**Отчет по найденным уязвимостям**

**Задание 1.1**

**Описание функционала выполнения кода:** веб-приложение на Flask. При GET запросе по корневому пути в браузере отображается форма загрузки файла и кнопка отправки его на сервер. POST запрос по корневому пути служит для обработки отправленного файла и сохранение его в папку upload. GET запрос /share?filename={file} откроет загруженный файл и отобразит на странице его содержимое.

**Описание найденных уязвимостей:** *Command Injection и XSS*.1)из-за обычной конкатенации строк при разрешении пути сохранения файла получается, что, задав имя файла ../customlog.py мы можем изменить поведение сервера при GET или POST по корневому пути. Внутри обработчика пути / вызывается customlog.log(). Тем самым, можно заставить сервер выполнить любой python код, реализовав в загружаемом файле функцию log(). Это может быть чтение конфига или переменных окружения, копирование в папку upload важных файлов, содержимое которых можно будет просмотреть через /share?filename=passwords.txt например. Отправить данные на удаленный сервер и т.д.

2) Также можно загрузить опасный для пользователей файл. Например html с кодом на javascript, который компрометирует cookie пользователя на веб-сервере и другие подобные файлы.

Оба примера эксплуатации представлены ниже.

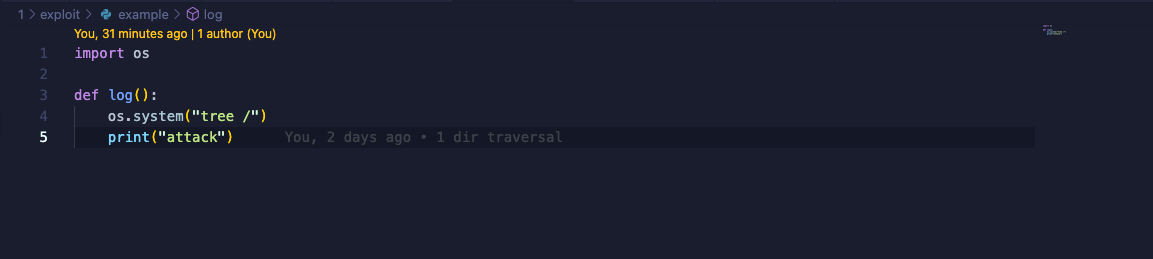
**Пример эксплойта:**

Рисунок 1 – файл, загружаемый на сервер (1/exploit/example)

****

Рисунок 2 – отправка кода эксплойта на сервер (1/exploit/exploit.py)

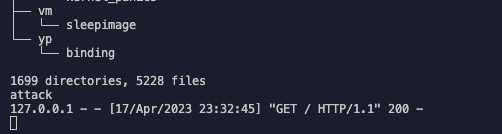


Рисунок 3 – пример эксплуатации уязвимости

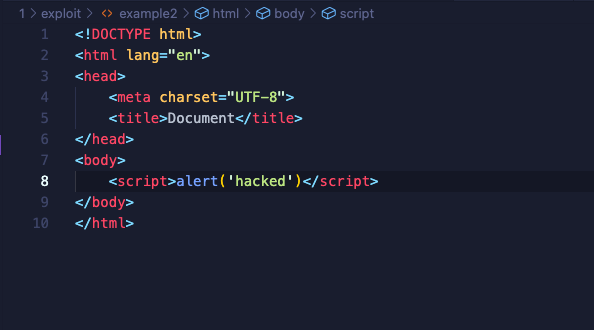


Рисунок 4 – файл с вредоносным js (1/exploit/example2)



Рисунок 5 – отправка вредоносного файла на сервер (1/exploit/exploit2.py)

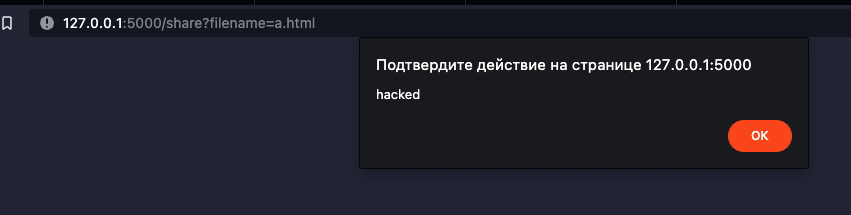


Рисунок 6 – пример эксплуатации уязвимости

**Рекомендация по устранению уязвимости:** использовать функции, 'очищающие' название загружаемого файла (например werkzeug.utils.secure\_filename(filename) для нашего примера). Также стоит проверять, какой контент содержит в себе загружаемый файл.

**Задание 1.2**

**Описание функционала выполнения кода:**

**Описание найденных уязвимостей:**

**Пример эксплойта:**

**Рекомендация по устранению уязвимости:**

**Задание 1.3**

**Описание функционала выполнения кода:**

**Описание найденных уязвимостей:**

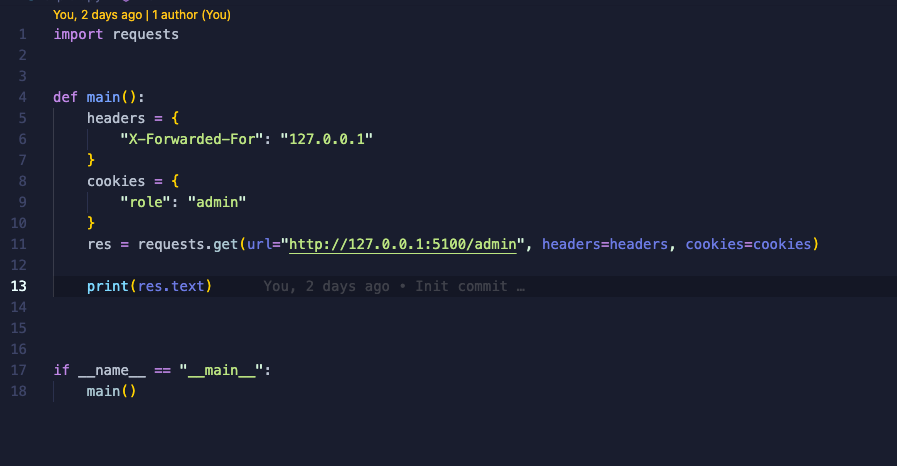
**Пример эксплойта:**

**Рекомендация по устранению уязвимости:**

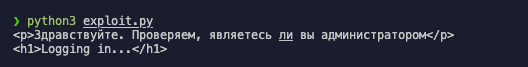
**Задание 1.4**

**Описание функционала выполнения кода:** веб-приложение на golang. Путь /admin проверяет роль пользователя и ip, с которого он отправил запрос. При успешной проверке на права администратора в браузере отображается “Logging in”.

**Описание найденных уязвимостей:** *BrokenAuth*. легко подделать параметры запроса, чтобы сервер определил пользователя, как админа. Достаточно подставить куки 'role' со значением 'admin' и заголовок 'X-Forwarded-For' со значением 'localhost' или '127.0.0.1'. Пример эксплойта на python представлен ниже.

**Пример эксплойта:**

**Рисунок . – код эксплойта (4/exploit.py)**

****

**Рисунок . – эксплуатация уязвимости**

**Рекомендация по устранению уязвимости:** не использовать значение куки role с простым значением, использовать сгенерированные на сервере JWT (подписанные сервером, stateless) или стандартные сессии (stateful) без возможности перебрать их значения за вменяемое время. Тогда неприятности могут возникнуть только в том случае, если какой-либо админ нечаянно "сольет" свой токен.

**Задание 1.5**

**Описание функционала выполнения кода:**

**Описание найденных уязвимостей:**

**Пример эксплойта:**

**Рекомендация по устранению уязвимости:**

**Задание 1.6**

**Описание функционала выполнения кода:**

**Описание найденных уязвимостей:**

**Пример эксплойта:**

**Рекомендация по устранению уязвимости:**

**Задание 1.7**

**Описание функционала выполнения кода:** если в качестве query параметра передан file={filename} выводит содержимое этого файла, иначе выводится страница index.html.

**Описание найденных уязвимостей:** *LFI*.В query параметр file передать значение абсолютного пути (например /etc/passwd или /root/.ssh/id\_rsa, если выполняется от имени root). В /etc/passwd можно перебрать пользователей и их директории и просмотреть их файлы (приватные ключи, конфиги).

**Пример эксплойта:**

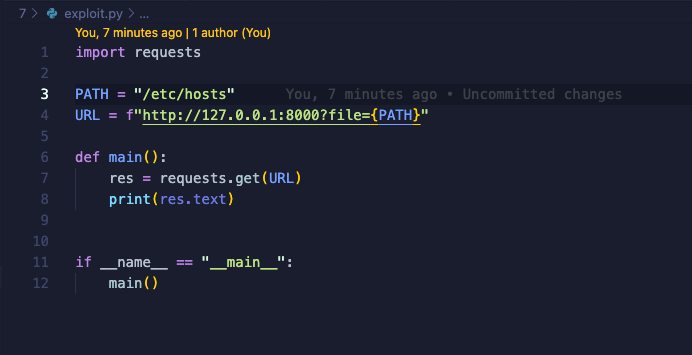
****

Рисунок . – пример кода эксплойта (7/exploit.py)

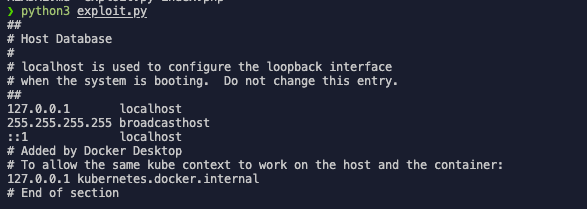


Рисунок . – эксплуатацию уязвимости

**Рекомендация по устранению уязвимости:** проводитьпроверку, что файл находится в директории проекта, например в php при помощи str\_starts\_with(filePath, projectPath)

**Задание 1.8**

**Описание функционала выполнения кода:** при открытии страницы выполняется перенаправление по переданному в query параметре r пути. Происходит базовая фильтрация пути, пытаясь избежать точек и слешей.

**Описание найденных уязвимостей:** *OpenRedirect и XSS*. Во-первых можно поэксплуатировать XSS, например так: http://127.0.0.1:8000/?r=javascript:alert(%27attacked%27), есть ограничение - точки внутри скрипта заменяются на \_, выражение должно быть без точек. Во-вторых можно использовать OpenRedirect, заменив стандартную точку китайской, например: http://127.0.0.1:8000/?r=https:xakep%E3%80%82ru

**Пример эксплойта:**

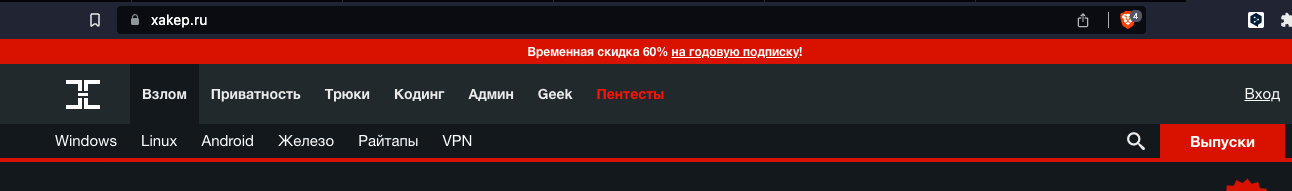
****

Рисунок . – эксплуатация уязвимости (перенаправление на сторонний ресурс)

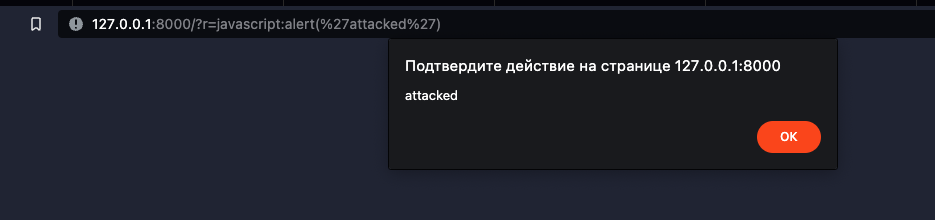


Рисунок . – эксплуатация уязвимости (выполнение javascript)

**Рекомендация по устранению уязвимости:** Проверять введенноеы в параметр r значение через белый список (константные строки или регулярное выражение), чтобы избежать XSS и OpenRedirect на нежелательные ресурсы.

**Задание 1.9**

**Описание функционала выполнения кода:** веб-приложение на Flask с использованием шаблонов. Обработчик пути /home.htmlберет query параметр search из запроса и ищет в базе данных продукты c этим параметром. Если продукты не найдены, выводит шаблон ошибки 404, в который подставляется значение из параметра search.

**Описание найденных уязвимостей:** *SSTI*. Подставленный в шаблон параметр можно представить в качестве исполняемого кода. Достаточно в качестве query параметра search например передать {{config.items()}} исходя из вывода можно узнать конфиг веб приложения, а там может содержаться информация о базе данных (хост, логин, пароль), тем самым можно выкрасть важные данные.

**Пример эксплойта:**

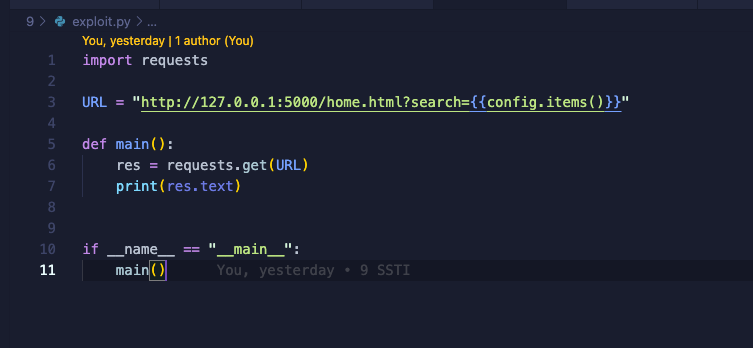
****

Рисунок . – пример кода эксплойта



Рисунок . – эксплуатация уязвимости

**Рекомендация по устранению уязвимости:** очищать query параметр search от {}, а лучше сделать возможными только [a-z, A-Z, 0-9].

**Задание 2**

**Описание функционала выполнения кода:** веб-приложениена Django с использованием Django ORM и базы данных PostgreSQL.

**Описание найденных уязвимостей:** *SQL Injection.*

**Пример эксплойта:**

**Рекомендация по устранению уязвимости:** проверять и очищать введенные пользователями данные.