Введение

Данная исследовательская работа посвящена безопасности Django framework.

Django – это большой веб-фраеморк написанный на Python, реализующий паттерн MVC, в который встроено все необходимое для постороения большинства веб-приложений. Django позволяет удешевить разработку за счет встроенных механизмов работы с базой данных, рендеринга HTML-шаблонов, роутинга, а также большого количества библиотек, как специфичных для Django, так и из богатой экосистемы Python.

Цели

Целью данной работы является исследование Django framework с точки зрения защиты инфромации. Приступая к данной работе я заранее понимаю, что найти реальные уязвимости является достаточно сложной и практически нереальной задачей. Причиной тому является зрелость данной технологии: django существует с 2005-го года, обладает большим сообществом разработчиков, используется компаниями IT-гигантами. С учетом данного факта я ставлю перед собой следующие цели:

* Исследовать структуру Django: выделить основные компоненты и опеределить их зону отвественности.
* Исследовать механизмы авторизации, аутентификации и идентификация спецефичные для веб-приложений, реализовать их на практике с помощью Django и на практике объяснить их достоинства и недостатки.
* Исследовать встроенные механизмы защиты и предпринять попытки обойти их, как в стандартных ситуациях, так и в ситуациях, когда разработчиком допущены ошибки, создающие уязвимости.

Более глобальной целью является выделение и обоснование знаний, которые помогут Django-разрабочикам создавать безопасные веб-приложения.

Структура работы

Данная работа будет разбита на 3 главы, соответственно отвечающие за каждую сформулированную цель.

*Глава 1.* Изучение структуры Django. Как framework работает, из каких слоев состоит, какие задачи решает каждый слой.

*Глава 2.* Практическая реализация различных способов авторизации и аутентификации (base authorization, session authorization, token authrization, oauth 1.0, oauth 2.0) и демонстрация на практике достоинств и недостатков каждого способа.

*Глава 3.* Изучение встроенных механизмов защиты от распостраненных видов атак: csrf, clickjacing, ddos, sql инъекции, подбор пароля, намеренное допущение ошибок, которые позволят данные атаки провести успешно и релизация защиты от них.

Дополнительно в главы будут встраиваться кейсы, которые будут нумероваться отдельно. Кейс – это конфигурция Django-проекта и последовательность действий, которые позволят либо получить неправомерный доступ к ресурсу поставляемому приложением, либо помешать правомерному доступу.

Кейсы:

1. Проверка админки

2. Тестирование свободного места

3. Подмена токена через JS

4. SQL raw запрос

Глава 1

Данная глава посвящана описанию Django. В ней я поставраюсь ответить следующей на вопросы:

* Когда появился и развивался Django?
* Какие задачи Django решает?
* Для чего используется?
* Какая архитектура у Django?

История Django

Выпущен фраемворк Django был в 2005 году, т.е. более чем в 15 лет назад. На протяжении всего своего жизненного цикла он активно развивался и продолжает развиваться (версия 4.0 была представлена 7 декабря 2021 года) (ссылка).

С момента появления Django веб-разработка сильно изменилась: браузеры стали функциональнее и стандартизирование, активное развитие получил javascript, json захватил мир общения между сервером и клиентом, повсеместным стало REST-API, в тренд пришла асинхронность.

На протяжении своей истории Django все старался соответствовать трендам: в 2011 году был представлен Django Rest Framework для легкой реализации REST API поверх Django, в 2013 появилась поддержка Python 3, в 2019 с релизом версии 3.0 появилась поддержка асинхронности.

За счет долгой истории Django, с одной стороны, успел раздуться и обрасти редкоиспользуемыми в мире современной разработки (например, RSS-ленты), с другой обладает встроенными возможностями для реализации практически любых требований. Однако главное, что спустя долгий путь развития порог входа в разработку на Django остался низким.

Какие задачи Django решает

Дописать. Основные задачи, решаемые Django:

* Взаимодействия кода на Python и базы данных. Django обладает собственной ORM, которая поддерживает транзакции и миграции. Обладает собственными абстракциями над SQL, позволяющими взаимодействовать с базой данных абсолютно не зная SQL (хотя для профессионального разработчика это и обязательно).
* Постороение административной панели сайта. В Django есть встроенная возможность, которая позволяет малыми силами создать стандартизированную панель для создания, изменения и удаления объектов в БД.
* Построение HTML шаблонов. В Django встроена система HTML шаблонов с собственными тегами и наследованием, которая позволяет строить отобржаение данных в виде HTML.
* Маршрутизации запросов. В Django встроен диспетчер URL на основе регулярных выражений, который позволяет преобразовывать запросы в вызовы функций.
* Авторизация и утентификация пользователей.
* Работа с формами.

Для чего Django используется?