Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

«Анализ уязвимостей веб-приложения с помощью OWASP ZAP»

по дисциплине «Информационная безопасность»

Выполнил:

Студент группы P3432

Чмурова М.В.

Преподаватель:

Рыбаков Степан Дмитриевич

Санкт-Петербург

2025

Задание

1. Установите OWASP ZAP (бесплатный инструмент).
2. Запустите встроенный браузер ZAP и перейдите на тестовый сайт (например, http://testphp.vulnweb.com/).
3. Проведите "Быстрое сканирование" (Quick Scan) сайта.
4. Проанализируйте результаты сканирования: найдите 3-5 различных типов уязвимостей (например, XSS, SQLi).
5. Сделайте скриншоты найденных уязвимостей и кратко опишите суть каждой.

Выполнение

Для выполнения был скачен OWASP ZAP на компьютер MacOS.

«Быстрое сканирование» проводилось на тестовом сайте http://testphp.vulnweb.com и с использованием браузера Chrome.

При открытии приложения открывается начальная страница:

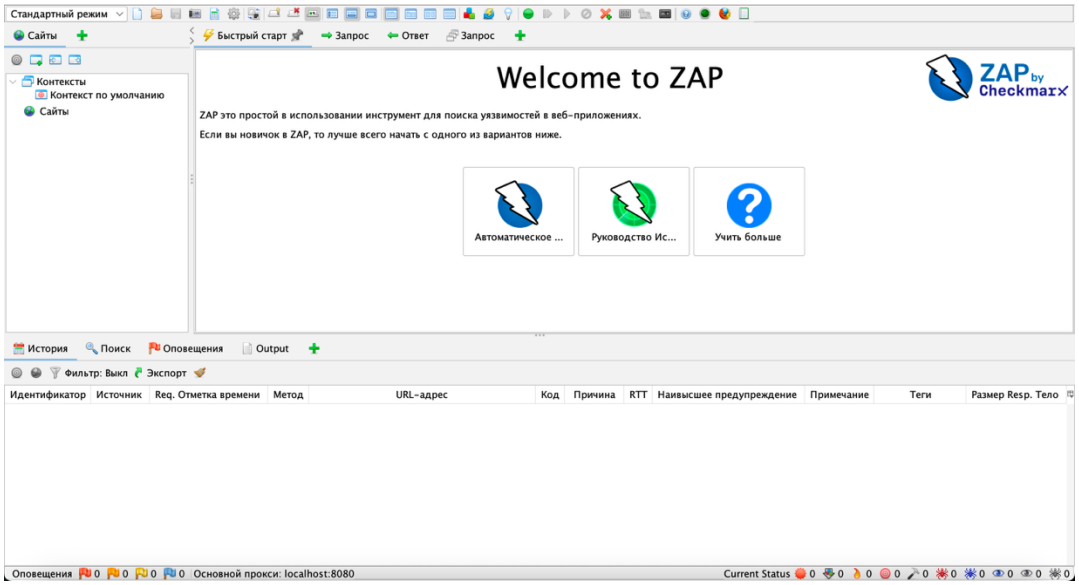

Рисунок 1. Начальный экран

Для дальнейшей работы выбирается вкладка «Быстрый старт» и выбирается режим «Автоматическое сканирование вашего приложения», которое выглядит следующим образом:

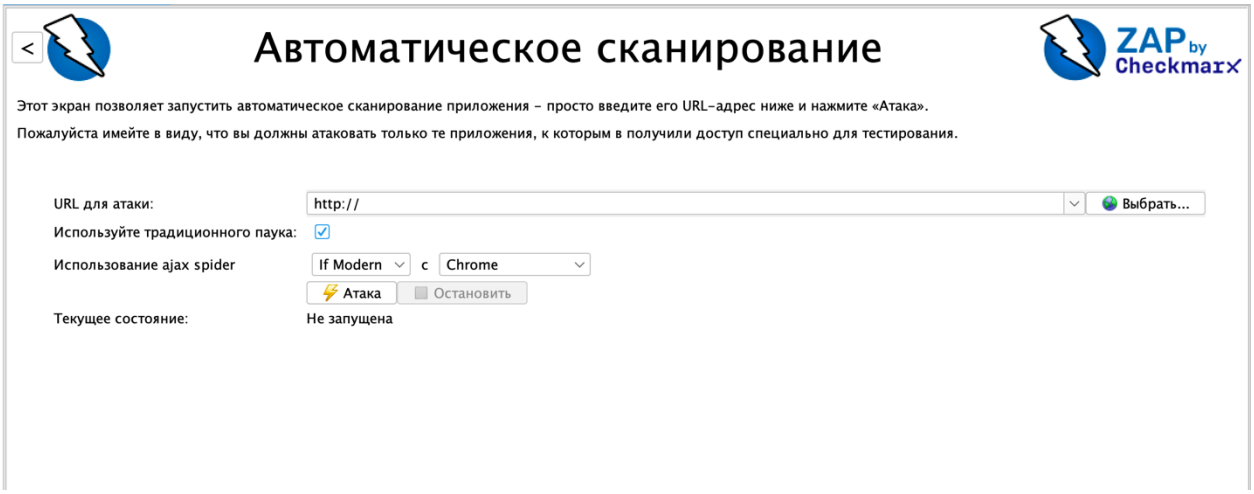

Рисунок 2. Автоматическое сканирование

Автоматическое сканирование имеет следующие поля для указания:

- URL для атаки: URL-адрес сайта, который необходимо проверить на уязвимости
- Используйте традиционного паука: использовать ли обычного паука. Паук – механизм обхода сайта. Он автоматически переходит по ссылкам, формам, редиректам, чтобы собрать все возможные URL и точки входа на сайт, которые потом будут проверяться на уязвимости
- Использование ajax spider: предлагается использования более глубокого анализа структуры сайта с использованием браузера. Ajax паук с помощью встроенного браузера выполняет JavaScript и имитирует действия пользователя, чтобы найти динамически загружаемые страницы и запросы на сайте

После нажатия кнопки «Атака» начинается процесс анализа сайта: нахождение всех URL и отправление запросов на поиски уязвимостей.

Так, паук нашел 126 URL:

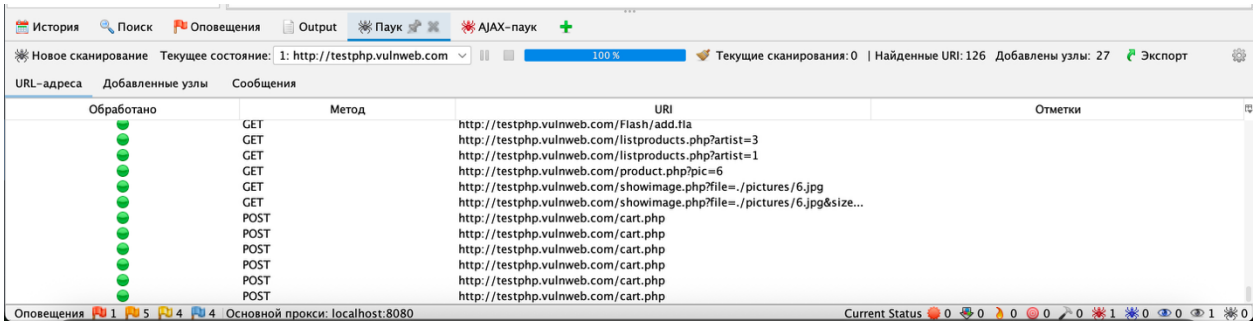

Рисунок 3. Демонстрация работы паука

А ajax паук нашел 2008 URL:

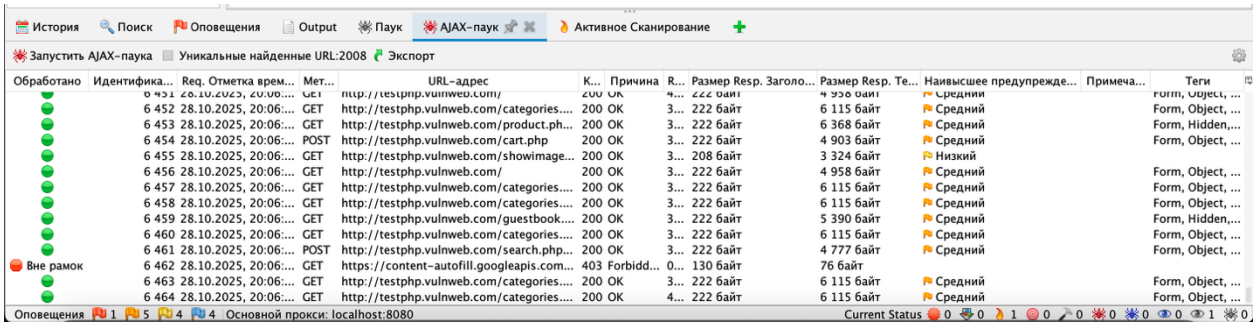

Рисунок 4. Демонстрация работы ajax-паука

После пассивного сканирования (паук + ajax паук) начинается активное сканирование, которое генерирует специальные запросы к сайту с целью

обнаружения уязвимостей. Оно уже более агрессивное и глубокое. Выглядит следующим образом:

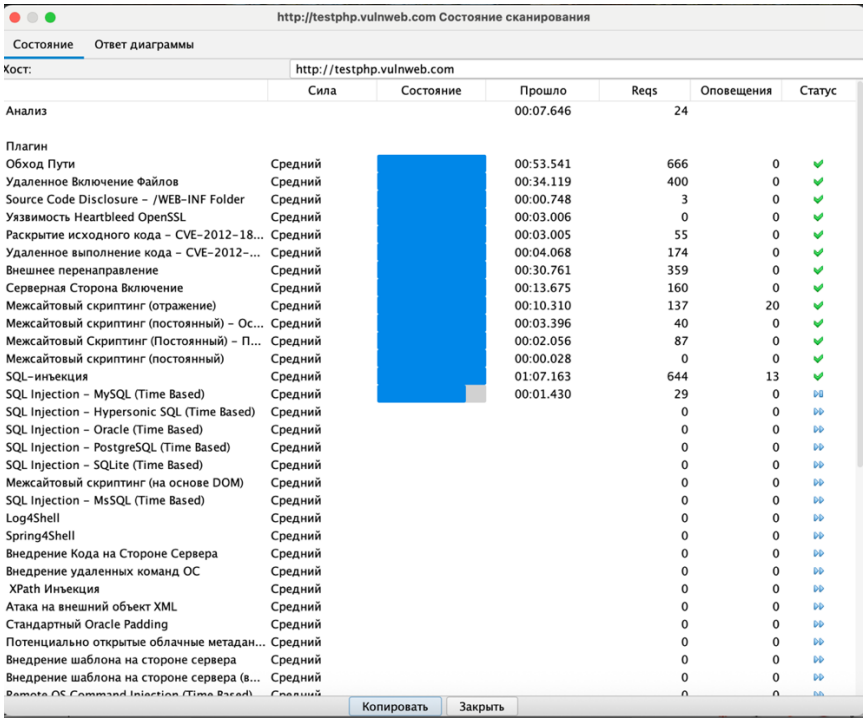

Рисунок 5. Активное сканирование

Все активное сканирование заняло ~13 минут.

По окончании работы программы были найдены следующие уязвимости:

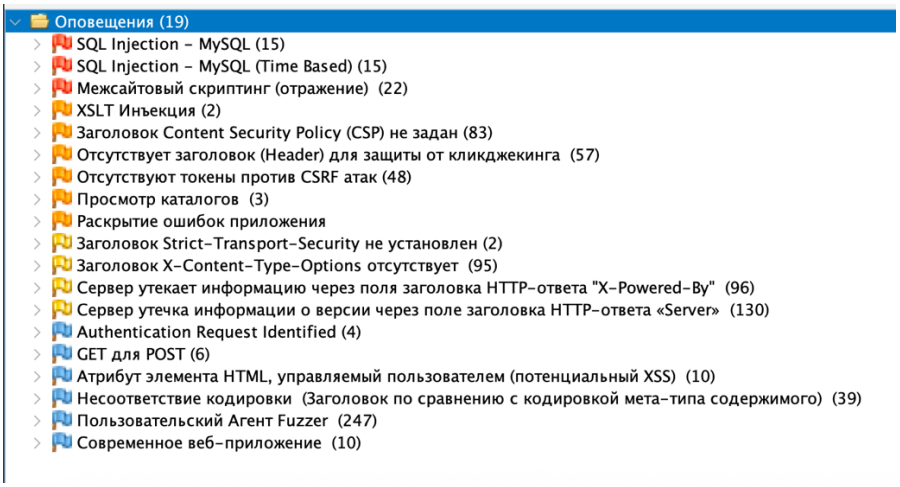

Рисунок 6. Найденные оповещения

Всего 19 оповещений. Из них:

- Высокоприоритетных: 3,
- Среднеприоритетных: 6,

- Низкоприоритетных: 4,
- Информационных: 6

Выберем несколько найденных уязвимостей и проанализируем их, кликнув на уязвимость два раза для получения дополнительной информации

Анализ найденных уязвимостей

1. SQL Injection

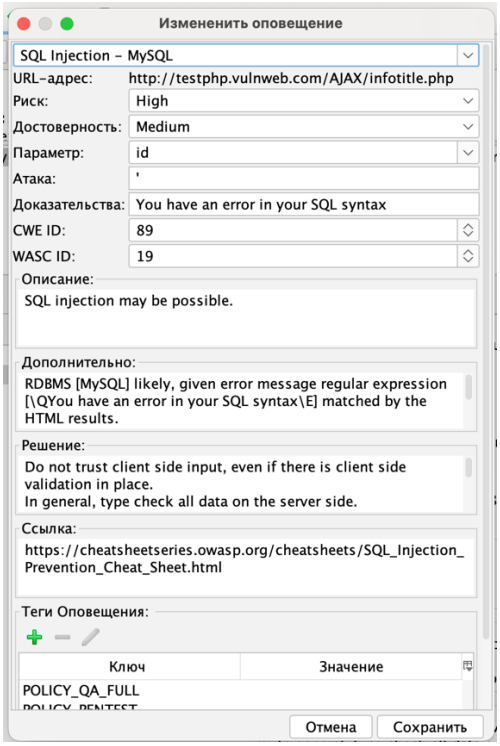

Рисунок 7. Уязвимость SQL Injection

SQLi (SQL injection) – уязвимость при которой злоумышленник может изменить или вставить запрос, чтобы получить неавторизованный доступ к данным или изменить их.

Конкретно в найденной ошибке получаем сообщение «You have an error in your SQL syntax» означает, что строка SQL, которую сервер пытается выполнить, стала синтаксически некорректной — обычно, потому что входные данные от пользователя попали внутрь SQL-строки напрямую.

Для решения этой проблемы необходимо использовать параметризованные запросы (prepared statements) и серверную валидацию/приведение типов — ни в коем случае не конкатенировать пользовательский ввод в SQL.

Кроме того, были найдены time-based SQL инъекции - разновидность слепой SQL-инъекции, при которой атакующий не получает данные напрямую, но заставляет базу данных выполнять задержку, и по времени

ответа извлекает информацию: задержка = «1» (истина), отсутствие задержки = «0» (ложь). За счет этого появляется возможность извлечь данные, даже если приложение не возвращает ошибки/результаты

В моем сканировании было найдено 15 различных time-based MySQL Injection уязвимостей:

Рисунок 8. Time-based уязвимости

2. Межсайтовый скриптинг (отражение)

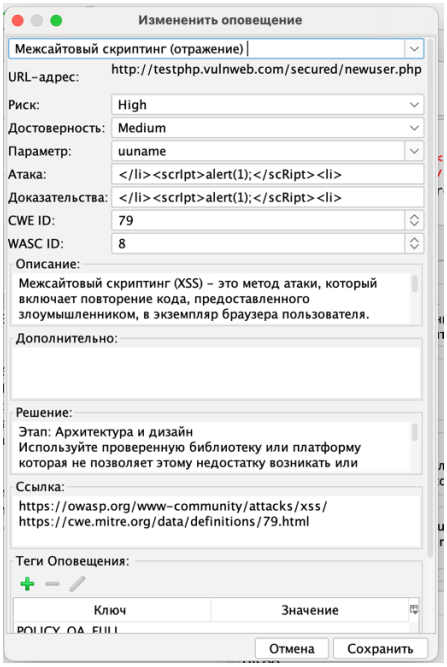

Рисунок 9. Межсайтовый скриптинг (отражение)

XSS – вид атаки при которой сайт отображает на странице данные, введённые пользователем, без проверки и экранирования, из-за чего злоумышленник может вставить свой JavaScript-код, который выполнится в браузере других пользователей.

На скриншоте видно, что ZAP протестировал параметр uuname (например, поле формы) и вставил туда вот такой тестовый скрипт: <scrIpt>alert(1);</scRipt>
. Сервер вернул этот код обратно в ответ, и браузер его выполнил. Это означает, что сайт уязвим к отражённому XSS (Reflected XSS)

Для устранения данной уязвимости необходимо всегда экранировать/санитизировать данные перед вставкой в HTML (включая атрибуты и контент), применять серверную валидацию/white-list, использовать безопасные шаблонизаторы или фреймворки (которые делают автоэкранирование), установить Content Security Policy и флаги куки (HttpOnly, Secure, SameSite)

При переходе на URL найденный в одной из XSS уязвимости видно, что вставленный скрипт действительно вызывает alert(1):

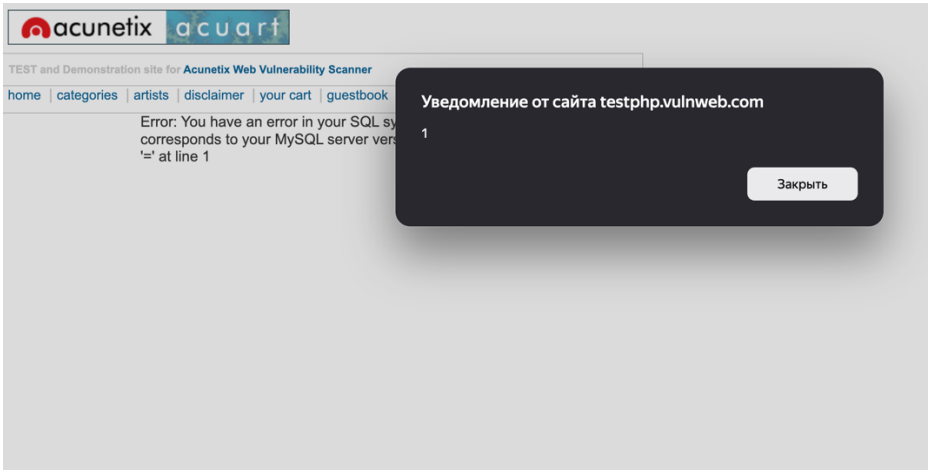

Рисунок 10. Выполнение alert(1) при XSS уязвимости

3. Заголовок Content Security Policy (CSP) не задан

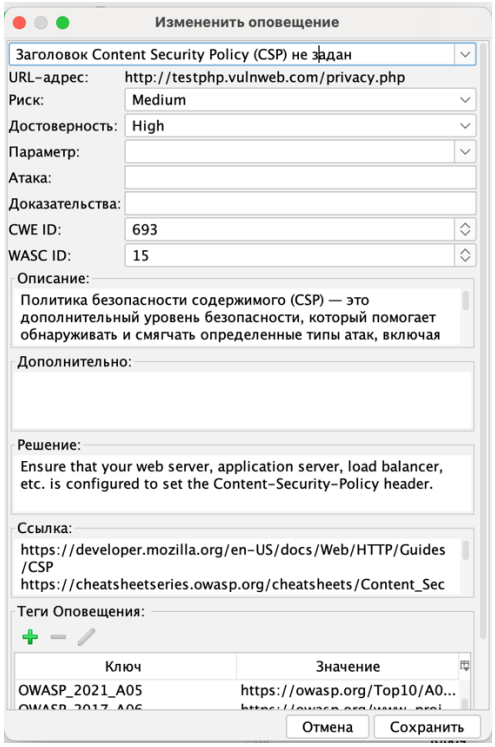

Рисунок 11. Заголовок CSP не задан

Отсутствие заголовка CSP озачает отсутствие политики безопасности содержимого в HTTP-ответах сервера. Это уменьшает защиту от XSS, подмены скриптов и загрузки злонамеренных ресурсов

Для решения этой проблемы необходимо установить HTTP-заголовок Content-Security-Policy (CSP) на веб-сервере или в приложении.Указать разрешённые источники для скриптов, стилей и контента, например:

```
Content-Security-Policy: default-src 'self'; script-src 'self';
```

Это ограничит выполнение внешних скриптов и снизит риск XSS и внедрения вредоносного кода.

4. Отсутствуют токены против CSRF атак

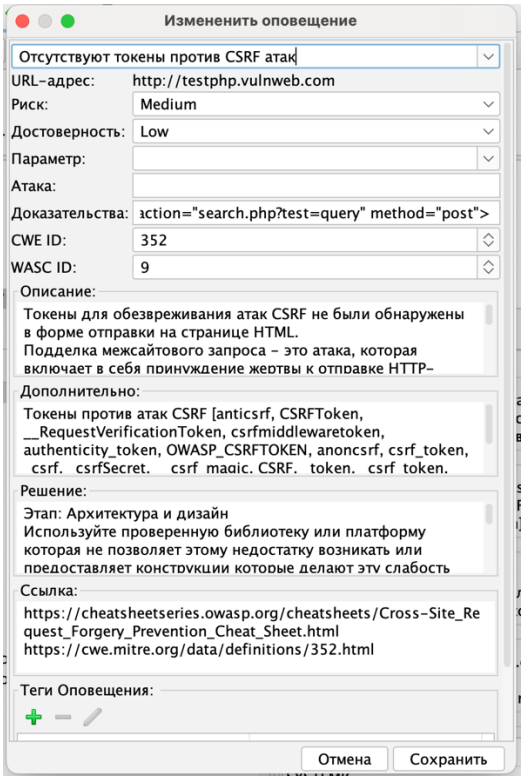

Рисунок 12. Уязвимость отсутствия токенов от CSRF атак

CSRF (Cross-Site Request Forgery) - атака, при которой злоумышленник заставляет браузер жертвы выполнить нежелательный запрос от имени пользователя на доверенный сайт, где он уже авторизован. Например, если пользователь вошёл в свой аккаунт, злоумышленник может с помощью поддельной формы заставить браузер отправить запрос на изменение пароля, перевод денег и т. д., без ведома пользователя.

ZAP обнаружил, что на странице (в форме search.php) отсутствует CSRF-токен — специальный уникальный параметр, который должен добавляться в каждую форму или запрос. Без этого токена сервер не отличит легитимный запрос пользователя от поддельного

Для решения этой проблемы необходимо добавить CSRF-токен в каждую форму. Например:

```
<input type="hidden" name="csrf_token" value="random_string">
```

Или использовать какие-либо встроенные механизмы фреймворков (аннотации). Кроме того, необходимо сначала обеспечить защиту от XSS-атак, так как, если этого не сделать, то csrf-токен может быть украден

Вывод

В ходе данной лабораторной работы было проведено сканирование с помощью OWASP ZAP сайта http://testphp.vulnweb.com. В процессе было найдено несколько уязвимостей различного уровня, а также 2 высокоприоритетные и 2 среднеприоритетные были использованы для подробного анализа. Наиболее опасными оказались: отражённый XSS (позволяет выполнить произвольный JS в браузере жертвы) и SQL-инъекция (позволяет модифицировать/читать данные БД)