

Разработка учебного приложения для демонстрации Path Traversal и небезопасного eval

Болтовский Гавриил Александрович

Приамурский государственный университет им. Шолом-Алейхема

Студент

Аннотация

Целью данной статьи является создание учебного веб-приложения для демонстрации уязвимостей Path Traversal и небезопасного eval. Программа написана на языке программирования Python с использованием фреймворка Flask, шаблонов Jinja2 и контейнеризации Docker для воспроизводимого запуска. Результатом исследования стала готовая программа с описанием структуры, примерами кода и заданиями CTF для проверки освоения материала.

Ключевые слова: учебное веб-приложение, веб-безопасность, Path Traversal, небезопасный eval

Development of an educational application to demonstrate Path Traversal and unsafe eval

Boltovsky Gavriil Alexandrovich

Sholom-Aleichem Priamursky State University

Student

Abstract

The purpose of this article is to create an educational web application to demonstrate Path Traversal vulnerabilities and unsafe eval. The program is written in Python using the Flask framework, Jinja2 templates, and Docker containerization for reproducible execution. The result of the research is a ready-made program with a description of the structure, code examples, and CTF tasks to test the mastery of the material.

Keywords: educational web application, web security, Path Traversal, unsafe eval

1 Введение

1.1 Актуальность исследования

В образовательной практике по защите программных систем часто отсутствует безопасное пространство, где начинающие могут увидеть уязвимость «вживую», понять её механику и границы ответственности разработчика. Две типичные проблемы – обход файловых путей и исполнение произвольных выражений – регулярно приводят к утечкам данных и несанкционированным операциям в реальных сервисах.

Создание контролируемого стенда с предсказуемым поведением и прозрачной логикой даёт возможность выстроить у студента причинно-следственную цепочку «ошибка проектирования → наблюдаемый эффект → корректный паттерн». Формат STF-заданий фиксирует минимум успеха (получение флага) и позволяет объективно проверить, что обучающийся не только прочитал теорию, но и разобрался в практическом аспекте эксплуатации и её предотвращения – строго в тестовой среде.

1.2 Обзор исследований

Совокупность источников фиксирует методические и технологические ориентиры развертывания учебной песочницы. И. В. Капгер, Е. Е. Журилова, А. А. Миронова конкретизируют требования к инфраструктуре и регламентам при создании учебного стенда, подчёркивая необходимость пошаговых инструкций и контроля доступа; эти положения релевантны внедрению песочницы на Flask с размещением на Docker-хостах [1]. Г. А. Воробьёв, И.И.Павленко обосновывают формат ситуационного центра и интеграцию с профильными лабораториями, в рамках которых программный стенд может выступать одним из рабочих полигонов для отработки сценариев [2]. Г.В.Беликов, И. Д. Крылов, В. А. Селищев систематизируют природу SQL-инъекции и подходы к противодействию; соответствующие материалы непосредственно транслируются в задания по демонстрации обхода авторизации и по последующему устранению уязвимости, включая переход от хранения паролей в открытом виде к хэшированию [3]. И. А. Лубкин описывает метрики оценки средств защиты от атак, основанных на возвратно-ориентированном программировании; указанные показатели целесообразно использовать для формализации оценивания результатов практикума и сопоставления вариантов конфигурации стенда [4].

1.3 Цель исследования

Сконструировать и описать учебное веб-приложение с намеренно оставленными уязвимостями обхода путей и вычисления произвольных выражений, пригодное для самостоятельной работы студентов и формализованной проверки результатов в формате STF.

1.4 Постановка задачи

Был спроектирован минимальный пользовательский сценарий: просмотр файлов журнала, вычисление выражения и чтение методических указаний, – в котором уязвимости проявляются как побочный эффект неверных проектных решений. Была сформирована структура проекта с явным разделением маршрутов, шаблонов и статических ресурсов, добавлены учебные данные и формулировки заданий, обеспечивающие проверяемый результат (флаг) без вмешательства преподавателя.

Были описаны границы безопасного использования стенда, подготовлены критерии оценки (получение конкретной строки-флага, определение имени пользователя контейнера) и организована

воспроизводимость запуска, чтобы студенты работали в идентичной среде и получали сопоставимые результаты. –

2 Методы исследования

В качестве каркаса использовался микро-фреймворк Flask благодаря его простоте маршрутизации и прозрачной интеграции с шаблонами Jinja2, что позволило быстро собрать интерфейс: список логов, просмотр содержимого, форма «калькулятора» и страница задания. Статические стили вынесены для отделения представления от логики, а учебные данные размещены в отдельном каталоге, включая демонстрационный файл и директорию с флагом для проверки задач. Намеренно небезопасные места оставлены в коде в учебных целях: чтение файла по параметру без нормализации пути и вычисление введённого выражения без песочницы – чтобы студенты наблюдали эффект и затем соотносили его с корректными практиками.

Для воспроизводимости и изоляции применялась контейнеризация (Dockerfile на базе python:3.11-slim) с фиксированными зависимостями (requirements.txt), что исключило различия окружений и позволило формировать проверяемые СTF-цели: чтение определённого файла и определение имени пользователя процесса внутри контейнера. Такой методический набор упрощает развёртывание стенда, задаёт чёткие точки входа и обеспечивает контролируемую, безопасную для инфраструктуры практику в тестовой среде.

Главную страницу приложения можно увидеть на рисунке (рис. 1)

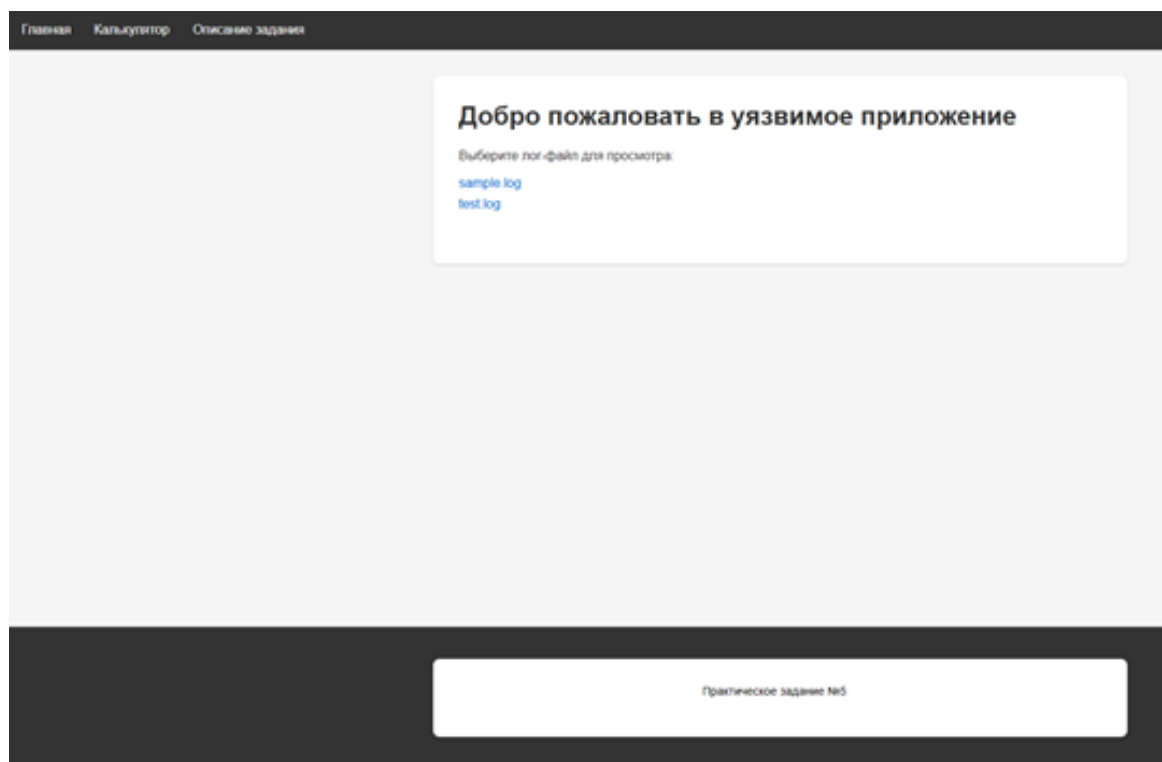


Рисунок 1 – Главная страница приложения

3 Результаты и обсуждения

Далее будут изложены принципы поэтапного конструирования учебного приложения и иллюстрирующие фрагменты кода. Все примеры предназначены исключительно для тестовой аудитории, в изоляции контейнера, и сопровождаются пояснением учебных границ.

Разумно начинать с минимального каркаса сервера и явного определения рабочей директории с журналами. Локальная структура задаёт единообразную точку входа, а шаблоны позволяют отделить представление от логики. Базовая инициализация приложения, секретного ключа и каталогов представлена на рисунке (рис. 2).

```
1 from flask import Flask, request, render_template, abort, send_from_directory, redirect, url_for, flash
2 import os
3
4 app = Flask(__name__)
5 app.secret_key = 'super_secret_key'
6
7
8 BASE_DIR = os.path.dirname(__file__)
9 LOG_DIR = os.path.join(BASE_DIR, 'logs')
10
11 @app.route('/')
12 def index():
13     files = [f for f in os.listdir(LOG_DIR) if os.path.isfile(os.path.join(LOG_DIR, f))]
14     return render_template('index.html', logs=files)
```

Рисунок 2 – Описание локальной структуры

Доступ к отдельному журналу организуется через параметр запроса. В учебных целях демонстрируется, как отсутствие нормализации и проверки допустимого пути приводит к чтению произвольных файлов в рамках прав процесса. Код намеренно оставлен без фильтрации для наблюдения эффекта и последующего обсуждения корректных паттернов защиты (рис. 3).

```
17 @app.route('/view_log')
18 def view_log():
19     filename = request.args.get('filename', '')
20     filepath = os.path.join(LOG_DIR, filename)
21     if not os.path.isfile(filepath):
22         abort(404)
23     with open(filepath, 'r', encoding="UTF8") as f:
24         content = f.read()
25     return render_template('logs.html', content=content, filename=filename)
```

Рисунок 3 – Фрагмент уязвимого кода приложения

Вторая дисциплинарная точка – вычисление произвольного выражения. Форма передаёт текст на сервер, где демонстрируется небезопасный вызов вычислителя. В учебной постановке это делается намеренно, чтобы студенты увидели, как прямое вычисление пользовательского ввода нарушает границы исполнения кода и почему такая конструкция недопустима в продуктивных системах (рис. 4).

```
28 @app.route('/calculate', methods=['GET', 'POST'])
29 def calculate():
30     result = None
31     expr = ''
32     if request.method == 'POST':
33         expr = request.form.get('expr', '')
34         try:
35             result = eval(expr) # Опасный eval
36         except Exception as e:
37             flash(f'Ошибка: {e}', 'danger')
38             return redirect(url_for('calculate'))
39     return render_template('calculate.html', result=result, expr=expr)
```

Рисунок 4 – Фрагмент кода с небезопасным eval

Пояснения и регламент задания удобнее вынести на отдельную страницу, чтобы в тексте формулировок зафиксировать цели, границы эксперимента и артефакты проверки. Это снижает нагрузку на преподавателя и повышает воспроизводимость (рис. 5).

```
42 @app.route('/assignment')
43 def assignment():
44     return render_template('assignment.html')
```

Рисунок 5 – Маршрут для пояснений и формулировкой задания

Шаблоны делают интерфейс прозрачным и минимальным, сохраняя единый навигационный каркас. Базовая разметка задаёт навигацию, контейнер и нижний колонтитул, а блок сообщений применим для краткой обратной связи при работе формы.

Для воспроизводимости и безопасности экспериментов удобно изолировать окружение с помощью контейнера. Описание образа фиксирует базовый интерпретатор, зависимости и команду запуска, тем самым выравнивая условия выполнения между студентами и упрощая автоматизацию проверки результатов (рис. 6).

```
1 FROM python:3.11-slim
2 ENV PYTHONDONTWRITEBYTECODE=1 PYTHONUNBUFFERED=1 FLASK_APP=app.py
3 WORKDIR /app
4 COPY requirements.txt .
5 RUN pip install --no-cache-dir -r requirements.txt
6 COPY . .
7 EXPOSE 5000
8 CMD ["flask", "run", "--host=0.0.0.0"]
```

Рисунок 6 – Описание контейнера

Минимальный перечень зависимостей ограничивается серверным каркасом и его системными библиотеками, что делает развёртывание предсказуемым и быстрым, а также снижает количество несущественных различий между локальными и контейнерными средами.

Построение такого стенда выполняет сразу две методические функции. Во-первых, студент наблюдает прямую причинно-следственную связь между проектным решением и наблюдаемым эффектом на уровне маршрутов, параметров и файловой системы. Во-вторых, формулируемые проверочные цели в виде извлекаемых артефактов в локальном контуре позволяют объективно фиксировать результат без обращения к внешним ресурсам. Обсуждение же корректных паттернов – нормализации и белых списков для путей, отказа от вычисления пользовательского кода и использования безопасных интерпретаторов – выстраивается уже после демонстрации, что подчёркивает учебный смысл эксперимента и не переносит опасные конструкции в продуктивную практику.

Ознакомиться с обучающим приложением можно в GitHub репозитории [5].

4 Выводы

Сконструирован и описан воспроизводимый учебный стенд – минимальное веб-приложение с намеренными уязвимостями обхода путей и небезопасного вычисления выражений.

Библиографический список

1. Капгер И. В., Журилова Е. Е., Миронова А. А. О разработке учебно-лабораторного стенда для изучения аппаратного модуля доверенной загрузки «Аккорд» // Вестник Пермского национального исследовательского политехнического университета. Электротехника, информационные технологии, системы управления. 2016. №. 17. С. 131-142.
2. Воробьев Г. А., Павленко И. И. Ситуационный центр в обучении техническим и гуманитарным аспектам информационной безопасности // Информационное противодействие угрозам терроризма. 2015. Т. 1. №. 25. С. 117-122.
3. Беликов Г. В., Крылов И. Д., Селищев В. А. SQL-инъекция как способ обхода авторизации // Известия Тульского государственного университета. Технические науки. 2021. №. 12. С. 217-221.
4. Лубкин И. А. Доклады Томского государственного университета систем управления и радиоэлектроники // Доклады Томского Государственного университета систем управления и радиоэлектроники Учредители: Томский государственный университет систем управления и радиоэлектроники. 2021. Т. 24. №. 4. С. 46-51.
5. GitHub course_materials URL: https://github.com/Gavriilbolt/course_materials/tree/main/01_basic_info_security/Prac%231software (дата обращения: 03.09.2025).