DOI 10.37882/2223-2966.2023.7-2.24

РАЗРАБОТКА АНАЛИЗАТОРА НАДЕЖНОСТИ ВЕБ-ПРИЛОЖЕНИЯ НА ОСНОВЕ МОДЕЛИРОВАНИЯ СЕТЕВЫХ АТАК

DEVELOPMENT OF A WEB APPLICATION SECURITY ANALYZER BASED ON NETWORK ATTACK MODELING

A. Rusakov
V. Filatov
S. Dolzhenkov
D. Golubev
D. Saraev
A. Shakhidzhanyan

Summary: The article discusses the basics of web application security, including the different types of attacks that can be performed on web applications. It defines a model of current threats to web applications, highlighting the causes of vulnerabilities. Described the types of attacks such as: injection attacks and authentication and access control vulnerabilities. Reviewed and described the methods of vulnerabilities detection, implemented in the developed solution using special tools. Considered the unique approach to detection and identification of vulnerabilities of the developed security analyzer. Also, the technical aspects of the application are covered, with details of the backend and frontend. The key modules and functions that make the web-application security analyzer operational are described.

Keywords: web application security analysis, vulnerability detection, web-application security threat model, web application security analyzer development.

Русаков Алексей Михайлович
старший преподаватель, МИРЭА — Российский технологический университет
rusal@bk.ru
Филатов Вячеслав Валерьевич
доцент, МИРЭА — Российский технологический университет
filv@mail.ru
Долженков Сергей Сергеевич
ассистент, МИРЭА — Российский технологический университет
dolzhenkov@mirea.ru
Голубев Данил Дмитриевич
МИРЭА — Российский технологический университет
zelenstaff@gmail.com
Сараев Даниил Андреевич
МИРЭА — Российский технологический университет
Den4k543@gmail.com
Шагиджанян Андре Альбертович
МИРЭА — Российский технологический университет
andreshah1@yandex.ru

Аннотация: В статье рассмотрены основы безопасности веб-приложений, включая различные типы атак, которые могут быть совершены на веб-приложения. Определена модель актуальных угроз веб-приложений, в которой выделены причины возникновения уязвимостей. Описаны такие типы атак, как: инъекционные атаки и уязвимости аутентификации и контроля доступа. Рассмотрены и описаны методы обнаружения уязвимостей, реализованные в разрабатываемом решении с использованием специальных инструментов. Рассмотрен уникальный подход к обнаружению и определению уязвимостей разрабатываемого анализатора безопасности. Также, рассмотрены технические аспекты приложения, с деталями бэкенда и фронтенда. Описаны ключевые модули и функции, обеспечивающие работу анализатора безопасности веб-приложений.

Ключевые слова: анализ безопасности веб-приложений, выявление уязвимостей, модель угроз безопасности веб-приложений, разработка анализатора безопасности веб-приложений.

Введение

Разработка веб-приложений включает в себя все больше различных методов и средств построения приложения. Тем не менее, безопасность веб-приложений часто упускается из виду. Из-за разработки приложения на скорую руку и/или плохо написанного кода большинство доступных веб-приложений являются уязвимыми и желанной мишенью для злоумышленников.

Уязвимости могут быть вызваны целым рядом факторов, включая плохие методы написания кода, отсутствие проверки входных данных и устаревшее программное обеспечение. Организациям важно регулярно оценивать и тестировать свои веб-приложения на наличие уязвимостей, а также внедрять такие меры безопасности, как брандмауэры, системы обнаружения вторжений и методы безопасного программирования для защиты от атак.

Исходя из всего вышесказанного, тема разработки анализатора защищенности веб-приложения является актуальной в силу роста атак на веб-приложения. Анализатор веб-безопасности является важным первым шагом в обеспечении безопасности приложений, поскольку он помогает выявлять и оценивать потенциальные уязвимости в веб-приложениях. Эти инструменты сканиру-

ют код, конфигурацию и инфраструктуру приложения, чтобы обнаружить любые слабые места, которые могут быть использованы злоумышленниками. Выявив эти уязвимости на ранней стадии, организации могут предпринять шаги по их устранению до того, как они смогут быть использованы.

Общие сведения по безопасности веб-приложений

Риски безопасности веб-приложений могут возникать в результате взаимодействия между приложением и его пользователями, а также обработки данных на стороне сервера [1]. В отличие от традиционной сетевой безопасности, безопасность веб-приложений требует уникального набора мер безопасности. Веб-приложения разработаны для взаимодействия с пользователями, а это означает, что они уязвимы для широкого спектра угроз, таких как атаки на проверку входных данных, атаки с использованием SQL-инъекций, межсайтовые скрипты (XSS) и атаки на подделку запросов на стороне сервера (SSRF) [2]. Защита конфиденциальных данных, предотвращение несанкционированного доступа к веб-приложениям и сохранение целостности и доступности онлайн-сервисов — все это зависит от уровня защищенности веб-приложений. Веб-приложения могут подвергаться различным атакам, если не приняты достаточные меры безопасности. Нарушение безопасности может оказать большое влияние на инфраструктуру приложения в целом, включая финансовые потери, ущерб репутации и юридическую ответственность.

Модель актуальных угроз безопасности веб-приложений

Одной из наиболее серьезных проблем, с которыми сталкивается безопасность веб-приложений, является постоянно меняющийся ландшафт угроз. Злоумышленники постоянно находят новые способы использовать уязвимости веб-приложений и получить доступ к конфиденциальным данным. Модель угроз веб-приложений представляет собой структуру, описывающую угрозы, с которыми сталкиваются веб-приложения. Модель угроз веб-приложений должна учитывать все элементы, которые содержит приложение, векторы атак, которые могут использовать злоумышленники, и потенциальный ущерб, который злоумышленник может нанести. Распространенной моделью для понимания угроз безопасности веб-приложений является проект OWASP Top 10, который предоставляет полный список наиболее критических рисков безопасности, с которыми повседневно сталкиваются веб-приложения [5]. Проект OWASP Top 10 также может быть использован в качестве справочного материала для определения типов угроз безопасности, которым может подвергаться веб-приложение, и способов снижения этих рисков.

Первым шагом в обеспечении безопасности веб-приложения является понимание типов атак, к которым оно может быть уязвимо. Определение типов атак является важнейшим шагом в обеспечении безопасности приложения.

Инъекционные атаки

Инъекционные атаки представляют собой процесс вставки вредоносного кода злоумышленником в операционную систему веб-приложения, серверную базу данных или другие слабые места, своего рода уязвимость в системе безопасности веб-приложения. Инъекционные атаки бывают различных форм, таких как SQL-инъекция, NoSQL-инъекция и внедрение команд операционной системы.

Одним из наиболее распространенных видов инъекционных атак является SQL-инъекция. Это происходит, когда злоумышленник вставляет вредоносные инструкции SQL в серверную базу данных веб-приложения [3]. В результате злоумышленник имеет возможность редактировать или извлекать конфиденциальные данные из базы данных. Существует несколько типов атак SQL-инъекций, включая атаки вслепую (blind-based), основанные на логических значениях (boolean-based), основанные на времени (time-based) и атаки за пределами (Out-Of-Bound) [4].

Внедрение NoSQL аналогично SQL-инъекции, однако оно затрагивает только базы данных NoSQL [5]. В базах данных NoSQL отсутствует установленная схема данных, в отличие от обычных баз данных SQL, что делает их более восприимчивыми к инъекционным атакам. Подобно атакам с использованием SQL-инъекций, атаки с использованием NoSQL-инъекций включают вставку вредоносного кода в параметры запроса веб-приложения. Атаки с внедрением NoSQL могут включать в себя, среди прочего, внедрение дополнительных запросов или изменение уже существующих запросов для получения незаконных данных из базы данных. Атаки с внедрением NoSQL также могут быть использованы злоумышленниками для обхода систем аутентификации и контроля доступа или запуска произвольного кода на сервере.

Инъекционная атака, известная как внедрение команд операционной системы, нацелена на операционную систему веб-приложения [6]. Этот метод предоставляет злоумышленнику доступ к операционной системе сервера веб-приложений и позволяет ему выполнять любые команды операционной системы, которые он выберет. Данный вид атаки может быть осуществлен путем ввода вредоносных команд в параметры запроса веб-приложения или другие поля ввода, которые позже используются для генерации команд операционной системы. Выполняя команды от имени привилегированного

пользователя, злоумышленники также могут использовать этот вид атаки для обхода систем аутентификации и контроля доступа.

Уязвимости аутентификации и контроля доступа

Другой типичный класс угроз безопасности веб-приложений связан с аутентификацией и контролем доступа. Из-за этих недостатков злоумышленники могут обойти процессы аутентификации и получить доступ к конфиденциальным данным без авторизации. Когда веб-приложение неправильно управляет аутентификацией пользователя и токенами сеанса, оно уязвимо для нарушенной аутентификации (Broken authentication), а также для управления сеансом (Session management). Это может привести к краже данных и компрометации учетных записей пользователей. Когда веб-приложение предоставляет прямую ссылку на внутренний объект, такой как файл или запись базы данных, без требуемых ограничений доступа, это определяется как небезопасная прямая ссылка на объект [7]. Злоумышленники могут получить конфиденциальную информацию или отредактировать данные, которые в результате не должны быть доступны. Нарушенные уязвимости контроля доступа позволяют злоумышленнику получить несанкционированный доступ к функциям или данным веб-приложения. Это может произойти, когда веб-приложение неэффективно применяет средства контроля доступа или, когда они настроены неправильно.

Методы определения уязвимостей веб-приложения

SQL-инъекция является техникой инъекции кода, которая может использовать уязвимость в уровне базы данных веб-приложения. Эта техника обычно используется хакерами для манипулирования SQL-запросами и получения несанкционированного доступа к системе. Одним из стандартных методов обнаружения уязвимости SQL-инъекции является ввод метасимволов SQL в поля ввода пользователя и наблюдение за реакцией приложения. Если приложение возвращает ошибку SQL, это может свидетельствовать об уязвимости [8]. Однако этот метод требует глубокого понимания синтаксиса и семантики SQL и является довольно трудоемким. Одним из наиболее эффективных инструментов для обнаружения уязвимостей SQL-инъекций является sqlmap. Он автоматизирует процесс обнаружения и эксплуатации уязвимостей SQL-инъекций и предоставляет доступ к фундаментальной базе данных уязвимого приложения. Инструмент поддерживает различные типы SQL-инъекций, включая слепые на основе булевых функций (Blind Boolean-based), слепые на основе времени (Blind time-based), на основе ошибок (error-based), на основе UNION-запросов (UNION query-based), на основе вложенных запросов и атаки за пределами (Out-Of-Bound).

Межсайтовый скриптинг (XSS) — это еще одна широко распространенная уязвимость безопасности в веб-приложениях. Она позволяет злоумышленникам внедрять вредоносные сценарии на веб-страницы, просматриваемые другими пользователями, что потенциально может привести к перехвату сеанса, краже личных данных или искажению работы веб-приложения. Традиционные методы обнаружения XSS-уязвимостей включают ввод полезной нагрузки JavaScript в поля ввода пользователя и наблюдение за тем, выполняет ли браузер скрипт [8]. Подобно обнаружению SQL-инъекций, этот метод требует хорошего понимания JavaScript и различных контекстов, в которых он может быть выполнен. XSStrike является инструментом, который упрощает и повышает эффективность обнаружения XSS. Он использует интеллектуальную генерацию полезной нагрузки и обнаружение аномалий для поиска XSS-уязвимостей. Он также оснащен мощным механизмом fuzzing и поддерживает HTTP-методы GET и POST, cookies, многопоточность и множество параметров.

Подделка запросов со стороны сервера (SSRF) представляет из себя уязвимость, которая позволяет злоумышленнику заставить сервер выполнять запросы от его имени. Она может быть использована для взаимодействия с внутренними ресурсами, выполнения действий от имени сервера или для атаки на другие системы. Обнаружение уязвимостей SSRF заключается в том, чтобы обманом заставить сервер выполнить запрос в непредусмотренном месте. Обычно это включает манипулирование полями URL или любой функцией, позволяющей серверу получать ресурсы из Интернета [8]. See-Surf — это мощный инструмент для обнаружения SSRF уязвимостей. Он работает путем сканирования параметров и конечных точек, которые могут быть уязвимы для SSRF, а затем пытается выполнить запросы к заранее определенным конечным точкам.

Подход к обнаружению потенциальных угроз

Разрабатываемый анализатор безопасности веб-приложений, использует уникальный подход к оценке уязвимостей, используя комбинацию инструментов с открытым исходным кодом, таких как sqlmap, XSStrike и see-surf для обнаружения таких потенциальных угроз, как SQL-инъекции, межсайтовый скриптинг (XSS) и подделка запросов на стороне сервера (SSRF). Благодаря интеграции инструментов с открытым исходным кодом, разрабатываемый анализатор имитирует кибератаки в реальном мире для выявления и устранения уязвимостей в веб-приложениях, тем самым, обеспечивая более точное представление о реальных кибератаках, выявляя уязвимости, которые автоматические анализаторы безопасности могут не обнаружить. Имитируя эти атаки, разрабатываемое решение может помочь разработчикам понять, как их приложения могут реагировать на реаль-

ные угрозы. Так же, подобные анализаторы тестирования на проникновение могут использовать опыт специалистов по безопасности или этичных хакеров, которые могут мыслить, как злоумышленники и использовать свой опыт для выявления уязвимостей, которые автоматические инструменты могут не обнаружить.

Архитектура программного обеспечения

Архитектура инструмента анализа надежности веб-приложений построена вокруг двух ключевых компонентов: бэкенда, работающего на FastAPI и Uvicorn, и фронтенда, работающего на ReactJS.

Связь между фронтендом (ReactJS) и бэкендом (FastAPI) в реализуемом решении устанавливается с помощью HTTP(S) запросов и ответов. Это стандартная модель взаимодействия клиента и сервера, при которой клиент отправляет запрос, а сервер возвращает ответ.

FastAPI предназначен для обработки различных типов запросов, таких как GET, POST, PUT, DELETE и др. Каждый из этих запросов соответствует определенному типу операций. Например, запрос GET обычно используется для получения данных, запрос POST — для отправки данных, запрос PUT — для обновления данных, а запрос DELETE — для удаления данных.

На клиентской части приложение ReactJS использует HTTP(S) запросы для связи с бэкендом FastAPI. Для этого используется Fetch API или библиотеки, такие как Axios, которые предоставляют более удобный и мощный API для выполнения запросов и обработки ответов. Когда приложению ReactJS нужно взаимодействовать с бэкендом (например, получить или отправить данные), оно делает HTTP(S) запрос к соответствующей конечной точке на сервере FastAPI. Сервер обрабатывает запрос, взаимодействует с базой данных или выполняет другие необходимые операции, а затем отправляет ответ обратно клиенту. Этот ответ затем обрабатывается приложением ReactJS, которое соответствующим образом обновляет пользовательский интерфейс. Преимущество такого подхода заключается в том, что он позволяет эффективно и динамично обновлять пользовательский интерфейс на основе взаимодействия с пользователем и ответов сервера без необходимости полной перезагрузки страницы. Этот метод взаимодействия клиента и сервера составляет основу взаимодействия между бэкендом FastAPI и фронтендом ReactJS в разрабатываемом анализаторе надежности.

Используя сильные стороны FastAPI и ReactJS и их эффективное взаимодействие, инструмент анализа надежности веб-приложений способен обеспечить мощный и отзывчивый пользовательский опыт.

Реализация анализатора надежности

Реализация анализатора безопасности включает в себя хорошо структурированный бэкенд, использующий фреймворк FastAPI и ASGI-сервер Uvicorn. Серверная часть отвечает за взаимодействие с инструментами SQLmap, XSStrike и See-Surf и предоставляет конечные точки для взаимодействия с фронтендом ReactJS-приложения. Кодовая база является модульной, с отдельными модулями Python, предназначенными для каждой функциональности для лучшей обслуживаемости и масштабируемости. Ниже приведена структура кода бэкенда:

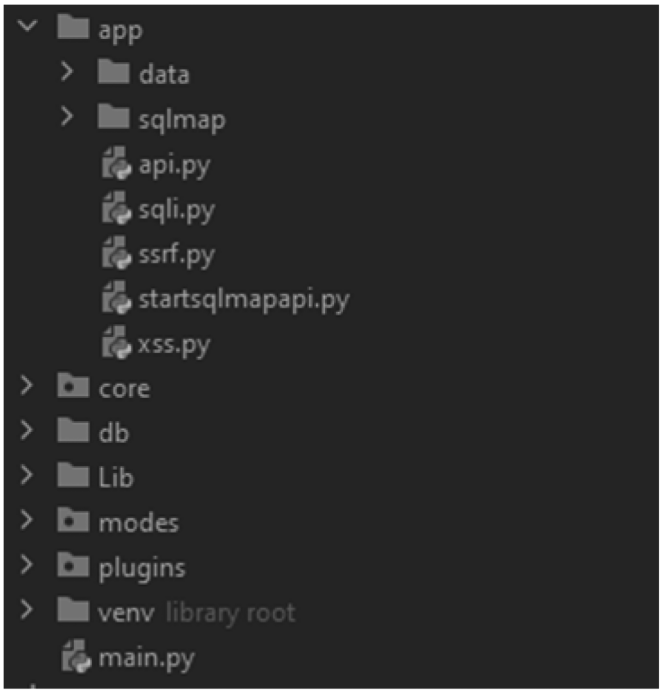

Рис. 1. Структура серверной части

1. Расположенная по адресу `code/main.py`, является точкой входа в приложение бэкенда. Отвечает за инициализацию и запуск ASGI-сервера Uvicorn, который используется для запуска приложения FastAPI. Uvicorn представляет из себя молниеносную реализацию ASGI-сервера, использующую uvloop и httptools, что делает Uvicorn идеальным выбором для запуска асинхронных веб-приложений на Python.
2. Файл, расположенный по адресу `code/app/api.py`, содержит основные маршруты бэкенд-приложения. Эти маршруты определяют конечные точки API, с которыми взаимодействует фронтенд-приложение. Фреймворк FastAPI упрощает создание и управление этими маршрутами, поддерживая валидацию запросов, сериализацию и автоматическое документирование API.
3. Модуль sqli.py, расположенный в каталоге `code/app`, предназначен для обнаружения уязвимо-

стей SQL Injection. Этот модуль взаимодействует с инструментом SQLmap через его REST JSON API, отправляя запросы к SQLmap API и обрабатывая ответы. Модуль отвечает за инициирование сканирования SQLmap и получение результатов, которые затем отправляются в качестве ответов на соответствующие запросы API с фронтенда.

4. Модуль startsqlmapapi.py, также находящийся в каталоге `code/app`, запускает подпроцесс, который запускает сервер SQLmap REST API. Этот сервер необходим модулю sqli.py для взаимодействия с SQLmap. Подпроцесс запускается при инициализации внутреннего приложения, гарантируя, что сервер SQLmap API готов принимать запросы.
5. Модуль xss.py отвечает за обнаружение уязвимостей межсайтового скриптинга (XSS). Этот модуль взаимодействует с инструментом XSStrike, инициируя сканирование на наличие XSS-уязвимостей и обрабатывая результаты. Результаты затем отправляются в качестве ответов на соответствующие API-запросы с фронтенда.
6. Модуль ssrf.py занимается обнаружением уязвимостей Server-Side Request Forgery (SSRF). Он взаимодействует с инструментом See-Surf, инициируя сканирование SSRF и обрабатывая результаты. Как и в других модулях, результаты отправляются в виде ответов на соответствующие API-запросы с фронтенда.

Такая модульная структура кода позволяет четко разделить обязанности, что облегчает понимание, обслуживание и масштабирование кодовой базы. Каждый модуль отвечает за определенный тип обнаружения уязвимостей, что гарантирует, что изменения в одном модуле не повлияют на другие. Такое разделение также позволяет легко добавлять новые модули в будущем, если возникнет необходимость в расширении типов обнаружения уязвимостей.

Структура клиентской части состоит из нескольких компонентов, каждый из которых отвечает за различные аспекты функциональности приложения:

1. `public/index.html`: Это основной HTML-файл, который загружается, когда кто-то посещает сайт. Он служит контейнером для приложения React.
2. `src/routes/App.jsx`: Здесь определяется маршрутизация приложения с помощью `react-router-dom`. Он включает два основных маршрута: `Queues` и `Results`. Каждый маршрут соответствует отдельному компоненту, который отображается, когда пользователь переходит по соответствующему маршруту.
3. `src/routes/Queues.jsx`: Этот компонент управляет отображением списка очередей, показывая статус каждого элемента очереди.

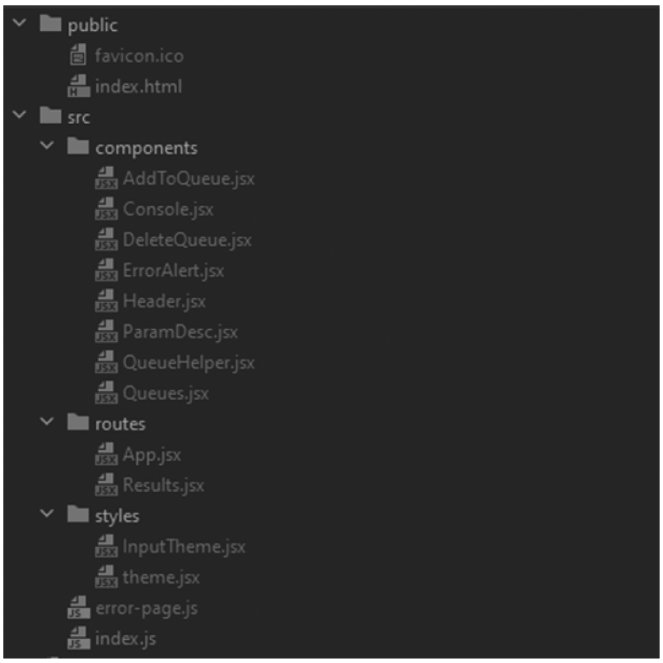

Рис. 2. Структура клиентской части

4. `src/routes/Results.jsx`: Этот компонент отвечает за отображение результатов анализа. Он отображает результаты, визуализируя их и обрабатывая определенным образом, в зависимости от элемента очереди.
5. `src/components/AddToQueue.jsx`: Этот компонент обрабатывает функциональность для добавления новых элементов в очередь. Он включает форму и другие элементы ввода данных для конфигурирования вектора анализа, добавляя новый элемент в очередь. Так же, содержит функцию инициализации процесса анализа.
6. `src/components/DeleteQueue.jsx`: Этот компонент позволяет пользователям удалять элементы из очереди.
7. `src/components/ErrorAlert.jsx`: Этот компонент отвечает за отображение сообщений об ошибках для пользователя.
8. `src/components/Header.jsx`: Содержит заголовок приложения.
9. `src/components/ParamDesc.jsx`: Реализован для отображения описания каждого параметра конфигурации тестирования.
10. `src/components/QueueHelper.jsx`: Полезный компонент, который предоставляет дополнительную функциональность для элементов очереди, реализует функцию удаления элементов из очереди.

Компоненты организованы в иерархическом порядке, при этом `App` выступает в качестве корневого компонента. Внутри `App` маршрутизация настроена на отображение либо очередей `Queues`, либо результатов `Results` в зависимости от текущего URL. Компонент

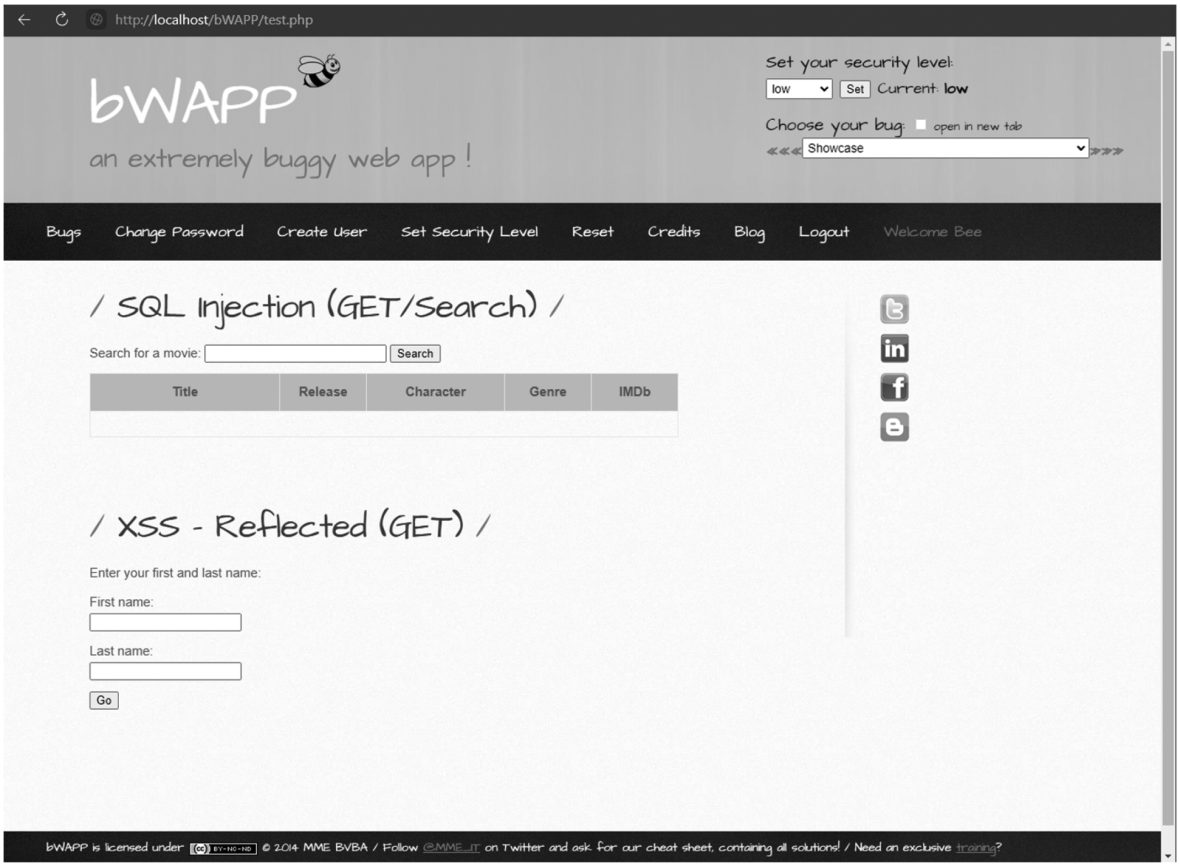

Рис. 3. Уязвимое приложение bWAPP

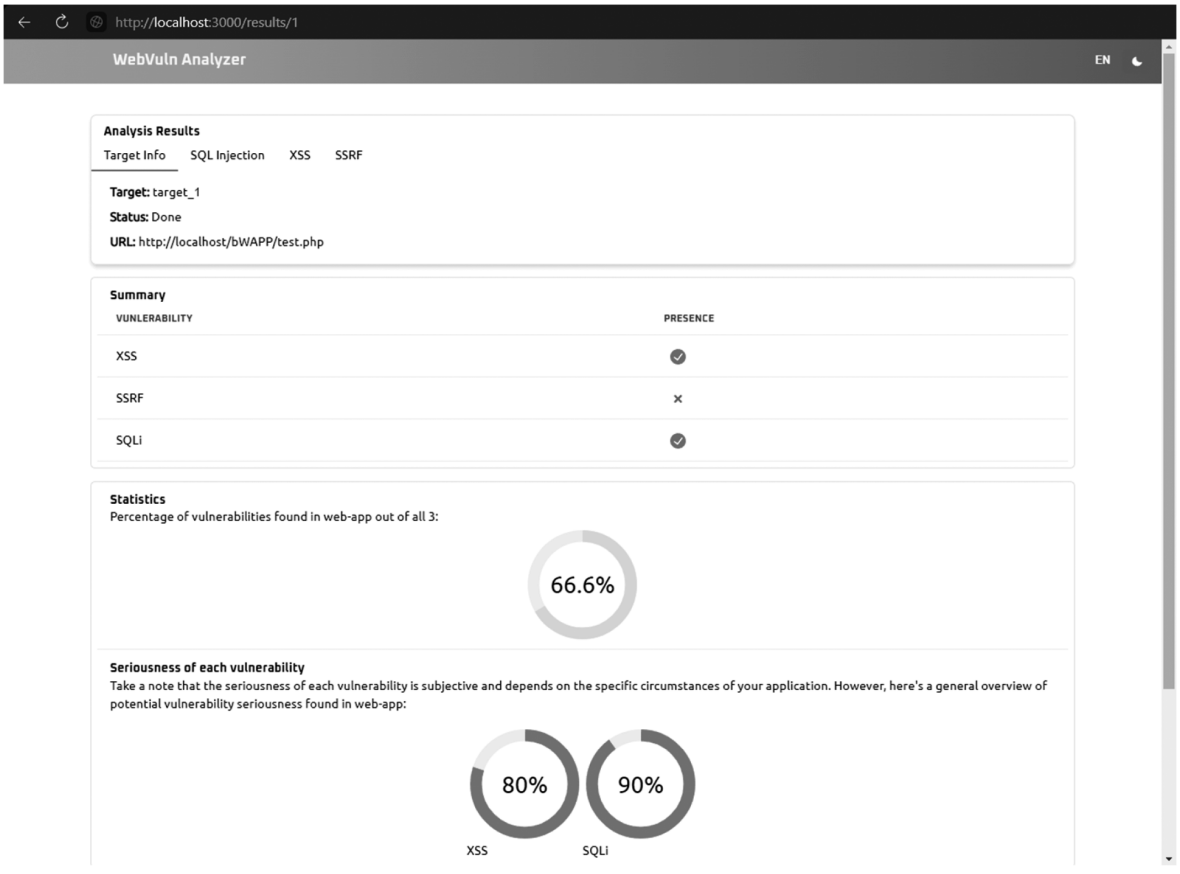

Рис. 4. Результаты анализа SQLi и XSS

Рис. 5. Уязвимое приложение SSRF Lab

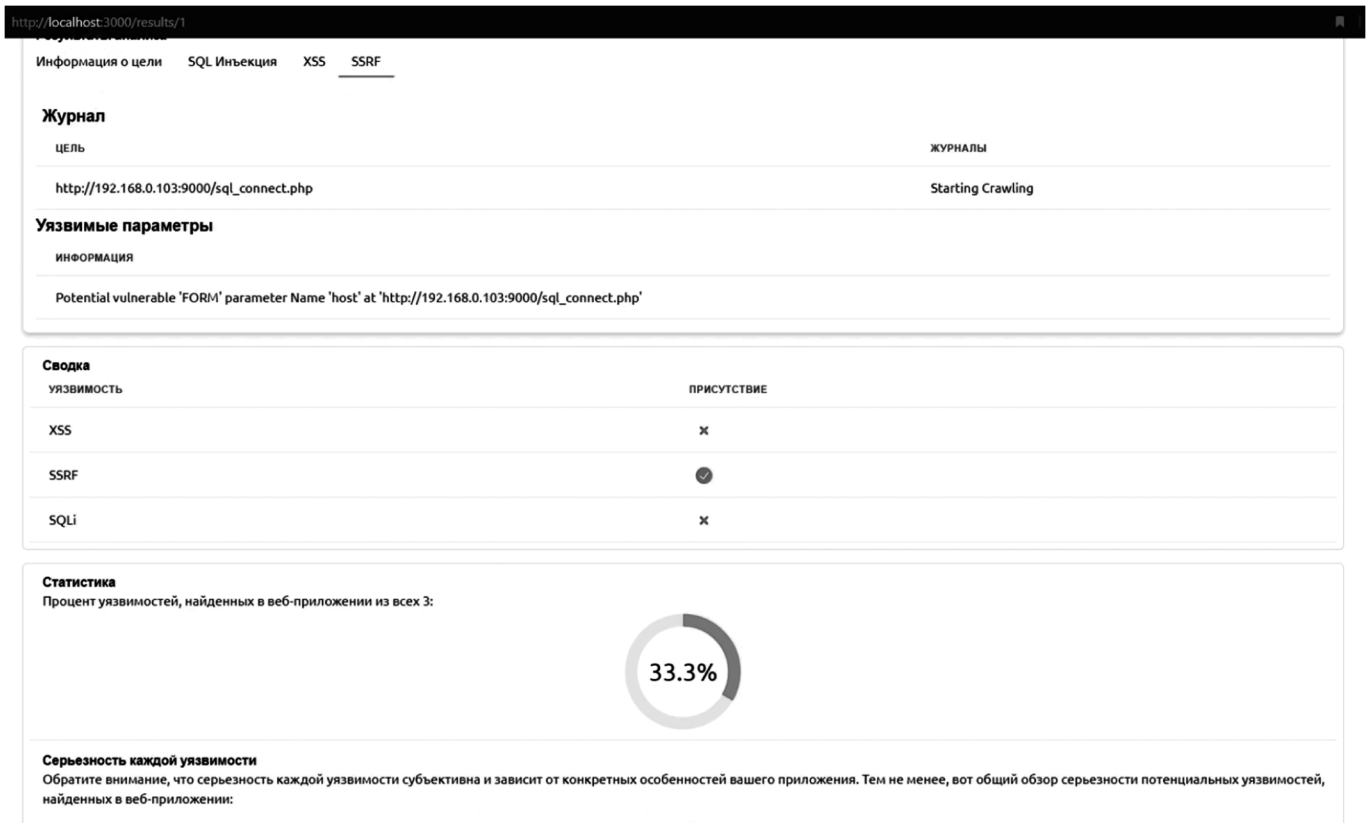

Рис. 6. Результаты анализа SSRF

`Queues` затем использует `AddToQueue` и другие компоненты для обеспечения взаимодействия с очередью.

Анализ результатов работы системы

Необходимо проанализировать результаты работы системы, а именно способность правильно определять соответствующие уязвимости. Приложение-жертва, используемое для этой цели, — 'bWAPP', является уязвимым веб-приложением.

Система выдает результаты тестирования на основе обнаруженных уязвимостей SQLi и XSS, тем самым выступая в качестве эффективного инструмента анализа безопасности.

Для тестирования модуля определения SSRF будет использовано Server-Side Request Forgery (SSRF) vulnerable Lab — веб-приложение для тестирования соответствующей уязвимости.

Модуль определения SSRF успешно выполнил запросы к внешним ресурсам и определил потенциально уязвимые параметры, тем самым, свидетельствуя о присутствии уязвимости.

Заключение

Таким образом, было разработано гибкое и эффективное решение анализа безопасности веб-приложений, выявляющее соответствующие уязвимости. Персональный анализатор веб-приложений эффективно продемонстрировал свои возможности в обнаружении значительных уязвимостей безопасности, таких как SQL Injection (SQLi), Cross-Site Scripting (XSS) и Server Side Request Forgery (SSRF). Для уменьшения последствий обнаруженных уязвимостей и усиления защиты от потенциальных атак были даны целевые рекомендации. Эти рекомендации направлены не только на немедленное устранение выявленных уязвимостей, но и на создание надежной, долгосрочной практики безопасного написания кода. Также, рассмотрены и описаны методы обнаружения уязвимостей, реализованные в разрабатываемом решении с использованием специальных инструментов. Эти инструменты значительно расширили возможности по обнаружению уязвимостей веб-приложений, предлагая всестороннее сканирование и высокую степень автоматизации.

ЛИТЕРАТУРА

1. Huang H.C. et al. Web application security: threats, countermeasures, and pitfalls //Computer. — 2017. — Т. 50. — №. 6. — С. 81–85. (дата обращения: 05.01.2023)
2. 8 Critical Web Application Vulnerabilities and How to Prevent Them. [Электронный ресурс] URL: https://brightsec.com/blog/web-application-vulnerabilities/ (дата обращения: 08.01.2023)
3. Sharma, Chandershekhar & Jain, Sushil. (2017). SQL Injection Attacks on Web Applications. (дата обращения: 20.01.2023)
4. Dizdar A. SQL injection attack: Real life attacks and code examples //Retrieved April. — 2021. — Т. 15. — С. 2022. (дата обращения: 27.01.2023)
5. Eassa A.M. et al. NoSQL injection attack detection in web applications using RESTful service //Programming and Computer Software. — 2018. — Т. 44. — С. 435–444. (дата обращения: 31.01.2023)
6. ALAHMAD M., ALKANDARI A., ALAWADHI N. SURVEY OF OS COMMAND INJECTION WEB APPLICATION VULNERABILITY ATTACK //Journal of Engineering Science and Technology. — 2022. — Т. 17. — №. 1. — С. 0075–0084. (дата обращения: 02.02.2023)
7. Prasad P. Mastering Modern Web Penetration Testing. — Packt Publishing Ltd, 2016. (дата обращения: 05.02.2023)
8. Swead M.A., Almustafa M.M. Developing a methodology for web applications vulnerabilities analysis and detection. — 2019. (дата обращения: 04.03.2023)
9. Голубев Д.Д. Исходный код системы анализа безопасности веб-приложений. https://github.com/5vver/WebVuln-Analyzer. (2023).