розробники ANTLR4, має певні особливості та є варіацією розширеної форми Бекуса-Наура. Так, наприклад, замість символу « $::=$ » при описі правил використовується « $:$ ». Символи, які

визначаються, не беруться у кутові дужки, а, отже, будь-яке слово записане у нотації, буде розпізнане генератором аналізаторів, як нетермінальний символ, та буде опрацьовуватися подібно до змінних у мовах програмування. Якщо розробник правил хоче вказати, що дана конструкція є кінцевим рядком, який складається з символів заданого алфавіту, його треба взяти у лапки, подібно до значень змінних типу `string` при написанні коду програм.

Необхідно також звернути увагу на те, що ANTLR4 є генератором як синтаксичних, так і лексичних аналізаторів. Нотації для обох містять певні відмінності. Візуально насамперед це відображається у тому, що назви правил для лексерів починається з великої літери, натомість правила для парсерів пишуть з малої. Так, наприклад, нотацію (2.5) можна переписати відповідно до вимог ANTLR4 (2.6). У даному випадку `DIGIT` буде лексичним правилом, а `number` – синтаксичним.

```
DIGIT      :      [0-9]      ;
number     :      DIGIT+    ;                                (2.6)
```

Очевидно, що правила мають вигляд, відмінний від того, який був поданий раніше. Квадратні дужки вказують на діапазон допустимих значень. У даному випадку це цифри від нуля до дев'яти. Знак «+» після токена `DIGIT` у правилі `number` означає те, що вказаний токен повинен зустрітися принаймні один раз. Окрім нього можливе використання «?» та «*». «?» свідчить про те, що зазначений символ зустрінеться від 0 до 1 разу, а «*» – що він зустрінеться від 0 до безлічі разів.

Правила (2.6) можна було записати інакшим чином. У наведеному вище прикладі токенами будуть усі десяткові цифри, з яких складатиметься число. Якщо корисно розпізнати саме число, натомість окремі цифри не матимуть значення для подальшого дослідження, тоді правило `number` слід змінити на лексичне, а для `DIGIT` використати ключове слово `fragment`, яке означає, що

даний токен буде використаний виключно, як частина іншого. Модифікуємо попередній приклад (2.6) з урахуванням даних змін:

fragment DIGIT : [0-9] ;
NUMBER : DIGIT+ ; (2.7)

Насправді, вираз (2.4), можна спростити записавши його у поданому нижче вигляді (2.5).

NUMBER : [0-9]+ ; (2.8)

Правила, як синтаксичні, так і лексичні надають можливість описати альтернативи. Таким чином, змінимо правила (2.7) та (2.8) і додамо можливість подання чисел у шістнадцятковій системі. У такому випадку нотація матиме наступний вигляд:

fragment HEX_DIGIT : ('0'..'9' | 'a'..'f' | 'A'..'F') ;
NUMBER : [0-9]+ | '0' 'x' HEX_DIGIT+ ; (2.9)

У даному фрагменті зустрічаються нові спеціальні символи. Використання дужок у даному випадку не є обов'язковим, що можна побачити із синтаксису для NUMBER. Натомість блоки у дужках називають підправилами. Запис ('0'..'9' | 'a'..'f' | 'A'..'F') свідчить про те, що певна з альтернатив повинна зустрітися рівно один раз. Також слід зауважити використання «..». Це є одним із способів задання інтервалів подібно до квадратних дужок. Проте таким чином можна задати лише ряд послідовних символів.

Отже, вище описані базові моменти, на які треба було звернути увагу перш, ніж перейти безпосередньо до опису правил для виявлення шкідливих вхідних параметрів. Інші деталі будуть пояснюватися за потреби при створенні нотації.

2.3.1 Базові правила для аналізу будь-яких конструкцій

Шкідливі конструкції, які дослідник повинен відшукати, зазвичай містять ключові слова певної мови програмування, розмітки або запитів. Ці слова англійського походження, а, отже, складаються із літер латинського алфавіту. Тому перш за все треба описати правила для парсингу букв, як атомарних токенів. Так як окремі літери не є предметом дослідження, слід вжити ключове слово `fragment`, передбачене синтаксисом ANTLR4. У таблиці 2.1 подане правило, яке описує літеру А. Правила для інших букв мають ідентичний вигляд.

Таблиця 2.1 – Правило опису фрагмента (токена) А

Токен	Визначення	Тип представлення
fragment A :	[Aa]	Звичайний символ ASCII або Unicode
	'%'[46]'1'	URL-кодування
	'\\1'[04]'1'	ASCII кодування (8)
	'\\x'[46]'1'	ASCII кодування (16)
	'\\u00'[46]'1'	Unicode кодування
	'&#' '0'? '0'? '0'? '0'? '0'? '0'? ('65' '97') ';'?	HTML кодування (10)
	'&#'[Xx] '0'? '0'? '0'? '0'? '0'? '0'? [46]'1' ';'? ;	HTML кодування (16)

Число у дужках означає систему числення. Так, наприклад, HTML кодування (10) означає, що код поданий у десятковій системі. У подальшому дані правила можуть бути доповненні іншими форматами подання.

Було прийняте рішення ігнорувати QUOTE при розпізнаванні рядків. Причиною цьому стала неоднозначність сприйняття конструкцій та невірне трактування. Так замість того аби розпізнати ланцюг як це зображено на рисунку 2.9, програма може розпізнати його як на наступній ілюстрації (рис. 2.10).

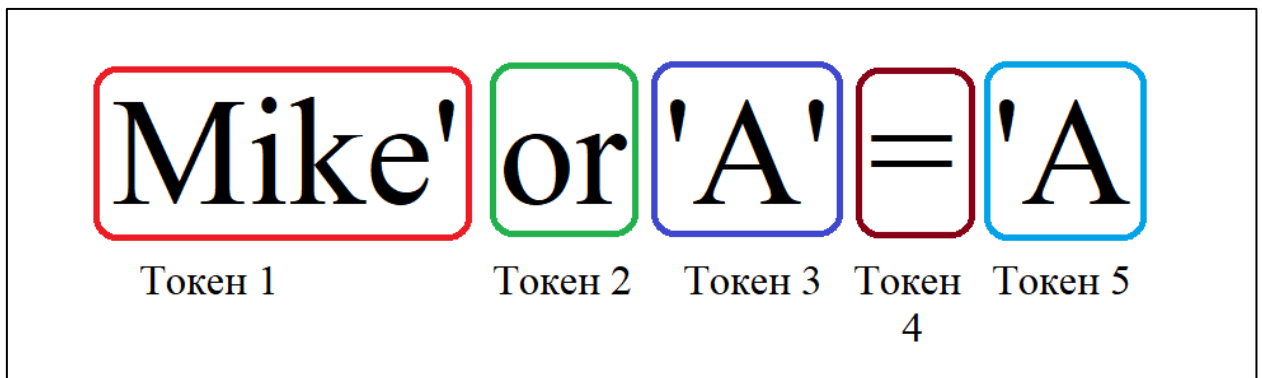


Рисунок 2.9 – Один із варіантів розпізнавання послідовності з лапками

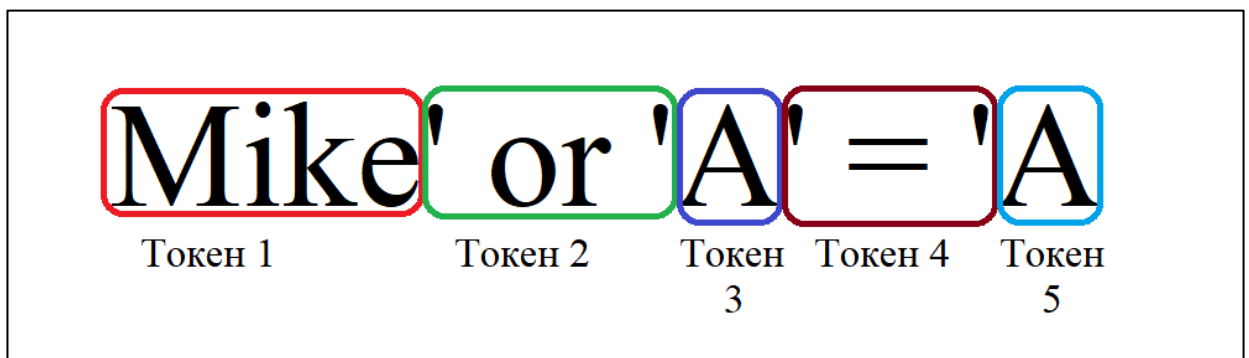


Рисунок 2.10 – Інший варіант розпізнавання послідовності з лапками

Далі опишемо спеціальний токен CUTTABLE (таблиця 2.3). Як можна помітити у описі фігурують інші токени, які ще не були описані. Річ у тім, що ANTLR надає змогу писати правила у будь-якій послідовності. Їх частини у будь-якому разі будуть розпізнані. Натомість чим раніше описаний токен, тим вищий у нього пріоритет. Таким чином зазвичай найбільш загальні лексеми рекомендується описувати після часткових. У даній роботі ми будемо дотримуватися того порядку, який був обраний для коректного аналізу вхідних параметрів.

Таблиця 2.3 – Правило розпізнавання спеціального токена CUTTABLE

Токен	Визначення
CUTTABLE :	COMMENT HASHTAG NULL_BYTE HOR_TAB VER_TAB LINE_FEED CAR_RET QUOTE ;

Правило, описане вище, може бути доповнене іншими лексемами за необхідності. На момент написання дисертації було виявлено, що саме згадані конструкції найчастіше використовуються для приховання атаки. Серед них:

- COMMENT – представляє коментарі;
- HASHTAG – символ «#»;
- NULL_BYTE – нульовий байт;
- HOR_TAB – горизонтальна табуляція;
- VER_TAB – вертикальна табуляція;
- LINE_FEED – зміна рядка;
- CAR_RET – повернення каретки;
- QUOTE – символ лапок, який ми вже описували.

Токен QUOTE вже згадувався вище, інші, крім COMMENT, описуються подібно до того, як це зображено в таблицях 2.1 та 2.2, але із заміною кодів на відповідні. COMMENT описується як вибір із кількох варіантів. Правила опису даної лексеми можна побачити нижче (2.10). По суті, ці правила є представленням коментарів для різних мов програмування та SQL.

COMMENT : (SLASH '*' .*? '*' SLASH | (SLASH SLASH | HASHTAG HASHTAG | HYPHEN HYPHEN) .*? (LINE_FEED|'\n'))-> skip ; (2.10)

Також були описані правила для токенів, що представляють крапку, кому, круглі та квадратні дужки, двокрапки та крапки з комою, а також символів роздільників таких, як пробіли, символи табуляції, нових рядків та «+» (токен SEPARATOR).

Наступним кроком необхідно також записати правила для виявлення токенів, які відповідають числам. Для запису чисел найчастіше використовують десяткову та шістнадцяткову систему представлення. Десяткові числа можуть бути як цілими, так і дробовими, як додатними, так і від'ємними. Враховуючи дані зауваження пропонуються наступні правила аналізу (2.11):

fragment

HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

NUMBER : '-'[1-9][0-9]*

| '-'[1-9][0-9]* POINT [0-9]*

| [0-9]+

| [0-9]+ POINT [0-9]*

| '0b' [01]+

| '0o' [0-7]+

| '0x' HEX_DIGIT+ ; (2.11)

Описані у цьому пункті правила складають «фундамент» для подальших більш специфічних нотацій, які будуть розглядатися далі у даній роботі.

2.3.2 Нотації для виявлення SQL-ін'єкцій

Як зазначалося у попередньому розділі, розрізняють декілька типів SQL-ін'єкцій, а саме:

- SQL ін'єкції на основі помилок;
- SQL ін'єкції на основі UNION;
- сліпі SQL ін'єкції на основі контенту (або булевих значень);
- сліпі SQL ін'єкції на основі часу.

Розпочати опис правил пропонується з SQL ін'єкцій на основі UNION через чітко визначений синтаксис таких конструкцій та їх невелику кількість. Розрізняють такі варіації запитів:

- UNION SELECT ... ;
- UNION (SELECT ... ;
- UNION ALL SELECT ... ;
- UNION ALL (SELECT ... ;
- UNION DISTINCT SELECT ... ;
- UNION DISTINCT (SELECT ...

Окрім пробілів між цими словами можуть стояти символи табуляції, переносу рядку або «+», тобто токен SEPARATOR. Також може бути присутня певна кількість токенів CUTTABLE. Слід також зауважити, що обидва згадані токени можуть чергуватися між собою у будь-якій послідовності. Окрім них також може бути присутній токен SQLKEY, який представляє певні ключові слова мови SQL, які теж можуть бути вирізані. У певних місцях також може бути присутнім символи круглої дужки. За таких умов правила синтаксичного аналізатора матимуть наступний вигляд, поданий у таблиці 2.4.

До того ж символи, які можуть бути вирізані, можуть міститися й усередині токенів UNION, SELECT, ALL та DISTINCT. Лексичні правила для

даних токенів будуть визначені послідовно через фрагменти (таблиця 2.5). Надалі цей підхід буде застосовуватися для усіх конструкцій.

Таблиця 2.4 – Правила парсингу конструкції UNION SELECT

Назва правила	Опис
union_select	UNION (SEPARATOR CUTTABLE SQLKEY L_PAREN)+ SELECT UNION (SEPARATOR CUTTABLE SQLKEY)+ ALL (SEPARATOR CUTTABLE SQLKEY L_PAREN)+ SELECT UNION (SEPARATOR CUTTABLE SQLKEY)+ DISTINCT (SEPARATOR CUTTABLE SQLKEY L_PAREN)+ SELECT;

Таблиця 2.5 – Правила лексичного аналізу для UNION

Назва токена	Опис
fragment ON	O (CUTTABLE SQLKEY)* N ;
fragment ION	I (CUTTABLE SQLKEY)* ON ;
fragment NION	N (CUTTABLE SQLKEY)* ION ;
UNION	U (CUTTABLE SQLKEY)* NION REVERSE (CUTTABLE SQLKEY SEPARATOR)* L_PAREN (CUTTABLE SQLKEY SEPARATOR)* NOINU (CUTTABLE SQLKEY SEPARATOR)* R_PAREN;

Як можна побачити, у таблиці 2.5 використовується нотація для конструкції, яка містить функцію REVERSE, яка обертає поданий рядок. Ця функція часто використовується зловмисниками, щоб обійти засоби захисту,

які зазвичай використовують WAF. Таким чином «union» замінюється на REVERSE(«noinu»), що знайшло відображення у правилах, запропонованих вище. Нотації для tokenів REVERSE та NOINU не надані, натомість вони мають подібну структуру до UNION. Токени SELECT, DISTINCT та ALL описуються подібним чином. Рекурсивний опис дає змогу покрити усі можливі випадки розташування CUTTABLE та SQLKEY у середині згаданих конструкцій у простий та легкий для сприйняття людиною спосіб.

Наступним видом, для якого будуть описані правила, є сліпі ін'єкції на основі булевих значень. Таким чином у будь-якому разі параметри, які будуть подаватися на вхід програмі, міститимуть логічні оператори «AND» або «OR». На ілюстрації (рис. 2.11) зображено основний шаблон, який необхідно виявити.

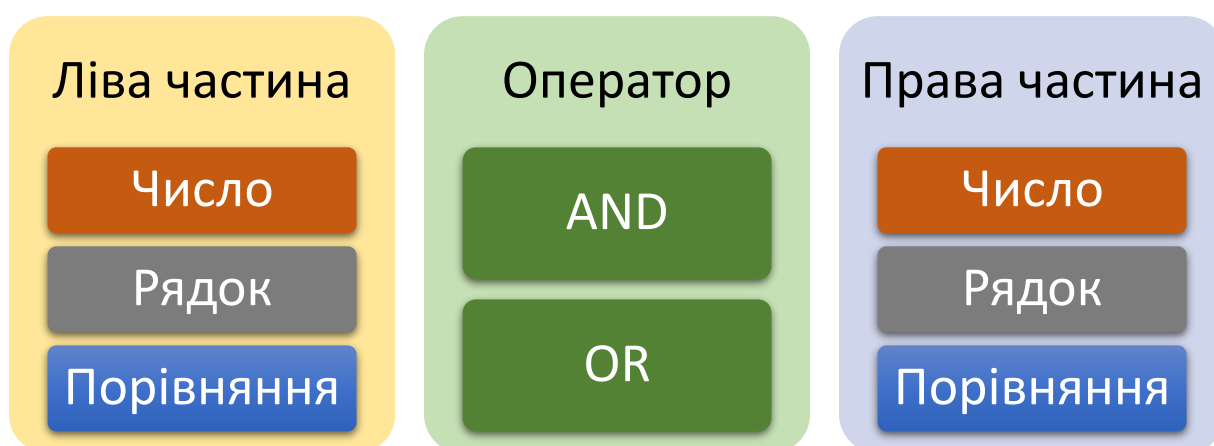


Рисунок 2.11 – Структура булевого виразу

Слід зауважити, що рядок може у лівій частині може бути відкритий зліва, а у правій частині – справа відповідно. Порівняння мають вигляд, приведений на рисунку 2.3. У якості вбудованих функцій, які можуть бути використанні при порівнянні, зазвичай використовують `ascii()`, `lower()`, `upper()`, `substring()`, `user()` та інші, які повертають значення простих (вбудованих) типів. Роль вкладеного запиту виконує пошуковий вираз, тобто той, який містить ключове слово SELECT та робить певну вибірку.

Ліва частина	Оператор	Права частина
<ul style="list-style-type: none"> • Число • Рядок • Вбудована функція • Вкладений запит 	<ul style="list-style-type: none"> • = • != • <> • > • < • >= • <= 	<ul style="list-style-type: none"> • Число • Рядок • Вбудована функція • Вкладений запит

Рисунок 2.12 – Структура виразу порівняння

Особливості описаної структури логічних виразів були враховані при створенні правил для їх виявлення (таблиця 2.6).

Таблиця 2.6 – Правила лексичного та синтаксичного аналізу для сліпих SQL ін'єкцій на основі контенту

Назва правила	Опис
boolean_sql_i	(OR AND) (SEPARATOR CUTTABLE SQLKEY)+ pred_part;
pred_part	comp NUMBER WORD ;
comp	comp_part (SEPARATOR CUTTABLE SQLKEY)* COMP_SIGN (SEPARATOR CUTTABLE SQLKEY)* comp_part ;
comp_part	NUMBER sql_func L_PAREN (SEPARATOR CUTTABLE SQLKEY)* select_statement .*? R_PAREN WORD ;
sql_func	SQL_FUNC (SEPARATOR CUTTABLE SQLKEY)* L_PAREN .*? R_PAREN ;
SQL_FUNC	ASCII CONCAT SUBSTRING VERSION HEX UNHEX MIN MAX AVG SUM ;

При написання нотацій для ANTLR4 прийнято спочатку писати правила для парсерів, а потім – для лексерів. Перелік функцій у SQL_FUNC може бути розширений. Усі токени, які входять до його переліку визначаються подібно до того як це описано в таблиці 2.7.

Таблиця 2.7 – Опис токена SUM

Назва токена	Опис
fragment UM	U (CUTTABLE SQLKEY)* M ;
SUM	S (CUTTABLE SQLKEY)* UM ;

Наступним типом, який буде розглядатися, будуть сліпі SQL ін'єкції на основі часу. Дані ін'єкції направлені на виклик певної функції, яка спричиняє затримку, за умов виконання або навпаки невиконання певної умови. Таким чином детектування таких атак зводиться до виявлення використання таких функцій. Зазвичай використовують:

- SLEEP (MySQL);
- BENCHMARK (MySQL);
- WAITFOR DELAY (MS SQL);
- PG_SLEEP (PostgreSQL);
- GENERATE_SERIES (PostgreSQL).

SLEEP, PG_SLEEP та WAITFOR DELAY спричиняють затримку безпосереднім способом, адже це і є їхнім призначенням. BENCHMARK у свою чергу використовується для тестування продуктивності. Дана функція приймає два аргументи: процедуру та число, яке вказує на те, скільки її треба виконати. Таким чином, якщо процедура вимагає значного часу для виконання, а число повторів також є великим, виникає затримка, яку можна зафіксувати. Функція GENERATE_SERIES генерує певний набір елементів. Якщо розмір такого набору є достатньо великий, це може спричинити затримку.

Отже, для згаданих функцій створені наступні правила (таблиця 2.8).

Таблиця 2.8 – Правила лексичного та синтаксичного аналізу для сліпих SQL ін'єкцій на основі затримки часу

Назва правила	Опис
timebased_sqli	waitfor_delay benchmark_sqli sleep_sqli pg_sleep_sqli gen_series_sqli ;
waitfor_delay	WAITFOR (SEPARATOR CUTTABLE SQLKEY)+ DELAY (SEPARATOR CUTTABLE SQLKEY)+ TIMESTRING ;
benchmark_sqli	BENCHMARK (SEPARATOR CUTTABLE SQLKEY)* L_PAREN (SEPARATOR CUTTABLE SQLKEY)* NUMBER (SEPARATOR CUTTABLE SQLKEY)* COMMA .+? R_PAREN ;
sleep_sqli	SLEEP (SEPARATOR CUTTABLE SQLKEY)* L_PAREN (SEPARATOR CUTTABLE SQLKEY)* NUMBER (SEPARATOR CUTTABLE SQLKEY)* R_PAREN ;
gen_series_sqli	GENERATE_SERIES (SEPARATOR CUTTABLE SQLKEY)* L_PAREN (SEPARATOR CUTTABLE SQLKEY)* (NUMBER WORD) (SEPARATOR CUTTABLE SQLKEY)* COMMA (SEPARATOR CUTTABLE SQLKEY)* (NUMBER WORD) (COMMA (SEPARATOR CUTTABLE SQLKEY)* (NUMBER WORD)))? (SEPARATOR CUTTABLE SQLKEY)* R_PAREN ;

Кінець таблиці 2.8

Назва правила	Опис
pg_sleep_sqli	PG_SLEEP (SEPARATOR CUTTABLE SQLKEY)* L_PAREN (SEPARATOR CUTTABLE SQLKEY)* NUMBER (SEPARATOR CUTTABLE SQLKEY)* R_PAREN ;
TIMESTRING	('0'?[0-9]]'1'[0-9]]'2'[0-3]) COLON ('0'?[0-9]][1-5][0-9]) COLON ('0'?[0-9]][1-5][0-9]) ('0'?[0-9]]'1'[0-9]]'2'[0-3]) COLON ('0'?[0-9]][1-5][0-9]) COLON ('0'?[0-9]][1-5][0-9]) ;

Токени BENCHMARK, WAITFOR, SLEEP, PG_SLEEP, GENERATE_SERIES описуються подібно до SUM у таблиці 2.7.

SQL ін'єкції на основі помилок не містять яскраво виражених конструкцій, характерних лише для даного типу. Вони схожі на інші види, описані раніше, за синтаксисом, а відрізняються тим, що у результаті їхнього виконання клієнт отримає від серверу повідомлення з помилкою. Тому окремі правила для даного типу створюватися не будуть. Натомість замість нього пропонується розглянути послідовні запити («складені запити» від англ. «stacked queries»), які деякі дослідники відносять до ін'єкцій на основі UNION. Хоча вони не містять самого ключового слова, принцип роботи є подібним і полягає у виконанні окрім запиту, який передбачений логікою програми, ще одного (можливо, більше), впровадженого користувачем. Приклад зображений на рисунку 2.13. Спочатку закривається попередній запит. Цей фрагмент виділений зеленим кольором. У даному прикладі це число, але можливе використання також рядку. Закриття запиту відбувається за рахунок використання символу «;». Після цього йде інший запит. Червоним кольором виділені ключові слова мови SQL, які є однаковими для усіх запитів (у даному

випадку на видалення таблиці, можливі інші оператори) та які можуть бути виявлені при кожній спробі проведення атаки такого виду.

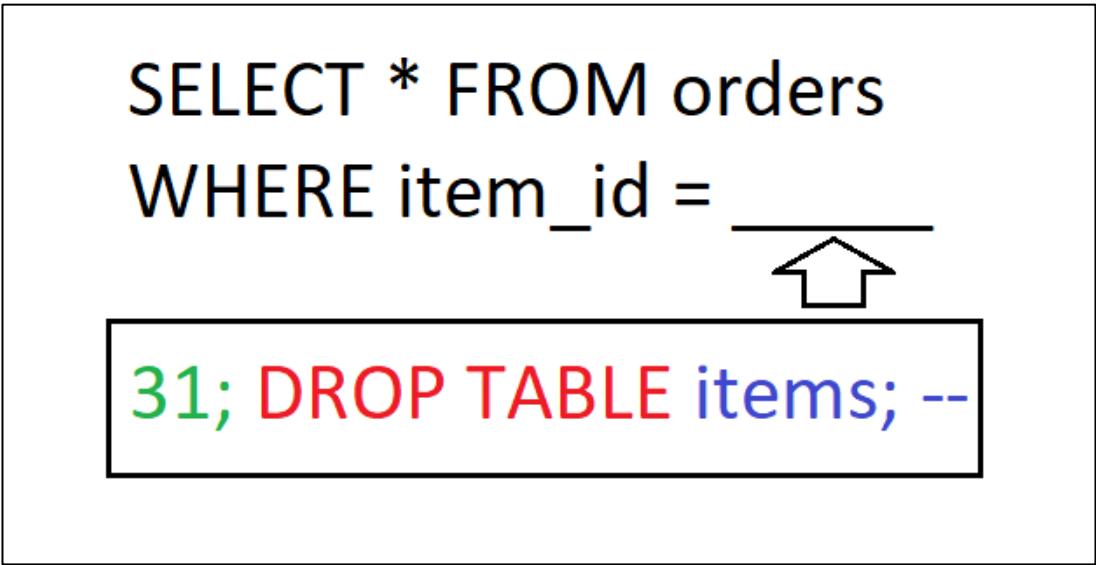


Рисунок 2.13 – Впровадження послідовного (складеного) запиту

Зазвичай використовують наступні конструкції:

- `SELECT ...;`
- `CREATE TABLE ...;`
- `ALTER TABLE ...;`
- `INSERT INTO ...;`
- `DROP TABLE ...;`
- `DELETE FROM ...;`
- `UPDATE table_name SET ...`

Таблиця 2.9 – Правила для складених запитів

Назва правила	Опис
stacked_query	<code>SEMICOLON (SEPARATOR CUTTABLE SQLKEY)* sql_statement L_PAREN (SEPARATOR CUTTABLE SQLKEY)* sql_statement .*? R_PAREN;</code>

Назва правила	Опис
sql_statement	select_statement create_table alter_table drop_table insert_into delete_from update_statement ;
select_statement	SELECT (SEPARATOR CUTTABLE SQLKEY)+ (select_part) ((SEPARATOR CUTTABLE SQLKEY)* COMMA (SEPARATOR CUTTABLE SQLKEY)* select_part)* ;
select_part	NUMBER sql_func WORD ;
create_table	CREATE (SEPARATOR CUTTABLE SQLKEY)+ TABLE
alter_table	ALTER (SEPARATOR CUTTABLE SQLKEY)+ TABLE;
drop_table	DROP (SEPARATOR CUTTABLE SQLKEY)+ TABLE ;
insert_into	INSERT (SEPARATOR CUTTABLE SQLKEY)+ INTO ;
delete_from	DELETE (SEPARATOR CUTTABLE SQLKEY)+ FROM;
update_statement	UPDATE (SEPARATOR CUTTABLE SQLKEY)+ WORD (SEPARATOR CUTTABLE SQLKEY)+ SET ;

Токени CREATE, TABLE, ALTER, DROP, INSERT, INTO, DELETE, FROM та SET визначені подібно до SUM у таблиці 2.7.

Усі описані вище правила зрештою є частинами основного правила **sqli**. Воно має наступний вигляд (таблиця 2.10).

Таблиця 2.10 – Базове правило для аналізу SQL ін'єкцій

Назва правила	Опис
sqli	.*? (union_select boolean_sql timebased_sql stacked_query) .*?;

2.3.3 Нотації для виявлення XSS

Спочатку опишемо базове правило, в якому подамо перелік шуканих конструкцій, від якого ми і будемо відштовхуватися (таблиця 2.11). Даний перелік створений на основі статті порталу OWASP [15].

Таблиця 2.11 – Базове правило для аналізу конструкцій XSS

Назва правила	Опис
xss_rules	.*? (script_tag_statement script_statement console_statement popup_statement to_string_usage dom_obj_injection js_function find_statement) .*? ;

Отже, **script_tag_statement** – це правило для пошуку включення HTML-тегів `<script>` та `</script>`. **Script_statement** описує включення типу «**javascript:alert(1)**». Замість «**javascript**» можуть вживатися також **ECMAScript**, **VBScript**, **ActionScript**, **TypeScript** тощо. **Console_statement** знаходить використання таких функцій як, наприклад, **console.log()** або **console.error()**, які виводять повідомлення у консолі браузера. **Popup_statement** використовується для виявлення функцій **alert()**, **prompt()**, **confirm()** та **MsgBox()** (**VBScript**). Дані функції виводять спливаючі вікна із заданою інформацією.

Далі описані деякі дещо незвичайні конструкції, які можуть використовуватися при XSS. Так за допомогою функції `toString()` можна замаскувати використання `prompt`, а вирази (2.12) та (2.13) є еквівалентними.

`867982141..toString(32)("XSS")` (2.12)

`prompt("XSS")` (2.13)

dom_obj_injection відповідає використанню виразів типу `document.cookie`, `top["confirm"](1)` та багатьом іншим за участю ключових слів `document`, `window`, `top`, `this`, `name`, `location`, `cookie` тощо. **Js_function** описує правила для `eval`, `setInterval`, `setTimeout`, `parseInt`.

Дуже цікавим є останнє із зазначених у таблиці 2.11 правило, а саме **find_statement**. Річ у тім, що можна задати певний масив елементів, а потім до нього застосувати функцію `find` з параметром – певною необхідною функцією, як це показано у (2.14). Результатом виконання буде виведення повідомлення «XSS» у спливаючому вікні.

`["XSS"].find(alert)` (2.14)

Таблиця 2.12 містить опис усіх вище згаданих правил.

Таблиця 2.12 – Правила для виявлення XSS

Назва правила	Опис
script_tag_statement	LT_SIGN CUTTABLE* SCRIPT .*? GT_SIGN? LT_SIGN CUTTABLE* SLASH CUTTABLE* SCRIPT CUTTABLE* GT_SIGN ;
script_statement	script_word CUTTABLE* COLON CUTTABLE* (console_statement popup_statement) ;
script_word	JAVASCRIPT ECMASCRIPT TYPESCRIPT VBSCRIPT ACTIONSCRIPT ;

Кінець таблиці 2.12

Назва правила	Опис
console_statement	CONSOLE CUTTABLE* POINT CUTTABLE* (LOG ERROR ASSERT GROUP WARN INFO TABLE) (CUTTABLE R_PAREN)* L_PAREN .*? R_PAREN ;
popup_statement	popup_word (CUTTABLE R_PAREN)* L_PAREN .*? R_PAREN ;
to_string_usage	POINT CUTTABLE* TOSTRING CUTTABLE* L_PAREN (SEPARATOR CUTTABLE)* NUMBER (SEPARATOR CUTTABLE)* R_PAREN ;
dom_obj_injection	(THIS DOCUMENT WINDOW TOP) CUTTABLE* L_BRACK .*? R_BRACK (THIS DOCUMENT WINDOW TOP) CUTTABLE* POINT CUTTABLE* (NAME LOCATION OPEN COOKIE DOMAIN WRITE HOSTNAME GET CREATE) ;
js_function	(EVAL SETINTERVAL SETTIMEOUT) CUTTABLE* L_PAREN .*? R_PAREN PARSEINT CUTTABLE* L_PAREN CUTTABLE* WORD CUTTABLE* COMMA CUTTABLE* NUMBER CUTTABLE* R_PAREN ;
find_statement	L_BRACK .*? R_BRACK CUTTABLE* FIND CUTTABLE* L_PAREN .*? R_PAREN ;

Взагалі, детально описати правила та повністю забезпечити неможливість проведення XSS атак неможливо, адже будь-який код може бути обфускований з використанням лише певних символів, серед яких круглі, квадратні та фігурні дужки, знаки «+», крапки та деякі інші, що робить такі конструкції нерозпізнаваними. У такому випадку найлегшим та найдієвішим

способом боротьби з атаками буде застосування фільтрації та заборона усіх вище згаданих символів.

2.3.4 Нотації для виявлення HTML ін'єкцій

Як вже зазначалось, HTML ін'єкції зазвичай використовуються для проведення XSS, а тому правил, поданих у попередньому пункті, насправді буде достатньо. Так, наприклад, (2.15) є прикладом, який подається на сайті OWASP, і він буде розпізнаний як атака, адже містить конструкцію «javascript:alert('XSS')», що відповідає правилу script_statement. Попри це у (2.15) також присутній тег DIV, що може вважатися, включенням HTML.

<DIV STYLE="background-image: url(javascript:alert('XSS'))"> (2.15)

Враховуючи вище сказане, усі правила у даному пункті будуть зведені до пошуку тегів та матимуть вигляд подібний до script_tag_statement (табл. 2.13) і при окремому використанні будуть скоріше корисними, якщо користувач хоче заборонити включення будь-яких елементів HTML розмітки.

Таблиця 2.13 – Правила для HTML ін'єкцій

Назва правила	Опис
html_tag_injection	LT_SIGN CUTTABLE* HTML_TAG .*? GT_SIGN? LT_SIGN CUTTABLE* SLASH CUTTABLE* HTML_TAG CUTTABLE* GT_SIGN ;
HTML_TAG	A P DIV SPAN IMG STYLE HTML BODY HEAD ;

2.3.5 Нотації для XXE ін'єкцій

У більшості випадків пошук зовнішніх сутностей XML (XXE) зводиться до пошуку сигнатури, яка містить тег ENTITY. Однак просто ENTITY

даватиме багато хибних результатів, адже можливі конструкції типу (2.16). Шукані сигнатури окрім цього повинні містити ще ключове слово SYSTEM та певний URL. Правила для аналізу вхідних параметрів на предмет наявності XXE ін'єкцій подані у таблиці 2.14. Список протоколів може бути розширений.

<!ENTITY car “Lamborghini Diablo”> (2.16)

Таблиця 2.14 – Нотації для XXE ін'єкцій

Назва правила	Опис
xxe	LT_SIGN (SEPARATOR CUTTABLE)* EXCL_SIGN (SEPARATOR CUTTABLE)* ENTITY (SEPARATOR CUTTABLE)+ WORD (SEPARATOR CUTTABLE)+ SYSTEM (SEPARATOR CUTTABLE)+ PROTOCOL CUTTABLE* COLON CUTTABLE* SLASH ;
PROTOCOL	FILE HTTP HTTPS FTP ;

Висновки до розділу 2

У даному розділі запропоноване рішення проблеми включення конструкцій, які можуть бути вирізані при попередній обробці вхідних даних, а також використання різного виду кодування. Описується модель побудови правил для виявлення таких конструкцій. Дана модель має ієрархічний характер, що дозволяє вносити зміни до окремих компонентів, не змінюючи загальний вираз. Водночас коригування одного фрагменту модифікує усі правила, до складу яких він входить. Це робить процедуру вдосконалення сигнатур для пошуку шкідливого навантаження більш зручною та швидкою.

Реалізація запропонованої моделі відбувається за допомогою форми Бекуса-Наура та генератора аналізаторів ANTLR4. Надане обґрунтування вибору саме такого інструменту, а також подані деякі загальні відомості, які

мають відношення до синтаксичного та лексичного аналізу, а також видів граматик. Розглянуті найпопулярніші види атак на веб-застосунки для SQL ін'єкцій та XSS. Для кожного виду описані правила виявлення. Додатково викладені правила для HTML ін'єкцій, які не покриті правилами XSS, а також для XXE, які переважно зводяться до пошуку конструкції єдиного виду.

3 ОЦІНКА РОЗРОБЛЕНОГО РІШЕННЯ

3.1 Створення тестового середовища

У якості тестового середовища виступатиме веб-застосунок. Так як до переліку мов програмування, у код яких перетворюються правила, описані за допомогою БНФ, входить Python3, він і стане інструментом для розробки. Причини вибору Python3:

- Орієнтованість на веб-розробку – велика кількість фреймворків;
- Легкість написання застосунків: за допомогою Flask типова програма «Hello world» пишеться у 5 рядків [16] (у даному випадку маємо готовий сервер).

Саме Flask був обраний у якості базового фреймворку. Так як акцент у даній презентації робиться зовсім не на програмуванні на Python, то будуть висвітлені лише необхідні для розуміння процесу аналізу вхідних параметрів моменти. З тих же міркування програма буде зовсім примітивною та обмеженою у функціоналі.

Отже, основною деталлю нашої програми повинно бути місце, куди користувач надсилатиме запити зі своїми параметрами. У даному випадку цю роль гратиме форма пошуку. У цілому веб-застосунок матиме вигляд дошки оголошень (рис. 3.1), у якій розміщена інформація про певні товари.

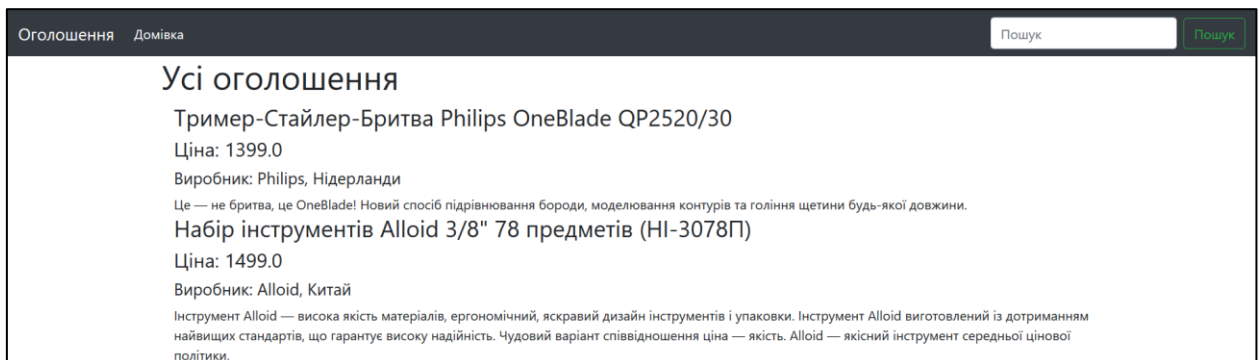


Рисунок 3.1 – Вигляд тестової програми

За умов пошуку URL запит матиме наступний вигляд:

`http://localhost/ads?q=Philips` (3.1)

Цікавими у контексті дослідження у даному випадку буде параметр «q». Однак окрім цього параметру можуть бути присутні інші. При проведенні автоматизованого тестування, деякі сканери вразливостей додають певні свої параметри, використовуючи свої бази найуживаніших аргументів. Дуже спрощено перебіг виконання програми можна зобразити так, як це показано на рисунку 3.2.

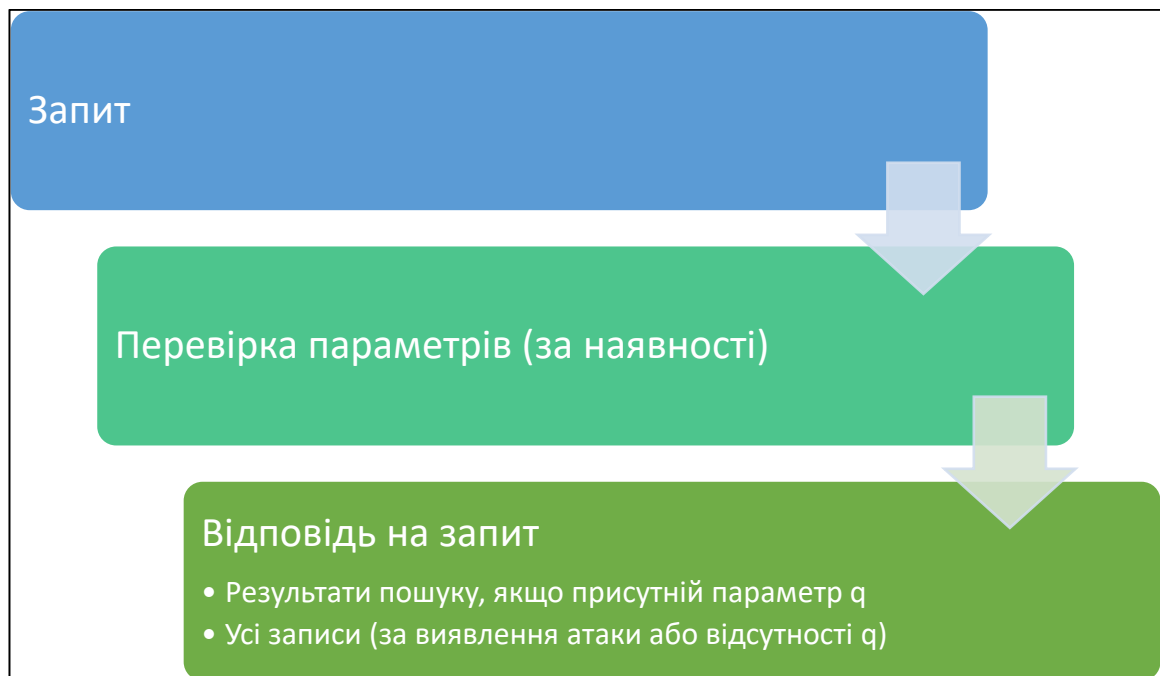


Рисунок 3.2 – Спрощена схема перебігу виконання програми

При генерації коду за допомогою ANTLR4 створюється 3 файли на кожен граматику: Lexer, Parser та Listener (або Visitor, на вибір). Очевидно, що Lexer займається лексичним аналізом, а Parser – синтаксичним. Виходом парсера слугує абстрактне синтаксичне дерево (АСТ), з яким і працюють у подальшому ті компоненти, які лишилися. Visitor – це реалізація загальновідомого шаблону проектування Відвідувач [17, с. 331], тому зупинятися на ньому не будемо, до того ж він не буде використовуватися у даній дисертації. Натомість буде використаний Listener. Він не є типовим

прикладом шаблону Спостерігач [17, с. 293], натомість більше схожий навіть на той самий Відвідувач. При використанні цього патерну також відбувається обхід дерева. При вході до кожної вершини (кожного правила) викликається відповідна функція *enterRule*, а при виході з неї – *exitRule*. Наприклад, для *boolean_sqli* це будуть *enterBoolean_sqli* та *exitBoolean_sqli*. На прикладі ілюстрації (рис. 3.3) можна побачити структуру АСД для певного запиту, який містить SQL ін'єкцію на основі контенту. Вхід буде відбуватися зверху донизу, вихід – у зворотному напрямку.

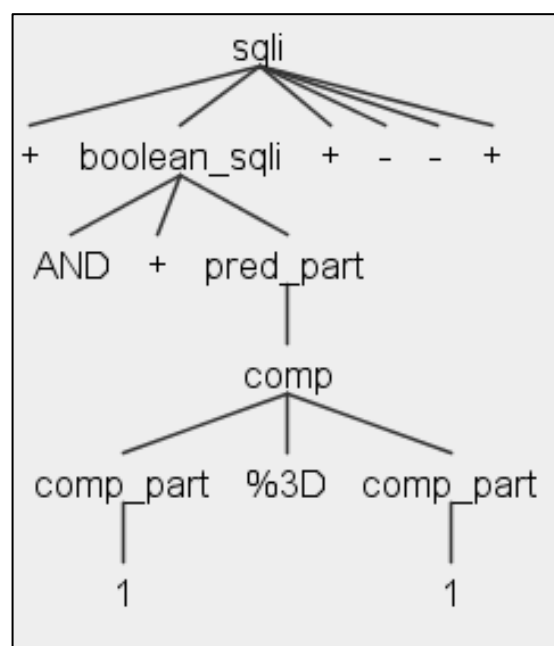


Рисунок 3.3 – Приклад вигляду АСТ

Стандартний Listener виступає у якості інтерфейсу. Аби реалізувати функції треба створювати власні класи, які успадковуватимуть методи класу Listener. У даному випадку маємо *SqliTestingListener*, *XssTestingListener*, *HtmlTestingListener* та *XxeTestingListener*. Кожен із них має змінну *state*, яка є відображенням стану та в якій фіксуватиметься наявність атаки. Для виявлення атаки достатньо реалізувати одну із функцій: входу або виходу із правила, що нас цікавить. Оскільки цікавим є правило, найбільш наближене до кореня АСТ, буде використаний метод *exitRule*. Приклад реалізації Listener наведено нижче (3.2).


```
class XxeTestingListener(xxeListener):

    def __init__(self):

        self.state = {'attack':False}

    def exitXxe(self, ctx:xxeParser.XxeContext):

        self.state = {'attack' : True, 'type':'XXE injection'}
(3.2)
```

У цілому процес аналізу параметрів та їхнього перетворення для кожної з чотирьох граматики відбувається за наступною схемою (рис. 3.4).



Рисунок 3.4 – Схема процесу аналізу параметра

Так як виявлення відбувається на стороні сервера застосунків, можна стверджувати, що реалізований певний варіант WAF на основі хосту.

3.2 Проведення тестів

ANTLR4 першочергово був написаний на Java. Тому до його складу входять деякі готові засоби для проведення синтаксичного аналізу та візуалізації результатів, що дає змогу провести ручне тестування та

переконалися у правильності побудованих правил. Розглянемо декілька прикладів. У першому прикладі спробуємо ввести SQL ін'єкцію (3.3).

```
123 REVdrop\u0045RSE(NO#####I/**/NU)+/*656a848d16aw1e21c261*/
+Sunion%65LE%00C/*5a`6s5d16a5s1d56546*/t '1', '2', '3'
```

(3.3)

Як можна побачити, у даному прикладі для маскування використовується функція REVERSE, для розбиття: ключові слова drop та select, коментарі, символи «#» та нульовий байт (%00), для кодування: коди Unicode (E => \u0045) та URL (E => %65). На рисунку 3.5 зображений результат парсингу даної конструкції. (3.4) є прикладом логу із файлу для фіксування атак та результатом виявлення впровадження (3.3). Формат логу є наступним: дата – адреса запиту – метод – тип атаки.

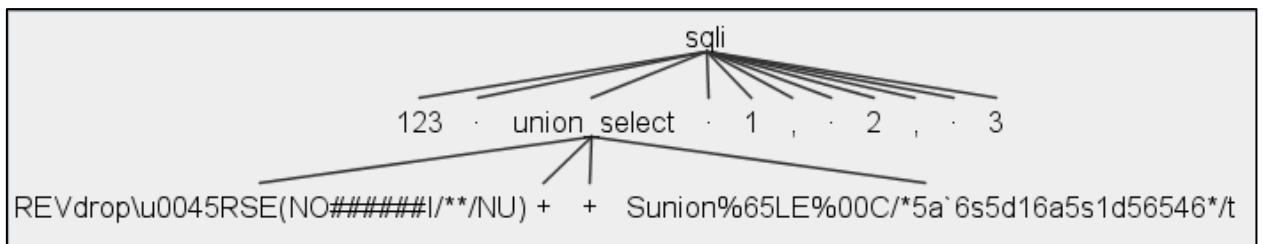


Рисунок 3.5 – Результат парсингу параметру (3.3)

2019-10-09 09:01:59.013272

```
/ads?q=123+REVdrop%5Cu0045RSE%28NO%23%23%23%23%23%23I%2F**
%2FNU%29%2B%2F*656a848d16aw1e21c261*%2F%2BSunion%2565LE%250
0C%2F*5a%606s5d16a5s1d56546*%2Ft+%271%27%2C+%272%27%2C+%273
%27 GET Union-based SQL injection
```

(3.4)

Розглянемо ще один приклад. Впровадимо у програму наступну конструкцію (3.5).

<IMG

```
SRC=&#0000106&#0000097&#0000118&#0000097&#0000115&#0000099&#00
00114&#0000105&#0000112&#0000116&#0000058&#0000097&#0000108&#0
```

000101rt('XSS')> (3.5)

За допомогою кодів HTML замасковано вираз «javascript:alert('XSS')». На рисунку 3.6 показане АСД для (3.4). З ілюстрації видно, що параметри міститься script_statement, який складається з script_word (у даному випадку javascript), двокрапки та popup_statement, яке містить popup_word (alert), та ('XSS') (символи лапок проігноровані). Результат виявлення даної атаки поданий нижче (3.6).

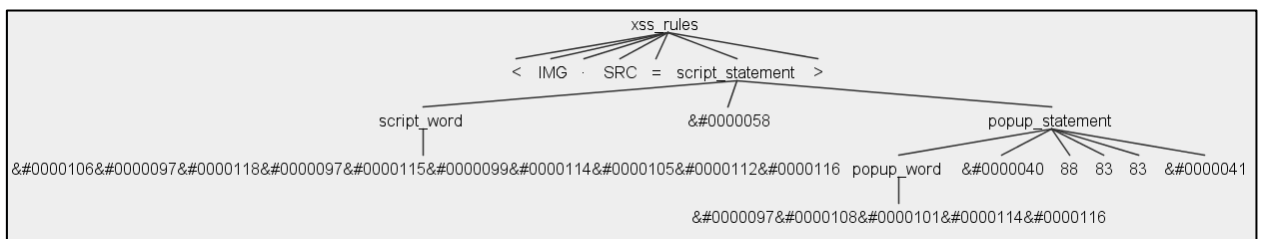


Рисунок 3.6 – Результат парсингу (3.5)

2019-10-09 09:07:07.845937

/ads?q=%3CIMG+SRC%3D%26%230000106%26%230000097%26%230000118%26%230000097%26%230000115%26%230000099%26%230000114%26%230000105%26%230000112%26%230000116%26%230000058%26%230000097%26%230000108%26%230000101%26%230000114%26%230000116%26%230000040%26%230000039%26%230000088%26%230000083%26%230000083%26%230000039%26%230000041%3E GET XSS: Usage of "JavaScript:", "VBScript:" etc + function (3.6)

Проте ручне тестування не дає повного уявлення про коректність запропонованого рішення. Тому необхідно запровадити автоматизацію. Усі запити фіксуються у файлі з логами, окремо фіксуються атаки. Ефективність роботи системи виявлення можна досліджувати шляхом порівняння цих двох файлів. Для автоматизованого тестування використовувалися такі програмні засоби:

- OWASP ZAP;
- Netsparker;
- Acunetix;
- sqlmap.

Для оцінки ефективності описаної у даній дисертації системи виявлення пропонується порівняти її із готовим рішенням, яке використовує підхід токенізації – libinjection. Даний засіб входить до складу популярних web application firewall з відкритим програмним кодом, як, наприклад, Modsecurity або Nginx Anti-XSS & SQL Injection. Даний продукт зосереджує свою увагу на SQL ін'єкціях та XSS. Засоби для тестування, подані вище, мають більші бази саме для атак даного типу. Методи маскування шкідливого навантаження, описані у попередніх розділах та які стали причиною написання даної дисертації, також переважно стосуються саме SQL ін'єкцій та XSS. Саме тому акцент при тестуванні був зроблений саме на них.

3.3 Оцінка результатів

Результати проведеного тестування наведені у таблиці 3.1.

Таблиця 3.1 – Результати тестування

Атака	Libinjection	Розроблене рішення	Загальна кількість шкідливих запитів
SQL ін'єкції (сканери)	10422	10416	12833
SQL ін'єкції (модифіковані)	2060	6783	12035
XSS (сканери)	45	47	203
XSS (модифіковані)	110	150	209

Таким чином можна побачити, що результати тестування для libinjection та розробленого рішення є приблизно однаковими для чистих (мається на увазі без жодних модифікацій) виводів сканерів. Було виявлено **81,21%** та **81,16%** SQL ін'єкцій, а також **22,17%** та **23,15%** спроб XSS атак відповідно. Низькі показники для Cross-Site Scripting attacks пов'язані із використанням сканерами дивних конструкцій для тестів. Так, наприклад, Netsparker для тестування XSS надсилав

%3CscRipt%3Enetsparker(0x0001F1)%3C/scRipt%3E (3.7),

тобто використовував замість alert(0x0001F1) netsparker(0x0001F1). Логіка розробників у створенні даної конструкції лишається не зрозумілою, адже неочевидною є ситуація, за якої дане навантаження мало призвести до певних наслідків.

Неабсолютне виявлення SQL ін'єкцій для описаного рішення пов'язане з недосконалістю описаних граматик, а саме відсутністю правил для деяких конструкцій, що є притаманним для засобів виявлення, які використовують сигнатурний підхід. Серед шкідливих параметрів, які обійшли засоби захисту можна виокремити наступні:

- SELECT (CASE WHEN (...) THEN ... ELSE ...));
- AND SELECT ...;
- AND NUMBER = CAST(...);
- AND NUMBER = CONVERT(...);
- AND NUMBER = (SELECT UPPER (XMLType(...)));
- SELECT CHAR(...);
- document.createElement(...);
- тощо.

На відміну від тестування на чистих виводах сканерів, використання модифікованого навантаження виявило певні розбіжності у роботі аналізованих рішень. До складу параметрів, які застосовувалися для

тестування, увійшли приклади зі сторінок веб-ресурсу OWASP [11, 15], а також масковані за допомогою вставлення символів, які можуть бути вирізані, та кодування конструкції із баз даних сканерів. Попередньо були відфільтровані параметри, які не мають сенсу, як (3.7), згаданий попередньо. Результати для SQL ін'єкцій є наступними: libinjection – **17,12%**; розроблене рішення – **56,36%**. Для XSS: **52,63%** та **71,77%** відповідно. Очевидною є перевага розробленого рішення. Однак необхідно зауважити, що дане тестування є достатньо суб'єктивним. Найбільш об'єктивні результати можна отримати лише за умов використання у реальних інформаційних системах, зібравши велику вибірку з шкідливих вхідних параметрів.

Розглянемо переваги та недоліки запропонованого рішення.

Серед переваг можна виокремити наступні:

- здатність виявляти передові атаки, пов'язані зі спробами обійти засоби захисту;
- можливість перетворення правил БНФ у код популярних мов програмування, через що вони можуть бути використані і у мережевому, і у хмарному, і у хостовому WAF;
- вирішення токенбрейкінгу за рахунок доповнення опису токена CUTTABLE;
- ієрархічна структура, яка дозволяє змінювати конкретні токени та субправила, залишаючи опис загальної конструкції без модифікацій. Водночас зміна одного компоненту впливає на усі правила, у яких він зустрічається. Це робить процес редагування простішим та зручнішим.

Серед недоліків вагомими є такі:

- треба приділяти значну увагу опису токенів. Якщо токен описаний як фрагмент, але може використовуватися самостійно, можуть бути проблеми з розпізнаванням конструкції у цілому;

- існує ризик отримати REDOS, тобто DOS-атаку на основі регулярних виразів, через використання символів «*» та «+» (наприклад CUTTABLE*), що вказують на нескінчену кількість знаків, які задовольняють правило; може бути вирішене встановленням верхньої границі, однак це не є простим завданням через особливості синтаксису РБНФ ANTLR4, а також через питання «якою має бути ця границя»;

Ще одним пунктом, що є скоріше не недоліком, а реалією життя, є питання, наскільки детально треба описувати правила, що є по суті проблемою компромісу та мінімізації помилок першого та другого роду. Для удосконалення нотацій слід провести більш тривале тестування у реальних умовах.

Висновки до розділу 3

У даному розділі надаються результати тестування для програмної реалізації моделі, описаної у другому розділі. Через простоту та швидкість розробки була обрана мова програмування Python3, фреймворк Flask. Запропоноване рішення порівнюється із популярним продуктом з відкритим програмним кодом libinjection.

Результати для чистих параметрів – виходів сканерів – свідчать про те, що розроблене рішення (SQLi – **81,16%**, XSS – **23,15%**) не поступається у ефективності libinjection (SQLi – **81,21%**, XSS – **22,17%**). Натомість за модифікації навантаження запропонована модель дає кращий результат (**56,36%** та **71,77%** проти **17,12%** та **52,63%** відповідно).

Був виокремлений ряд переваг, серед яких здатність виявляти передові техніки обходів механізмів захисту, реалізація для поширених мов програмування, легкість та швидкість модифікації правил. Натомість існують

певні недоліки такі як можливість REDOS, а також помилки через неточності у порядку опису конструкцій та проведення перевірок.

Описаний у даному розділі режим роботи програми полягає у блокуванні подальшого виконання запиту, однак у майбутньому допускається варіант із фільтрацією вхідних параметрів шляхом видалення шкідливих конструкцій.

4 РОЗРОБКА СТАРТАП-ПРОЕКТУ

У даному розділі наведені результати проведеного маркетингового аналізу перспектив реалізації запропонованого у попередніх розділах рішення для виявлення просунутих атак типу включення та оцінка можливостей його ринкового впровадження. Описані правила та згенерований програмний код самі по собі можуть бути продуктом, який може використовуватися клієнтами для їхніх власних розробок, що дозволить створити бізнес та отримувати прибутки. Натомість раніше у даній дисертації зазначалося, що проблема, яка вирішувалася під час цього дослідження, притаманна конкретному типу програмного та/або апаратного забезпечення, що має назву Web Application Firewall. Окрім цього згадувалися інші рішення типу IPS/IDS, NGFW тощо, однак зона їх відповідальності є значно більшою. З урахуванням цього фактору, а також того факту, що повноцінний фаєрвол прикладного рівня виконує ширший спектр функцій, аніж запропоноване рішення, далі у цьому розділі у якості стартап-проекту буде розглядатися WAF з певним стандартним для цього продукту набором механізмів, проте ядро якого складатимуть подані у попередніх розділах правила та програмні модулі їх реалізації. Дана ідея вважається основною для застосування на практиці.

4.1 Опис ідеї проекту

Початком маркетингового аналізу виступає опис ідеї проекту. Отже, даний проект спрямований на захист веб-застосунків від атак типу включення шкідливих конструкцій у якості вхідних параметрів, а також на виявлення та протидію маскуванню такого роду конструкцій за допомогою додавання спеціальних символів з передбаченням їхнього подальшого видалення при фільтрації та використання різного роду кодування. Зручно подавати опис ідеї у вигляді таблиці (табл. 4.1).

Таблиця 4.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Продукт типу WAF із покращеними механізмами захисту від атак типу включення.	1. Забезпечення безпеки веб-застосунків шляхом блокування виконання шкідливих запитів.	Захист від несанкціонованого доступу до даних та їхньої модифікації, а також від надмірного завантаження внутрішніх ресурсів (наприклад, використання функції BENCHMARK для MySQL), що може призвести до DoS.
	2. Забезпечення безпеки веб-застосунків шляхом видалення виявлених шкідливих конструкцій.	
	3. Процеси пов'язані із питаннями комп'ютерної форензики, такі як аналіз логів на предмет наявності шкідливих запитів.	Кращі можливості для виявлення причин збоїв веб-застосунку, бази даних, серверу тощо, усунення їх наслідків та попередження нових атак.

Наступним кроком є виокремлення основних конкурентів та порівняння їх між собою та запропонованим власним продуктом. Під час цього процесу виявляються сильні та слабкі сторони рішень, які порівнюються.

Конкурентами виступили: Imperva SecureSphere WAF (Конк.1), Cloudflare WAF (Конк.2) та ModSecurity (Конк.3). Перші два є представниками більш дорогих та потужних засобів захисту веб-застосунків. Третій є представником програмного забезпечення з відкритим вихідним кодом, який використовує libinjection, яка згадувалася у попередньому розділі. Результати порівняння подані у таблиці 4.2. Літери W (weak), N (neutral), S (strong) означають слабку, нейтральну та сильну сторону відповідно.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п / п	Техніко- економічні характеристики ідеї	(Потенційні) товари / концепції конкурентів				W	N	S
		Мій проект	Конк. 1	Конк. 2	Конк. 3			
1	Швидкодія	Висока	Висока	Висока	Висока		+	
2	Вартість	Середня	Висока	Висока	Низька		+	
3	Захист від SQL ін'єкцій;	Високий	Високий	Високий	Високий		+	
4	Захист від XSS;	Високий	Високий	Високий	Високий		+	
5	Захист від ін'єкцій коду;	Середній	Високий	Високий	Середня	+		
6	Поведінковий аналіз та аналіз репутації;	Середній	Середній	Високий	Низький		+	
7	Захист від XXE;	Високий	Низький	Середній	Низький			+
8	Захист від передових атак типу включення;	Високий	Середній	Середній	Середній			+

4.2 Технологічний аудит ідеї проекту

У даному підрозділі необхідно визначити технології, за допомогою яких можна реалізувати ідею. Потрібно провести аудит та зробити висновки щодо наявності та доступності даних технологій. Результати наведені у таблиці 4.3.

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п / п	Ідея проекту	Технології реалізації	Наявність технологій	Доступність технологій
1	Програмне забезпечення для захисту веб-застосунків	БНФ, генератор парсерів та лексерів, мови програмування: C++, Python, Java, JavaScript, C#, Go, Swift	Наявні	Доступні
2	Пристрій для захисту веб-застосунків	Апаратні компоненти: (плати, процесори тощо), БНФ, генератор парсерів та лексерів, мови програмування: C++, Python, Java, JavaScript, C#, Go, Swift	Наявні	Доступні
3	Програмне забезпечення для розслідування інцидентів ІБ	БНФ, генератор парсерів та лексерів, мови програмування: C++, Python, Java, JavaScript, C#, Go, Swift	Наявні	Доступні

Очевидно, що усі технології реалізації є доступними. Проте для подальшої роботи обрано перше технологію.

4.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку

проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів [18]. Першим етапом є аналіз попиту. Результати наведені у таблиці 4.4.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	10
2	Загальний обсяг продаж	> 100 млн. \$
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу	Немає
5	Специфічні вимоги для стандартизації, специфікації	Немає
6	Середня норма рентабельності в галузі, %	~75%

За попереднім оцінюванням можна зробити висновок, що даний ринок є привабливим для входження, як і будь-який пов'язаний з послугами у сфері інформаційної та кібернетичної безпеки. Так як щороку кількість компаній, які дбають про свою захищеність зростає, і прогнозується, що даний тренд буде зберігатися, а можливо посилюватися у найближчі роки, будь-який стартап має шанси знайти своїх клієнтів та зайняти свою нішу. Єдиною перепорою є наявність великих та більш потужних у плані можливостей конкурентів.

Наступним кроком є визначення потенційних груп клієнтів та їх характеристик. Після цього відбувається формування орієнтовного переліку вимог до товару.

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п / п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Захист веб-застосунків від атак	<ul style="list-style-type: none"> • підприємства; • окремі користувачі. 	<ul style="list-style-type: none"> • взаємодія з підприємствами може полягати у підписанні додаткових положень, документів, співпраці із адміністраторами та наданні їм додаткових можливостей з налаштування та модифікацій продукту; • окремі користувачі зазвичай відштовхуються від умов, які пропонує виробник. 	<ul style="list-style-type: none"> • можливість вибору режиму роботи (блокування запитів або фільтрація); • можливість вмикання / вимикання певних функцій; • зрозуміла система логування; • швидкодія; • зручний інтерфейс.

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. 4.6) [18].

Таблиця 4.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Відсутність репутації	Клієнт не має жодного уявлення ні про якість продукту, ні про його переваги над існуючими популярними рішеннями.	Проведення рекламної кампанії, участь у конференціях, форумах, виставках.

Далі описуються фактори можливостей продукту (таблиця 4.7).

Таблиця 4.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Поява нових типів атак на веб-застосунки	Поява нових атак означає той факт, що у світі інформаційних технологій ще немає готового рішення для даної проблеми.	Розробка інноваційного рішення, пропонування додаткових послуг.
2	Підвищення попиту на рішення безпеки	Усе більше компаній та окремих користувачів усвідомлюють, що забезпечення безпеки є невід’ємною частиною їх функціонування у інформаційному просторі.	Збільшення обсягів продажів, укладення нових договорів, перегляд цінової політики.

Ще однією невід’ємною частиною ринку є пропозиція, тому далі будуть оцінюватися її характеристики. Першим показником є конкуренція. Результати аналізу можна побачити у таблиці 4.8.

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (дії компанії для конкурентоспроможності)
1. Конкуренція – чиста.	Жоден з конкурентів не може впливати на загальну ситуацію на ринку або цей вплив є таким незначним, що ним можна знехтувати.	Розробка нових функцій до продукту, проведення рекламних кампаній задля зацікавлення клієнтів.
2. Рівень конкурентної боротьби – світовий.	Не залежить від географічного розташування.	Інтернаціональна маркетингова кампанія, технічна підтримка різними мовами.
3. За галузевою ознакою – міжгалузева.	Споживачі продукту можуть бути представниками різних галузей.	Пошук клієнтів, у галузях які тільки починають розвиватися.
4. Конкуренція за видами товарів: товарно-родова.	Різні за набором функцій продукти можуть виконувати захист веб-застосунків.	Виявлення переліку функцій, які є найбільш затребуваними.

Кінець таблиці 4.8

5. За характером конкурентних переваг: нецінова	Ціна на є основним чинником для споживача при виборі продукту.	Забезпечення найвищої якості при захисті веб-застосунків.
6. За інтенсивністю: не марочна	Ціни відрізняються для різних пакетів послуг.	Виділення унікальних цін.

Наступним кроком є аналіз умов конкуренції в галузі за моделлю М. Портера (табл. 4.9).

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти у галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	Cloudflare, Imperva, ModSecurity ...	Розмір вкладень, наявність патентів, якість продукту.	Поширення через інтернет у будь-яку точку світу.	Чутливість до зміни цін, відгуки.	Краща якість, нижча ціна, впізнаваність бренду.
Висновки	Достатньо інтенсивна боротьба між конкурентами.	Вихід на ринок можливий, однак конкуренти присутні.	Можуть диктувати умови: ціни на послуги.	Можуть вимагати додавання нових функцій.	Явних обмежень немає.

Аналізуючи таблицю 4.9 можна сказати, що можливість виходу на ринок є, однак боротьба між конкурентами є достатньо високою. На вибір клієнтів можуть впливати якість продукту, цінова політика та додавання нових функцій.

На основі аналізу конкуренції (табл. 4.9), а також із урахуванням характеристик ідеї проекту (табл. 4.2), вимог споживачів до товару (табл. 4.5) та факторів маркетингового середовища (табл. №№ 4.6-7) визначається та обґрунтовується перелік факторів конкурентоспроможності [18]. Результати даного аналізу представлені у таблиці 4.10.

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Репутаційний	Відома компанія з високими показниками успішності впровадження своїх рішень з дуже високою ймовірністю отримає більше нових клієнтів.
2	Технологічний	Нові якісні характеристики продукту можуть змусити клієнтів почати використовувати його навіть за наявності більш відомих конкурентів.
3	Економічний (ціновий)	Очевидно, що за однакових якостей, споживач з дуже високою ймовірністю обирає дешевший товар.

На основі факторів з таблиці 4.10 проводиться порівняльний аналіз сильних та слабких сторін проекту. Результати наведені у таблиці 4.11.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з стартап-проектом						
			-3	-2	-1	0	+1	+2	+3
1	Репутаційний	12							+
2	Технологічний	17		+					
3	Економічний	14						+	

Кінцевим кроком аналізу можливостей щодо впровадження проекту є складання SWOT-аналізу. На даному етапі будується матриця, яка складається з сильних (Strength) та слабких сторін (Weak), можливостей (Opportunities) та загроз (Troubles). Результати наведені у таблиці 4.12.

Таблиця 4.12 – SWOT-аналіз стартап-проекту

Сильні сторони: <ul style="list-style-type: none"> - Висока якість - Інноваційні рішення 	Слабкі сторони: <ul style="list-style-type: none"> - Маловідомість - Мала кількість оборотних коштів
Можливості: <ul style="list-style-type: none"> - Нові рішення для актуальних проблем - Розширення ринку - Нові потреби користувачів 	Загрози: <ul style="list-style-type: none"> - Продукти-замінники - Цінова політика конкурентів

Після проведення SWOT-аналізу та на його основі розробляються альтернативи ринкової поведінки (таблиця 4.13).

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Прямий пошук інвестицій	Середня	4 місяці
2	Залучення мінімальних початкових інвестицій та подальший розвиток	Висока	Півроку
3	Інтенсивна рекламна кампанія	Висока	3 місяці

Проведення рекламної кампанії виглядає заманливо з урахуванням коротких термінів реалізації, однак вона потребує значних витрат. Тому обрано залучення мінімальних початкових інвестицій з подальшим розвитком.

4.4 Розроблення ринкової стратегії проекту

Першим кроком розроблення ринкової стратегії є визначення стратегії охоплення ринку, а саме опис цільових груп потенційних споживачів, що проведено у Таблиці 4.14.

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Окремі користувачі	Висока	Високий	Висока	Середня
2	Компанії	Середня	Середній	Висока	Висока

Виходячи з показників таблиці 4.14, для початку доцільніше обрати окремих користувачів за цільову групу. Компанії часто співпрацюють одна з одною, маючи контракти та використовуючи продукти одна одної, тому ймовірність успіху за таких умов є дещо нижчою, аніж при першому варіанті. Натомість оцінка користувачів та співпраця з ними у питаннях розробки нових властивостей, задоволення потреб, удосконалення програми призведе до зростання популярності та до створення позитивної репутації.

Наступним етапом є формування базової стратегії розвитку в обраних сегментах ринку (таблиця 4.15).

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Окремі користувачі	Масовий маркетинг	Краща якість, втілення продукту у різних формах (як хмарний сервіс, як ПЗ, яке можна встановити на своєму пристрої), підтримка різних мов програмування	Стратегія диференціації

Отже, була обрана стратегія диференціації, адже товару надаються важливі з точки зору споживача примітних властивостей, які роблять товар унікальним.

Після цього обирається стратегія конкурентної поведінки (табл. 4.16).

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Ні	Буде шукати нових та забирати існуючих у конкурентів	Так, найкращі та найнеобхідніші для користувачів	Стратегія виклику лідера: флангова атака

Наступним етапом є розробка стратегії позиціонування (табл. 4.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 – Визначення стратегії позиціонування

Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
Якість захисту, зручність, ціна	Стратегія диференціації	Інноваційність, широка область застосування, простота використання та налаштування	Надійність, якість, зручність

4.5 Розроблення маркетингової програми стартапу

Спочатку необхідно сформуванати маркетингову концепцію товару, який отримає споживач, на підставі результатів попереднього аналізу конкурентоспроможності товару (табл. 4.18).

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Захист від атак	Захист від просунутих атак	Окрім базових атак продукт виявляє просунуті атаки, спрямовані на експлуатацію вад механізмів захисту
2	Кросплатформеність	Можливість використання як компонентів для застосунків	У разі використання у режимі host-based WAF, даний продукт може бути інтегрований у застосунки, написані на 7 популярних мовах програмування.
3	Зручність та простота експлуатації	Можливість увімкнення та вимкнення функцій	Зручний інтуїтивно зрозумілий для кожного користувача інтерфейс для контролю над функціями продукту

Наступним етапом є розробка трирівневої маркетингової моделі товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (табл. 4.19).

Таблиця 4.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
I. Товар за задумом	ПЗ для захисту веб-застосунків від атак
II. Товар у реальному виконанні	Властивості/характеристики:
	<ul style="list-style-type: none"> • Захист від передових атак • Керування функціями, налаштування • Зручна система логування
	Якість: для оцінювання якості продукту використовуються сканери веб-вразливостей та інші засоби аналізу, а також ручне тестування.
	Пакування: інсталяційний пакет для відповідної ОС при використанні «on-premise» («на підприємстві», тобто реальне фізичне розміщення ПЗ на власному комп'ютері).
III. Товар із підкріпленням	До продажу: узгодження підписки, можливість доступу до документації та демонстраційного веб-сайту
	Після продажу: підтримка, оновлення
За рахунок чого потенційний товар буде захищено від копіювання (актуально для «on-premise»): Необхідність введення активаційного коду при встановленні, обфускація.	

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 4.20). Аналіз проводиться експертним методом [18]. Рівень цін вказаний за рік ліцензійного користування.

Таблиця 4.20 – Визначення меж для встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня меж і встановлення ціни на товар / послугу
~8000-15000\$	~9000-17000\$	~40000\$	Так як вказані у перших двох стовпцях ціни є платою за послуги для кількості серверів від 5 до 10, або для хмарного захисту кількох застосунків (до 25), для окремих користувачів ціна може бути нижчою. Встановимо наступні ціни на послуги 5000-15000\$.

Після цього визначається оптимальна система збуту (таблиця 4.21). У даному випадку приймається рішення проводити збут власними силами через електронні ресурси (Інтернет).

Таблиця 4.21 – Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Купівля через інтернет	Надсилання ПЗ (за необхідності)	Нульового рівня	Власний сайт, маркетплейс

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів [18] (табл. 4.22).

Таблиця 4.22 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Бажання отримати якісний продукт за менші кошти	Інтернет-ресурси	Інноваційність, широка область застосування, простота використання та налаштування	Надати інформацію про товар (послугу) потенційним споживачам	Найкращий захист веб-застосунків

Висновки до розділу 4

У даному розділі проведено маркетинговий аналіз перспектив створення власного програмного забезпечення для захисту веб-застосунків, основою якого є реалізація моделі, описаної у попередніх розділах. Окрім цього оцінена можливість його ринкового впровадження.

У результаті було виявлено, що даний проект має можливість до ринкової комерціалізації. Незважаючи на достатньо велику кількість конкурентів, що є вагомим фактором проти розвитку власного стартапу, попит на рішення безпеки є достатньо високим через появу нових загроз та нових клієнтів.

Основною сильною рисою є новий рівень якості послуг, пов'язаний із виявленням передових атак та технік обходу популярних механізмів захисту. Також перевагою продукту є те, що він може поставлятися як хмарний сервіс, так і у режимі «on-premise», а також може бути представлений у вигляді програмних модулів, інтегрованих у застосунки, написані на 7 поширених мовах програмування. Серед недоліків головним є відсутність репутації. Це призводить до малої кількості клієнтів на перших етапах розвитку компанії.

Для початкового етапу пропонується зосередити увагу на окремих користувачах – власниках невеликих веб-ресурсів. За набуття популярності необхідно розширювати коло клієнтів, заключаючи договори з компаніями. У цілому подальша імплементація вважається доцільною.

ВИСНОВКИ

У даній роботі проведено дослідження сучасних атак на веб-застосунки з використанням маскування для обходу механізмів захисту. Виявлено два основних підходи до маскування: використання кодування та розбиття шкідливих вхідних параметрів конструкціями, які можуть бути видаленими при фільтрації на певному із етапів обробки даних. Розглянуті основні методи виявлення атак, які використовуються у актуальних рішеннях сьогодення.

За основу для власного рішення взятий метод токенизації. Описана модель побудови правил з урахуванням згаданих підходів до маскування. Правила описані за допомогою розширеної форми Бекуса-Наура та реалізовані у вигляді програмного модуля за допомогою ANTLR4. Також було створене тестове середовище – веб-сайт написаний на Python3 з використанням фреймворку Flask.

Були проведені тестування для створеної програми та популярного рішення для токенизації та виявлення SQL ін'єкцій та XSS – libinjection. Отримані результати вказують на приблизно однаковий рівень якості обох продуктів при виявленні атак за допомогою сканерів веб-вразливостей. Натомість описане у даній роботі рішення має перевагу при використанні маскування, що підтверджене другою серією тестів.

Аналіз можливостей реалізації стартап-проекту на основі запропонованого рішення показав позитивні результати. Незважаючи, на те що є на даному ринку є достатня кількість гравців, нова якість при виявленні атак та зростання попиту на рішення безпеки, що у свою чергу призводить до розширення кола споживачів, робить ймовірність успіху достатньо високою.

Запропоноване рішення може бути використане у продуктах для захисту веб-застосунків у реальному часі, таких як WAF, IPS, IDS, NGFW тощо. Можлива реалізація на 7 мовах програмування, що робить можливість

імплементції описаної ідеї як на хмарних платформах або окремому пристрої у локальній мережі, так і у вигляді програмних модулів для використання всередині застосунків. Так само корисним буде його використання для аналізу логів при розслідуванні інцидентів. Не виключеним є застосування у інших сферах, усе залежить від кінцевих цілей та потреб.

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 SQL Injection Attacks and Defense 2nd Edition / Justin Clark, Kevvie Fowler, Erlend Oftedal, Rodrigo Marcos Alvarez and others – Syngress, 2012 – 576 pages
- 2 XSS Attacks: Cross Site Scripting Exploits and Defense 1st Edition, Kindle Edition, Seth Fogie – Syngress, 2011 – 480 pages
- 3 DOM Based XSS, OWASP [Електронний ресурс]:[Web-сайт] – Електронні дані – Режим доступу: https://www.owasp.org/index.php/DOM_Based_XSS (дата звернення 09.10.2019) – Назва з екрана.
- 4 Types of Cross-Site Scripting, OWASP [Електронний ресурс] : [Web-сайт] – Електронні дані – Режим доступу: https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting (дата звернення 09.10.2019) – Назва з екрана.
- 5 “DOM Based Cross Site Scripting or XSS of the Third Kind” (WASC writeup), Amit Klein [Електронний ресурс]:[Web-сайт] – Електронні дані – 2005 – Режим доступу: <http://www.webappsec.org/projects/articles/071105.shtml> (дата звернення 09.10.2019) – Назва з екрана.
- 6 «Настольный справочник по атакам на XML-приложения», Журнал «Хакер» [Електронний ресурс]:[Web-сайт] – Електронні дані – 2012 – Режим доступу: <https://haker.ru/2012/12/11/xml-apps-attacks-manual/> (дата звернення 09.10.2019) – Назва з екрана.
- 7 Extensible Markup Language (XML) 1.0 (Fifth Edition) [Електронний ресурс]:[Web-сайт] – Електронні дані – 2008 – Режим доступу: <https://www.w3.org/TR/xml/> (дата звернення 09.10.2019) – Назва з екрана.
- 8 Testing for HTML Injection (OTG-CLIENT-003), OWASP [Електронний ресурс] : [Web-сайт] – Електронні дані – Режим доступу: [https://www.owasp.org/index.php/Testing_for_HTML_Injection_\(OTG-CLIENT-003\)](https://www.owasp.org/index.php/Testing_for_HTML_Injection_(OTG-CLIENT-003)) (дата звернення 09.10.2019) – Назва з екрана.
- 9 Category:OWASP Top Ten Project, OWASP [Електронний ресурс] : [Web-сайт] – Електронні дані – Режим доступу:

- https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (дата звернення 09.10.2019) – Назва з екрана.
- 10 Web Application Firewalls: Attacking detection logic mechanisms, Vladimir Ivanov, Black Hat USA – 2016 – 60 pages
- 11 SQL Injection Bypassing WAF, OWASP [Електронний ресурс] : [Web-сайт] – Електронні дані – Режим доступу: https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF (дата звернення 09.10.2019) – Назва з екрана.
- 12 А. Ахо, Дж. Ульман. Теория синтаксического анализа, перевода и компиляции. Т. 1. Пер. с англ. В.Н. Агафонова под ред. В. М. Курочкина. М.: Мир, 1978. 614 с.
- 13 Теория и реализация языков программирования: Учебное пособие./ В.А.Серебряков, М.П.Галочкин, Д.Р.Гончар,М.Г.Фуругян. - М.: МЗ Пресс, 2006. - 352 с.: ил.ISBN 5-94073-094-9
- 14 Adaptive LL(*) Parsing: The Power of Dynamic Analysis, Terrence Parr, Sam Harwell, Kathleen Fisher – Technical report – 2014
- 15 XSS Filter Evasion Cheat Sheet, OWASP [Електронний ресурс] : [Web-сайт] – Електронні дані – Режим доступу: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet (дата звернення 09.10.2019) – Назва з екрана.
- 16 Quickstart, Flask Documentation [Електронний ресурс] : [Web-сайт] – Електронні дані – Режим доступу: <http://flask.palletsprojects.com/en/1.1.x/quickstart/#quickstart> (дата звернення 09.10.2019) – Назва з екрана.
- 17 Design Patterns : elements of reusable object-oriented software / Erich Gamma ... [et al.]. p. cm.—(Addison-Wesley professional computing series) Includes bibliographical references and index. ISBN 0-201-63361-2 1. Object-oriented programming (Computer science) 2. Computer software—Reusability. I. Gamma, Erich. II. Series. QA76.64.D4 7 1994 005.1'2-dc2 0 94-3426 4 CIP

- 18 Розроблення стартап-проекту [Текст] : Методичні рекомендації до виконання розділу магістерських дисертацій для студентів інженерних спеціальностей / За заг. ред. О.А. Гавриша. - Київ : НТУУ «КПІ», 2016. - 28 с.

ДОДАТКИ

ДОДАТОК А

У даному додатку подані правила виявлення шкідливих конструкцій.

grammar common;

// lexer rules

fragment A : [Aa] | '%' [46] '1' | '\\1' [04] '1' | '\\x' [46] '1' | '\\u00' [46] '1' | '&#' '0' '0' '0' '0' '0' ('65' | '97') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '1' ';' ;

fragment B : [Bb] | '%' [46] '2' | '\\1' [04] '2' | '\\x' [46] '2' | '\\u00' [46] '2' | '&#' '0' '0' '0' '0' '0' ('66' | '98') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '2' ';' ;

fragment C : [Cc] | '%' [46] '3' | '\\1' [04] '3' | '\\x' [46] '3' | '\\u00' [46] '3' | '&#' '0' '0' '0' '0' '0' ('67' | '99') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '3' ';' ;

fragment D : [Dd] | '%' [46] '4' | '\\1' [04] '4' | '\\x' [46] '4' | '\\u00' [46] '4' | '&#' '0' '0' '0' '0' '0' ('68' | '100') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '4' ';' ;

fragment E : [Ee] | '%' [46] '5' | '\\1' [04] '5' | '\\x' [46] '5' | '\\u00' [46] '5' | '&#' '0' '0' '0' '0' '0' ('69' | '101') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '5' ';' ;

fragment F : [Ff] | '%' [46] '6' | '\\1' [04] '6' | '\\x' [46] '6' | '\\u00' [46] '6' | '&#' '0' '0' '0' '0' '0' ('70' | '102') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '6' ';' ;

fragment G : [Gg] | '%' [46] '7' | '\\1' [04] '7' | '\\x' [46] '7' | '\\u00' [46] '7' | '&#' '0' '0' '0' '0' '0' ('71' | '103') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '7' ';' ;

fragment H : [Hh] | '%' [46] '8' | '\\1' [15] '0' | '\\x' [46] '8' | '\\u00' [46] '8' | '&#' '0' '0' '0' '0' '0' ('72' | '104') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '8' ';' ;

fragment I : [Ii] | '%' [46] '9' | '\\1' [15] '1' | '\\x' [46] '9' | '\\u00' [46] '9' | '&#' '0' '0' '0' '0' '0' ('73' | '105') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] '9' ';' ;

fragment J : [Jj] | '%' [46] [Aa] | '\\1' [15] '2' | '\\x' [46] [Aa] | '\\u00' [46] [Aa] | '&#' '0' '0' '0' '0' '0' ('74' | '106') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] [Aa] ';' ;

fragment K : [Kk] | '%' [46] [Bb] | '\\1' [15] '3' | '\\x' [46] [Bb] | '\\u00' [46] [Bb] | '&#' '0' '0' '0' '0' '0' ('75' | '107') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] [Bb] ';' ;

fragment L : [Ll] | '%' [46] [Cc] | '\\1' [15] '4' | '\\x' [46] [Cc] | '\\u00' [46] [Cc] | '&#' '0' '0' '0' '0' '0' ('76' | '108') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] [Cc] ';' ;

fragment M : [Mm] | '%' [46] [Dd] | '\\1' [15] '5' | '\\x' [46] [Dd] | '\\u00' [46] [Dd] | '&#' '0' '0' '0' '0' '0' ('77' | '109') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] [Dd] ';' ;

fragment N : [Nn] | '%' [46] [Ee] | '\\1' [15] '6' | '\\x' [46] [Ee] | '\\u00' [46] [Ee] | '&#' '0' '0' '0' '0' '0' ('78' | '110') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] [Ee] ';' ;

fragment O : [Oo] | '%' [46] [Ff] | '\\1' [15] '7' | '\\x' [46] [Ff] | '\\u00' [46] [Ff] | '&#' '0' '0' '0' '0' '0' ('79' | '111') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [46] [Ff] ';' ;

fragment P : [Pp] | '%' [57] '0' | '\\1' [26] '0' | '\\x' [57] '0' | '\\u00' [57] '0' | '&#' '0' '0' '0' '0' '0' ('80' | '112') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [57] '0' ';' ;

fragment Q : [Qq] | '%' [57] '1' | '\\1' [26] '1' | '\\x' [57] '1' | '\\u00' [57] '1' | '&#' '0' '0' '0' '0' '0' ('81' | '113') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [57] '1' ';' ;

fragment R : [Rr] | '%' [57] '2' | '\\1' [26] '2' | '\\x' [57] '2' | '\\u00' [57] '2' | '&#' '0' '0' '0' '0' '0' ('82' | '114') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [57] '2' ';' ;

fragment S : [Ss] | '%' [57] '3' | '\\1' [26] '3' | '\\x' [57] '3' | '\\u00' [57] '3' | '&#' '0' '0' '0' '0' '0' ('83' | '115') ';' | '&#' [Xx] '0' '0' '0' '0' '0' [57] '3' ';' ;


```

// boolean_sqli
boolean_sqli : (OR|AND) (SEPARATOR|CUTTABLE|SQLKEY)+ pred_part ;
pred_part : comp | NUMBER | WORD ;

comp:          comp_part          (SEPARATOR|CUTTABLE|SQLKEY)*          COMP_SIGN
(SEPARATOR|CUTTABLE|SQLKEY)* comp_part ;

comp_part: NUMBER | sql_func | L_PAREN (SEPARATOR|CUTTABLE|SQLKEY)* select_statement .*?
R_PAREN | WORD ;

sql_func : SQL_FUNC (SEPARATOR|CUTTABLE|SQLKEY)* L_PAREN .*? R_PAREN ;

// ! end of boolean_sqli

// timebased_sqli
timebased_sqli : waitfor_delay | benchmark_sqli | sleep_sqli | pg_sleep_sqli | gen_series_sqli ;

waitfor_delay:          WAITFOR          (SEPARATOR|CUTTABLE|SQLKEY)+          DELAY
(SEPARATOR|CUTTABLE|SQLKEY)+ TIMESTRING ;

benchmark_sqli :          BENCHMARK          (SEPARATOR|CUTTABLE|SQLKEY)*          L_PAREN
(SEPARATOR|CUTTABLE|SQLKEY)* NUMBER (SEPARATOR|CUTTABLE|SQLKEY)* COMMA .+? R_PAREN ;

sleep_sqli: SLEEP (SEPARATOR|CUTTABLE|SQLKEY)* L_PAREN (SEPARATOR|CUTTABLE|SQLKEY)*
NUMBER (SEPARATOR|CUTTABLE|SQLKEY)* R_PAREN;

pg_sleep_sqli :          PG_SLEEP          (SEPARATOR|CUTTABLE|SQLKEY)*          L_PAREN
(SEPARATOR|CUTTABLE|SQLKEY)* NUMBER (SEPARATOR|CUTTABLE|SQLKEY)* R_PAREN;

gen_series_sqli :          GENERATE_SERIES          (SEPARATOR|CUTTABLE|SQLKEY)*          L_PAREN
(SEPARATOR|CUTTABLE|SQLKEY)* (NUMBER | WORD) (SEPARATOR|CUTTABLE|SQLKEY)*          COMMA
(SEPARATOR|CUTTABLE|SQLKEY)* (NUMBER | WORD) (COMMA (SEPARATOR|CUTTABLE|SQLKEY)*
(NUMBER | WORD))? (SEPARATOR|CUTTABLE|SQLKEY)* R_PAREN ;

// ! end of timebased_sqli

stacked_query : SEMICOLON (SEPARATOR|CUTTABLE|SQLKEY)* sql_statement | L_PAREN
(SEPARATOR|CUTTABLE|SQLKEY)* sql_statement .*? R_PAREN;

sql_statement : select_statement | create_table | alter_table | drop_table | insert_into |
delete_from | update_statement ;

select_statement :          SELECT          (SEPARATOR|CUTTABLE|SQLKEY)+          (select_part)
((SEPARATOR|CUTTABLE|SQLKEY)* COMMA (SEPARATOR|CUTTABLE|SQLKEY)* select_part)* ; // SELECT
( CASE WHEN ...

create_table : CREATE (SEPARATOR|CUTTABLE|SQLKEY)+ TABLE ;

alter_table : ALTER (SEPARATOR|CUTTABLE|SQLKEY)+ TABLE ;

drop_table : DROP (SEPARATOR|CUTTABLE|SQLKEY)+ TABLE ;

insert_into : INSERT (SEPARATOR|CUTTABLE|SQLKEY)+ INTO ;

delete_from : DELETE (SEPARATOR|CUTTABLE|SQLKEY)+ FROM ;

update_statement:          UPDATE          (SEPARATOR|CUTTABLE|SQLKEY)+          WORD
(SEPARATOR|CUTTABLE|SQLKEY)+ SET ;

select_part : NUMBER | sql_func | WORD ;

// ! end of parser rules

// lexer rules

```

fragment SE : S (CUTTABLE|SQLKEY)* E ;
 fragment RSE : R (CUTTABLE|SQLKEY)* SE ;
 fragment ERSE : E (CUTTABLE|SQLKEY)* RSE ;
 fragment VERSE: V (CUTTABLE|SQLKEY)* ERSE ;
 fragment EVERSE : E (CUTTABLE|SQLKEY)* VERSE ;
 REVERSE : R (CUTTABLE|SQLKEY)* EVERSE ;
 fragment TE : T (CUTTABLE|SQLKEY)* E ;
 fragment ATE : A (CUTTABLE|SQLKEY)* TE ;
 fragment EATE : E (CUTTABLE|SQLKEY)* ATE ;
 fragment REATE: R (CUTTABLE|SQLKEY)* EATE ;
 CREATE : C (CUTTABLE|SQLKEY)* REATE ;
 fragment LE : L (CUTTABLE|SQLKEY)* E ;
 fragment BLE : B (CUTTABLE|SQLKEY)* LE ;
 fragment ABLE : A (CUTTABLE|SQLKEY)* BLE ;
 TABLE : T (CUTTABLE|SQLKEY)* ABLE ;
 fragment ER : E (CUTTABLE|SQLKEY)* R ;
 fragment TER : T (CUTTABLE|SQLKEY)* ER ;
 fragment LTER : L (CUTTABLE|SQLKEY)* TER ;
 ALTER : A (CUTTABLE|SQLKEY)* LTER ;
 fragment OP : O (CUTTABLE|SQLKEY)* P ;
 fragment ROP : R (CUTTABLE|SQLKEY)* OP ;
 DROP : D (CUTTABLE|SQLKEY)* ROP ;
 fragment RT : R (CUTTABLE|SQLKEY)* T ;
 fragment ERT : E (CUTTABLE|SQLKEY)* RT ;
 fragment SERT : S (CUTTABLE|SQLKEY)* ERT ;
 fragment NSERT : N (CUTTABLE|SQLKEY)* SERT ;
 INSERT : I (CUTTABLE|SQLKEY)* NSERT ;
 fragment TO : T (CUTTABLE|SQLKEY)* O ;
 fragment NTO : N (CUTTABLE|SQLKEY)* TO ;
 INTO : I (CUTTABLE|SQLKEY)* NTO ;
 fragment ETE : E (CUTTABLE|SQLKEY)* TE ;
 fragment LETE : L (CUTTABLE|SQLKEY)* ETE ;
 fragment ELETE : E (CUTTABLE|SQLKEY)* LETE ;
 DELETE : D (CUTTABLE|SQLKEY)* ELETE ;
 fragment OM : O (CUTTABLE|SQLKEY)* M ;
 fragment ROM : R (CUTTABLE|SQLKEY)* OM ;

```

FROM          : F (CUTTABLE|SQLKEY)* ROM ;
fragment DATE : D (CUTTABLE|SQLKEY)* ATE ;
fragment PDATE      : P (CUTTABLE|SQLKEY)* DATE ;
UPDATE          : U (CUTTABLE|SQLKEY)* PDATE ;
fragment ET        : E (CUTTABLE|SQLKEY)* T ;
SET             : S (CUTTABLE|SQLKEY)* ET ;
// union_select
fragment NU       : N (CUTTABLE|SQLKEY)* U ;
fragment INU      : I (CUTTABLE|SQLKEY)* NU ;
fragment OINU     : O (CUTTABLE|SQLKEY)* INU ;
NOINU           : N (CUTTABLE|SQLKEY)* OINU ;
fragment ON       : O (CUTTABLE|SQLKEY)* N ;
fragment ION      : I (CUTTABLE|SQLKEY)* ON ;
fragment NION     : N (CUTTABLE|SQLKEY)* ION ;
UNION           : U (CUTTABLE|SQLKEY)* NION | REVERSE
(CUTTABLE|SQLKEY|SEPARATOR)* L_PAREN (CUTTABLE|SQLKEY|SEPARATOR)* NOINU
(CUTTABLE|SQLKEY|SEPARATOR)* R_PAREN;
fragment ES       : E (CUTTABLE|SQLKEY)* S ;
fragment LES      : L (CUTTABLE|SQLKEY)* ES ;
fragment ELES     : E (CUTTABLE|SQLKEY)* LES ;
fragment CELES    : C (CUTTABLE|SQLKEY)* ELES ;
TCELES          : T (CUTTABLE|SQLKEY)* CELES ;
fragment CT       : C (CUTTABLE|SQLKEY)* T ;
fragment ECT      : E (CUTTABLE|SQLKEY)* CT ;
fragment LECT     : L (CUTTABLE|SQLKEY)* ECT ;
fragment ELECT    : E (CUTTABLE|SQLKEY)* LECT ;
SELECT          : S (CUTTABLE|SQLKEY)* ELECT | REVERSE (CUTTABLE|SQLKEY|SEPARATOR)*
L_PAREN (CUTTABLE|SQLKEY|SEPARATOR)* TCELES (CUTTABLE|SQLKEY|SEPARATOR)* ')';
fragment ID       : I (CUTTABLE|SQLKEY)* D ;
fragment SID      : S (CUTTABLE|SQLKEY)* ID ;
fragment TSID     : T (CUTTABLE|SQLKEY)* SID ;
fragment ITSID    : I (CUTTABLE|SQLKEY)* TSID ;
fragment NITSID   : N (CUTTABLE|SQLKEY)* ITSID ;
fragment CNITSID  : C (CUTTABLE|SQLKEY)* NITSID ;
TCNITSID         : T (CUTTABLE|SQLKEY)* CNITSID ;
fragment NCT      : N (CUTTABLE|SQLKEY)* CT ;
fragment INCT     : I (CUTTABLE|SQLKEY)* NCT ;

```

```

fragment TINCT : T (CUTTABLE|SQLKEY)* INCT ;

fragment STINCT      : S (CUTTABLE|SQLKEY)* TINCT ;

fragment ISTINCT     : I (CUTTABLE|SQLKEY)* STINCT;

DISTINCT      :      D      (CUTTABLE|SQLKEY)*      ISTINCT      |      REVERSE
(CUTTABLE|SQLKEY|SEPARATOR)*      L_PAREN      (CUTTABLE|SQLKEY|SEPARATOR)*      TCNITSID
(CUTTABLE|SQLKEY|SEPARATOR)* ');

fragment LL      : L (CUTTABLE|SQLKEY)* L ;

fragment LLA     : LL (CUTTABLE|SQLKEY)* A ;

ALL      : A (CUTTABLE|SQLKEY)* LL | REVERSE (CUTTABLE|SQLKEY|SEPARATOR)* L_PAREN
(CUTTABLE|SQLKEY|SEPARATOR)* LLA (CUTTABLE|SQLKEY|SEPARATOR)* ');

// ! end of union_select

// boolean_sqli

COMP_SIGN      : GT_SIGN | LT_SIGN | EQ_SIGN | EXCL_SIGN EQ_SIGN      |
LT_SIGN GT_SIGN | LT_SIGN EQ_SIGN | GT_SIGN EQ_SIGN ;

fragment RO      : R (CUTTABLE|SQLKEY)* O ;

OR      :      O      (CUTTABLE|SQLKEY)*      R      |      REVERSE
(CUTTABLE|SQLKEY|SEPARATOR)*      L_PAREN      (CUTTABLE|SQLKEY|SEPARATOR)*      RO
(CUTTABLE|SQLKEY|SEPARATOR)* ');

fragment NA      : N (CUTTABLE|SQLKEY)* A ;

fragment DNA     : D (CUTTABLE|SQLKEY)* NA ;

fragment ND      : N (CUTTABLE|SQLKEY)* D ;

AND      :      A      (CUTTABLE|SQLKEY)*      ND      |      REVERSE
(CUTTABLE|SQLKEY|SEPARATOR)*      L_PAREN      (CUTTABLE|SQLKEY|SEPARATOR)*      DNA
(CUTTABLE|SQLKEY|SEPARATOR)* ');

fragment II      : I (CUTTABLE|SQLKEY)* I ;

fragment CII     : C (CUTTABLE|SQLKEY)* II ;

fragment SCII    : S (CUTTABLE|SQLKEY)* CII ;

ASCII         : A (CUTTABLE|SQLKEY)* SCII ;

fragment AT      : A (CUTTABLE|SQLKEY)* T ;

fragment CAT     : C (CUTTABLE|SQLKEY)* AT ;

fragment NCAT : N (CUTTABLE|SQLKEY)* CAT ;

fragment ONCAT   : O (CUTTABLE|SQLKEY)* NCAT ;

CONCAT          : C (CUTTABLE|SQLKEY)* ONCAT ;

fragment NG      : N (CUTTABLE|SQLKEY)* G ;

fragment ING     : I (CUTTABLE|SQLKEY)* NG ;

fragment RING    : R (CUTTABLE|SQLKEY)* ING ;

fragment TRING   : T (CUTTABLE|SQLKEY)* RING ;

fragment STRING  : S (CUTTABLE|SQLKEY)* TRING ;

fragment BSTRING : B (CUTTABLE|SQLKEY)* STRING ;

```



```

fragment UBSTRING : U (CUTTABLE|SQLKEY)* BSTRING ;
SUBSTRING          : S (CUTTABLE|SQLKEY)* UBSTRING ;
fragment SION       : S (CUTTABLE|SQLKEY)* ION ;
fragment RSION      : R (CUTTABLE|SQLKEY)* SION ;
fragment ERSION     : E (CUTTABLE|SQLKEY)* RSION ;
VERSION            : V (CUTTABLE|SQLKEY)* ERSION ;
fragment SER        : S (CUTTABLE|SQLKEY)* ER ;
USER               : U (CUTTABLE|SQLKEY)* SER ;
fragment EX         : E (CUTTABLE|SQLKEY)* X ;
HEX                : H (CUTTABLE|SQLKEY)* EX ;
fragment NHEX       : N (CUTTABLE|SQLKEY)* HEX ;
UNHEX              : U (CUTTABLE|SQLKEY)* NHEX ;
fragment AX         : A (CUTTABLE|SQLKEY)* X ;
MAX                : M (CUTTABLE|SQLKEY)* AX ;
fragment IN         : I (CUTTABLE|SQLKEY)* N ;
MIN                : M (CUTTABLE|SQLKEY)* IN ;
fragment VG         : V (CUTTABLE|SQLKEY)* G ;
AVG                : A (CUTTABLE|SQLKEY)* VG ;
fragment UM         : U (CUTTABLE|SQLKEY)* M ;
SUM                : S (CUTTABLE|SQLKEY)* UM ;

SQL_FUNC : ASCII | CONCAT | SUBSTRING | VERSION | HEX | UNHEX | MIN | MAX |
AVG | SUM ;

// ! end of boolean_sqli
// timebased_sqli

fragment FOR        : F (CUTTABLE|SQLKEY)* OR ;
fragment TFOR       : T (CUTTABLE|SQLKEY)* FOR ;
fragment ITFOR      : I (CUTTABLE|SQLKEY)* TFOR ;
fragment AITFOR     : A (CUTTABLE|SQLKEY)* ITFOR ;
WAITFOR            : W (CUTTABLE|SQLKEY)* AITFOR ;
fragment AY         : A (CUTTABLE|SQLKEY)* Y ;
fragment LAY        : L (CUTTABLE|SQLKEY)* AY ;
fragment ELAY       : E (CUTTABLE|SQLKEY)* LAY ;
DELAY              : D (CUTTABLE|SQLKEY)* ELAY ;
fragment RK         : R (CUTTABLE|SQLKEY)* K ;
fragment ARK        : A (CUTTABLE|SQLKEY)* RK ;
fragment MARK       : M (CUTTABLE|SQLKEY)* ARK ;

```

```

fragment HMARK          : H (CUTTABLE|SQLKEY)* MARK          ;
fragment CHMARK         : C (CUTTABLE|SQLKEY)* HMARK        ;
fragment NCHMARK        : N (CUTTABLE|SQLKEY)* CHMARK        ;
fragment ENCHMARK       : E (CUTTABLE|SQLKEY)* NCHMARK       ;
BENCHMARK              : B (CUTTABLE|SQLKEY)* ENCHMARK      ;
fragment EP             : E (CUTTABLE|SQLKEY)* P            ;
fragment EEP            : E (CUTTABLE|SQLKEY)* EP           ;
fragment LEEP           : L (CUTTABLE|SQLKEY)* EEP          ;
SLEEP                  : S (CUTTABLE|SQLKEY)* LEEP          ;
fragment G_SLEEP       : G (CUTTABLE|SQLKEY)* '_' (CUTTABLE|SQLKEY)* SLEEP ;
PG_SLEEP               : P (CUTTABLE|SQLKEY)* G_SLEEP       ;
fragment IES           : I (CUTTABLE|SQLKEY)* ES            ;
fragment RIES          : R (CUTTABLE|SQLKEY)* IES           ;
fragment ERIES         : E (CUTTABLE|SQLKEY)* RIES          ;
fragment SERIES        : S (CUTTABLE|SQLKEY)* ERIES         ;
fragment E_SERIES      : E (CUTTABLE|SQLKEY)* '_' (CUTTABLE|SQLKEY)* SERIES ;
fragment TE_SERIES     : T (CUTTABLE|SQLKEY)* E_SERIES      ;
fragment ATE_SERIES    : A (CUTTABLE|SQLKEY)* TE_SERIES     ;
fragment RATE_SERIES   : R (CUTTABLE|SQLKEY)* ATE_SERIES    ;
fragment ERATE_SERIES  : E (CUTTABLE|SQLKEY)* RATE_SERIES   ;
fragment NERATE_SERIES : N (CUTTABLE|SQLKEY)* ERATE_SERIES  ;
fragment ENERATE_SERIES : E (CUTTABLE|SQLKEY)* NERATE_SERIES ;
GENERATE_SERIES        : G (CUTTABLE|SQLKEY)* ENERATE_SERIES ;

// ! end of timebased_sqli

SQLKEY      : UNION      | SELECT | WAITFOR | DELAY | BENCHMARK | SLEEP | DROP      |
              INSERT | UPDATE  | DELETE;

// ! end of lexer rules

grammar xss;
import common;
// parser rules

xss_rules : .*? ( script_tag_statement | script_statement | console_statement | popup_statement
                | to_string_usage | dom_obj_injection | js_function | find_statement ) .*? ;

script_tag_statement : LT_SIGN CUTTABLE* SCRIPT .*? GT_SIGN? | LT_SIGN CUTTABLE* SLASH
CUTTABLE* SCRIPT CUTTABLE* GT_SIGN ;

script_statement      :                script_word                CUTTABLE*                COLON
CUTTABLE*(console_statement|popup_statement) ;

```

```

script_word      : JAVASCRIPT | ECMASCRIPT | TYPESCRIPT | VBSCRIPT | ACTIONSCRIPT;

console_statement:      CONSOLE      CUTTABLE*      POINT      CUTTABLE*
(LOG | ERROR | ASSERT | GROUP | WARN | INFO | TABLE) (CUTTABLE | R_PAREN)* L_PAREN .*? R_PAREN ;

popup_statement : popup_word (CUTTABLE | R_PAREN)* L_PAREN .*? R_PAREN ;

popup_word      : ALERT | CONFIRM | PROMPT | MSGBOX;

to_string_usage : POINT CUTTABLE* TOSTRING CUTTABLE* L_PAREN (SEPARATOR | CUTTABLE)*
NUMBER (SEPARATOR | CUTTABLE)* R_PAREN ;

dom_obj_injection: (THIS | DOCUMENT | WINDOW | TOP) CUTTABLE* L_BRACK .*? R_BRACK |
(THIS | DOCUMENT | WINDOW | TOP) CUTTABLE* POINT CUTTABLE*
(NAME | LOCATION | OPEN | COOKIE | DOMAIN | WRITE | HOSTNAME | GET | CREATE) ;

js_function      : (EVAL | SETINTERVAL | SETTIMEOUT) CUTTABLE* L_PAREN .*? R_PAREN
| PARSEINT CUTTABLE* L_PAREN CUTTABLE* WORD CUTTABLE* COMMA CUTTABLE* NUMBER CUTTABLE*
R_PAREN ;

find_statement   : L_BRACK .*? R_BRACK CUTTABLE* FIND CUTTABLE* L_PAREN .*? R_PAREN ;

// ! end of parser rules

// lexer rules

fragment PT      : P CUTTABLE* T      ;

fragment IPT     : I CUTTABLE* PT     ;

fragment RIPT    : R CUTTABLE* IPT    ;

fragment CRIPT   : C CUTTABLE* RIPT   ;

SCRIPT          : S CUTTABLE* CRIPT   ;

fragment LE      : L CUTTABLE* E      ;

fragment OLE     : O CUTTABLE* LE     ;

fragment SOLE    : S CUTTABLE* OLE    ;

fragment NSOLE   : N CUTTABLE* SOLE   ;

fragment ONSOLE  : O CUTTABLE* NSOLE  ;

CONSOLE         : C CUTTABLE* ONSOLE  ;

fragment OG      : O CUTTABLE* G      ;

LOG             : L CUTTABLE* OG      ;

fragment RR      : R CUTTABLE* R      ;

fragment ORR     : O CUTTABLE* RR     ;

fragment RORR    : R CUTTABLE* ORR    ;

ERROR           : E CUTTABLE* RORR    ;

fragment RT      : R CUTTABLE* T      ;

fragment ERT     : E CUTTABLE* RT     ;

fragment SERT    : S CUTTABLE* ERT    ;

fragment SSERT   : S CUTTABLE* SERT   ;

ASSERT          : A CUTTABLE* SSERT   ;

```

fragment UP : U CUTTABLE* P ;
 fragment OUP : O CUTTABLE* UP ;
 fragment ROUP : R CUTTABLE* OUP ;
 GROUP : G CUTTABLE* ROUP ;
 fragment RN : R CUTTABLE* N ;
 fragment ARN : A CUTTABLE* RN ;
 WARN : W CUTTABLE* ARN ;
 fragment FO : F CUTTABLE* O ;
 fragment NFO : N CUTTABLE* FO ;
 INFO : I CUTTABLE* NFO ;
 fragment BLE : B CUTTABLE* LE ;
 fragment ABLE : A CUTTABLE* BLE ;
 TABLE : T CUTTABLE* ABLE ;
 fragment ASCRIPT : A CUTTABLE* SCRIPT ;
 fragment VASCRIP : V CUTTABLE* ASCRIP ;
 fragment AVASCRIP : A CUTTABLE* VASCRIP ;
 JAVASCRIP : J CUTTABLE* AVASCRIP ;
 fragment MASCRIP : M CUTTABLE* ASCRIP ;
 fragment CMASCRIP : C CUTTABLE* MASCRIP ;
 ECMASCRIP : E CUTTABLE* CMASCRIP ;
 fragment ESCRIP : E CUTTABLE* SCRIP ;
 fragment PESCRIP : P CUTTABLE* ESCRIP ;
 fragment YPESCRIP : Y CUTTABLE* PESCRIP ;
 TYPESCRIP : T CUTTABLE* YPESCRIP ;
 fragment BSCRIP : B CUTTABLE* SCRIP ;
 VBSCRIP : V CUTTABLE* BSCRIP ;
 fragment NSCRIP : N CUTTABLE* SCRIP ;
 fragment ONSCRIP : O CUTTABLE* NSCRIP ;
 fragment IONSCRIP : I CUTTABLE* ONSCRIP ;
 fragment TIONSCRIP : T CUTTABLE* IONSCRIP ;
 fragment CTIONSCRIP : C CUTTABLE* TIONSCRIP ;
 ACTIONSCRIP : A CUTTABLE* CTIONSCRIP ;
 fragment LERT : L CUTTABLE* ERT ;
 ALERT : A CUTTABLE* LERT ;
 fragment MPT : M CUTTABLE* PT ;
 fragment OMPT : O CUTTABLE* MPT ;

```

fragment ROMPT      : R CUTTABLE* OMPT ;
PROMPT              : P CUTTABLE* ROMPT ;
fragment RM         : R CUTTABLE* M   ;
fragment IRM        : I CUTTABLE* RM  ;
fragment FIRM       : F CUTTABLE* IRM  ;
fragment NFIRM      : N CUTTABLE* FIRM ;
fragment ONFIRM     : O CUTTABLE* NFIRM ;
CONFIRM            : C CUTTABLE* ONFIRM ;
fragment UT         : U CUTTABLE* T    ;
fragment OUT        : O CUTTABLE* UT   ;
fragment EOUT       : E CUTTABLE* OUT  ;
fragment MEOUT      : M CUTTABLE* EOUT ;
fragment IMEOUT     : I CUTTABLE* MEOUT ;
TIMEOUT            : T CUTTABLE* IMEOUT ;
fragment TIMEOUT    : T CUTTABLE* TIMEOUT ;
fragment ETIMEOUT   : E CUTTABLE* TIMEOUT ;
SETTIMEOUT         : S CUTTABLE* ETIMEOUT ;
fragment AL         : A CUTTABLE* L    ;
fragment VAL        : V CUTTABLE* AL   ;
fragment RVAL       : R CUTTABLE* VAL  ;
fragment ERVAL      : E CUTTABLE* RVAL  ;
fragment TERVAL     : T CUTTABLE* ERVAL ;
fragment NTERVAL    : N CUTTABLE* TERVAL ;
INTERVAL           : I CUTTABLE* INTERVAL ;
fragment TINTERVAL  : T CUTTABLE* INTERVAL ;
fragment ETINTERVAL : E CUTTABLE* TINTERVAL ;
SETINTERVAL        : S CUTTABLE* ETINTERVAL ;
EVAL : E CUTTABLE* VAL ;
fragment INT      : I CUTTABLE* NT    ;
fragment EINT     : E CUTTABLE* INT    ;
fragment SEINT    : S CUTTABLE* EINT   ;
fragment RSEINT   : R CUTTABLE* SEINT  ;
fragment ARSEINT  : A CUTTABLE* RSEINT ;
PARSEINT         : P CUTTABLE* ARSEINT ;
fragment NG       : N CUTTABLE* G      ;
fragment ING      : I CUTTABLE* NG     ;

```

```

fragment RING      : R CUTTABLE* ING  ;
fragment TRING     : T CUTTABLE* RING ;
fragment STRING    : S CUTTABLE* TRING ;
fragment OSTRING   : O CUTTABLE* STRING ;
TOSTRING          : T CUTTABLE* OSTRING ;
fragment OX       : O CUTTABLE* X    ;
fragment BOX      : B CUTTABLE* OX   ;
fragment GBOX     : G CUTTABLE* BOX  ;
fragment SGBOX    : S CUTTABLE* GBOX ;
MSGBOX           : M CUTTABLE* SGBOX ;
fragment OW       : O CUTTABLE* W    ;
fragment DOW      : D CUTTABLE* OW   ;
fragment NDOW     : N CUTTABLE* DOW  ;
fragment INDOW    : I CUTTABLE* NDOW ;
WINDOW           : W CUTTABLE* INDOW ;
fragment NT       : N CUTTABLE* T    ;
fragment ENT      : E CUTTABLE* NT   ;
fragment MENT     : M CUTTABLE* ENT  ;
fragment UMENT    : U CUTTABLE* MENT ;
fragment CUMENT   : C CUTTABLE* UMENT ;
fragment OCUMENT  : O CUTTABLE* CUMENT ;
DOCUMENT         : D CUTTABLE* OCUMENT ;
fragment OP       : O CUTTABLE* P    ;
TOP              : T CUTTABLE* OP    ;
fragment IS       : I CUTTABLE* S    ;
fragment HIS      : H CUTTABLE* IS   ;
THIS             : T CUTTABLE* HIS   ;
fragment ME       : M CUTTABLE* E    ;
fragment AME      : A CUTTABLE* ME   ;
NAME            : N CUTTABLE* AME    ;
fragment ON       : O CUTTABLE* N    ;
fragment ION      : I CUTTABLE* ON   ;
fragment TION     : T CUTTABLE* ION  ;
fragment ATION    : A CUTTABLE* TION ;
fragment CATION   : C CUTTABLE* ATION ;
fragment OCATION  : O CUTTABLE* CATION ;

```

```

LOCATION          : L CUTTABLE* OCATION ;
fragment EN     : E CUTTABLE* N  ;
fragment PEN    : P CUTTABLE* EN  ;
OPEN           : O CUTTABLE* PEN  ;
fragment IE     : I CUTTABLE* E   ;
fragment KIE    : K CUTTABLE* IE  ;
fragment OKIE   : O CUTTABLE* KIE ;
fragment OOKIE  : O CUTTABLE* OKIE ;
COOKIE         : C CUTTABLE* OOKIE ;
fragment IN     : I CUTTABLE* N   ;
fragment AIN    : A CUTTABLE* IN  ;
fragment MAIN   : M CUTTABLE* AIN ;
fragment OMAIN  : O CUTTABLE* MAIN ;
DOMAIN         : D CUTTABLE* OMAIN ;
fragment TE     : T CUTTABLE* E   ;
fragment ITE    : I CUTTABLE* TE  ;
fragment RITE   : R CUTTABLE* ITE ;
WRITE          : W CUTTABLE* RITE ;
fragment TNAME  : T CUTTABLE* NAME ;
fragment STNAME : S CUTTABLE* TNAME ;
fragment OSTNAME : O CUTTABLE* STNAME ;
HOSTNAME       : H CUTTABLE* OSTNAME ;
fragment ET     : E CUTTABLE* T   ;
GET            : G CUTTABLE* ET   ;
fragment ND     : N CUTTABLE* D   ;
fragment IND    : I CUTTABLE* ND  ;
FIND           : F CUTTABLE* IND  ;
fragment ATE    : A CUTTABLE* TE  ;
fragment EATE   : E CUTTABLE* ATE ;
fragment REATE  : R CUTTABLE* EATE ;
CREATE         : C CUTTABLE* REATE ;
// ! end of lexer rules

grammar xxe;
import common;
// parser rules

```

```

x xe_rules : .*? x xe .*? ;

x xe : LT_SIGN (SEPARATOR|CUTTABLE)* EXCL_SIGN (SEPARATOR|CUTTABLE)* ENTITY
(SEPARATOR|CUTTABLE)+ WORD (SEPARATOR|CUTTABLE)+ SYSTEM (SEPARATOR|CUTTABLE)+
PROTOCOL CUTTABLE* COLON CUTTABLE* SLASH ;

// ! end of parser rules

// lexer rules
fragment TY : T CUTTABLE* Y ;
fragment ITY : I CUTTABLE* TY ;
fragment TITY : T CUTTABLE* ITY ;
fragment NTITY : N CUTTABLE* TITY ;
ENTITY : E CUTTABLE* NTITY ;
fragment EM : E CUTTABLE* M ;
fragment TEM : T CUTTABLE* EM ;
fragment STEM : S CUTTABLE* TEM ;
fragment YSTEM : Y CUTTABLE* STEM ;
SYSTEM : S CUTTABLE* YSTEM ;
PROTOCOL : FILE | HTTP | HTTPS | FTP ;
fragment LE : L CUTTABLE* E ;
fragment ILE : I CUTTABLE* LE ;
FILE : F CUTTABLE* ILE ;
fragment TP : T CUTTABLE* P ;
fragment TTP : T CUTTABLE* TP ;
HTTP : H CUTTABLE* TTP ;
HTTPS : HTTP CUTTABLE* S ;
FTP : F CUTTABLE* TP ;
// ! end of lexer rules

grammar html_injection;
import common;

// parser rules
html_injection : .*? ( html_tag_injection ) .*? ;

html_tag_injection : LT_SIGN CUTTABLE* HTML_TAG .*? GT_SIGN? | LT_SIGN CUTTABLE* SLASH
CUTTABLE* HTML_TAG CUTTABLE* GT_SIGN ;

// ! end of parser rules
// lexer rules

```



```

HTML_TAG : A | P | DIV | SPAN | IMG | STYLE | HTML | BODY | HEAD ;
fragment IV : I CUTTABLE* V ;
DIV          : D CUTTABLE* IV ;
fragment AN : A CUTTABLE* N ;
fragment PAN : P CUTTABLE* AN ;
SPAN        : S CUTTABLE* PAN ;
fragment MG : M CUTTABLE* G ;
IMG         : I CUTTABLE* MG ;
fragment LE : L CUTTABLE* E ;
fragment YLE : Y CUTTABLE* LE ;
fragment TYLE : T CUTTABLE* YLE ;
STYLE       : S CUTTABLE* TYLE ;
fragment ML : M CUTTABLE* L ;
fragment TML : T CUTTABLE* ML ;
HTML        : H CUTTABLE* TML ;
fragment DY : D CUTTABLE* Y ;
fragment ODY : O CUTTABLE* DY ;
BODY        : B CUTTABLE* ODY ;
fragment AD : A CUTTABLE* D ;
fragment EAD : E CUTTABLE* AD ;
HEAD        : H CUTTABLE* EAD ;

// ! end of lexer rules

```

ДОДАТОК Б

Код тестової програми:

View.py:

```
__author__ = 'ok'

from flask import render_template
from flask import request
from app import app
from models import Items
from sqlalchemy import func
from analysis import *
from datetime import datetime

user = { 'username' : 'Alex' }

@app.route('/')
@app.route('/home')
@app.route('/ads')
def index():
    file = open(app.config['LOG_FILE'], 'a+')
    now = datetime.now()
    file.write("{}\t{}\t{}\n".format(now, request.full_path, request.method))
    file.close()
    state = {'attack':False}
    for args in request.args.items():
        for arg in args:
            state = CheckArg(arg)
            if state['attack'] == True:
                break
        if state['attack'] == True:
            break

    if state['attack'] == False:
        q = request.args.get('q')
        if q:
            items = Items.query.filter(func.lower(Items.item_name).contains(q.lower())
func.lower(Items.description).contains(q.lower()))).all()
        else:
            items = Items.query.all()
```

```

else:
    file = open(app.config['ATTACK_LOG_FILE'], 'a+')
    file.write("{}\t{}\t{}\t{}\n".format(now, request.full_path, request.method, state['type']))
    file.close()
    items = Items.query.all()
    return render_template("index.html", items=items)

```

ads.py:

```

from app import app
if __name__ == '__main__':
    app.run(host='192.168.0.113')

```

app.py:

```

__author__ = 'ok'
from flask import Flask
from config import Configuration
from flask_sqlalchemy import SQLAlchemy

```

```

app = Flask(__name__)
app.config.from_object(Configuration)
db = SQLAlchemy(app)

```

config.py:

```

__author__ = 'ok'
import os
from datetime import datetime

def get_env_variable(name):
    try:
        return os.environ[name]
    except KeyError:
        message = "Expected environment variable '{}' not set.".format(name)
        raise Exception(message)

```

```

# the values of those depend on your setup
POSTGRES_URL = get_env_variable('POSTGRES_URL')
POSTGRES_USER = get_env_variable('POSTGRES_USER')
POSTGRES_DB = get_env_variable('POSTGRES_DB')
POSTGRES_PW = get_env_variable('POSTGRES_PW')

```

```

now = datetime.date(datetime.now())

```

```

class Configuration(object):

```

```

DEBUG = False
SQLALCHEMY_DATABASE_URI =
'postgresql+psycopg2://{user}:{pw}@{url}/{db}'.format(user=POSTGRES_USER,pw=POSTGRES_PW,url=
POSTGRES_URL,db=POSTGRES_DB)
SQLALCHEMY_TRACK_MODIFICATIONS = False
LOG_FILE = "D:\DyplomaLogs\logs{}.log".format(now)
ATTACK_LOG_FILE = "D:\DyplomaLogs\attacks{}.log".format(now)

```

models.py:

```

__author__ = 'ok'
from app import db

class Items(db.Model):
    item_id = db.Column(db.Integer, primary_key=True)
    item_name = db.Column(db.Text, unique=False, nullable=True)
    producer = db.Column(db.Text, unique=False, nullable=True)
    country = db.Column(db.Text, unique=False, nullable=True)
    price = db.Column(db.Float, unique=False, nullable=True)
    description = db.Column(db.Text, unique=False, nullable=True)

    def __init__(self, *args, **kwargs):
        super(Items, self).__init__(*args, **kwargs)

    def __repr__(self):
        return "<Item title: {} >".format(self.item_name)

```

analysis.py:

```

__author__ = 'ok'
from sqliParser import sqliParser
from sqliLexer import sqliLexer
from sqliListener import sqliListener
from xssParser import xssParser
from xssLexer import xssLexer
from xssListener import xssListener
from html_injectionParser import html_injectionParser
from html_injectionLexer import html_injectionLexer
from html_injectionListener import html_injectionListener
from xxeParser import xxeParser
from xxeLexer import xxeLexer
from xxeListener import xxeListener
from antlr4 import *

```

```
class SqliTestingListener(sqliListener):
```

```
    def __init__(self):
        self.state = {'attack': False}

    def exitBoolean_sqli(self, ctx:sqliParser.Boolean_sqliContext):
        self.state = {'attack' : True, 'type': 'Boolean-based SQL injection'}

    def exitStacked_query(self, ctx:sqliParser.Stacked_queryContext):
        self.state = {'attack' : True, 'type': 'Stacked query (SQL injection)'}

    def exitTimebased_sqli(self, ctx:sqliParser.Timebased_sqliContext):
        self.state = {'attack' : True, 'type': 'Time-based SQL injection'}

    def exitUnion_select(self, ctx:sqliParser.Union_selectContext):
        self.state = {'attack' : True, 'type': 'Union-based SQL injection'}
```

```
class XssTestingListener(xssListener):
```

```
    def __init__(self):
        self.state = {'attack': False}

    def exitConsole_statement(self, ctx:xssParser.Console_statementContext):
        self.state = {'attack' : True, 'type': 'XSS: Console statement inclusion'}

    def exitDom_obj_injection(self, ctx:xssParser.Dom_obj_injectionContext):
        self.state = {'attack' : True, 'type': 'XSS: DOM object (document, window etc.) inclusion'}

    def exitFind_statement(self, ctx:xssParser.Find_statementContext):
        self.state = {'attack' : True, 'type': 'XSS: "find" usage'}

    def exitJs_function(self, ctx:xssParser.Js_functionContext):
        self.state = {'attack' : True, 'type': 'XSS: usage of JS function'}

    def exitPopup_statement(self, ctx:xssParser.Popup_statementContext):
        self.state = {'attack' : True, 'type': 'XSS: usage of popup functions'}

    def exitScript_statement(self, ctx:xssParser.Script_statementContext):
        self.state = {'attack' : True, 'type': 'XSS: Usage of "JavaScript:", "VBScript:" etc + function'}

    def exitScript_tag_statement(self, ctx:xssParser.Script_tag_statementContext):
        self.state = {'attack' : True, 'type': 'XSS: script-tag injection'}

    def exitTo_string_usage(self, ctx:xssParser.To_string_usageContext):
        self.state = {'attack' : True, 'type': 'XSS: usage of "toString()"'}
```

```
class HtmlTestingListener(html_injectionListener):
```

```
    def __init__(self):
        self.state = {'attack': False}

    def exitHtml_tag_injection(self, ctx:html_injectionParser.Html_tag_injectionContext):
        self.state = {'attack' : True, 'type': 'HTML injection'}
```

```
class XxeTestingListener(xxeListener):
```

```
    def __init__(self):
```

```

        self.state = {'attack': False}
    def exitXxe(self, ctx: xxeParser.XxeContext):
        self.state = {'attack': True, 'type': 'XXE injection'}

```

```

def CheckSQLi(arg):
    input_stream = InputStream(arg)
    lexer = sqliLexer(input_stream)
    stream = CommonTokenStream(lexer)
    parser = sqliParser(stream)
    tree = parser.sqli()
    analyzer = SqliTestingListener()
    walker = ParseTreeWalker()
    walker.walk(analyzer, tree)
    return analyzer.state

```

```

def CheckXSS(arg):
    input_stream = InputStream(arg)
    lexer = xssLexer(input_stream)
    stream = CommonTokenStream(lexer)
    parser = xssParser(stream)
    tree = parser.xss_rules()
    analyzer = XssTestingListener()
    walker = ParseTreeWalker()
    walker.walk(analyzer, tree)
    return analyzer.state

```

```

def CheckHTML(arg):
    input_stream = InputStream(arg)
    lexer = html_injectionLexer(input_stream)
    stream = CommonTokenStream(lexer)
    parser = html_injectionParser(stream)
    tree = parser.html_injection()
    analyzer = HtmlTestingListener()
    walker = ParseTreeWalker()
    walker.walk(analyzer, tree)
    return analyzer.state

```

```

def CheckXXE(arg):
    input_stream = InputStream(arg)
    lexer = xxeLexer(input_stream)

```

```
stream = CommonTokenStream(lexer)
parser = xxeParser(stream)
tree = parser.xxe_rules()
analyzer = XxeTestingListener()
walker = ParseTreeWalker()
walker.walk(analyzer, tree)
return analyzer.state
```

```
def CheckArg(arg):
    state = CheckSQLi(arg)
    if state['attack'] == False:
        state = CheckXSS(arg)
    if state['attack'] == False:
        state = CheckHTML(arg)
    if state['attack'] == False:
        state = CheckXXE(arg)
    return state
```