Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет прикладной информатики

Дисциплина:

«Основы кибербезопасности»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

«Web Application Firewall»

Выполнил:

Швалов Даниил Андреевич, студент группы K4112c

_____

(подпись)

Проверил:

Кравчук Алексей Владимирович, доцент практики

_____

(отметка о выполнении)

_____

(подпись)

Санкт-Петербург
2025 г.

СОДЕРЖАНИЕ

1 Введение

Цель работы:

1) ознакомиться с принципами работы WAF;

2) научиться настраивать и тестировать WAF на примере open-source решений;

3) получить практический опыт анализа логов и правил фильтрации http(s)-трафика.

2 Ход работы

2.1 Настройка стенда

В начале выполнения лабораторной работы был загружен образ с WAF Coraza. Данный образ был распакован и импортирован в Virtual Box. Это видно на рисунке 1.

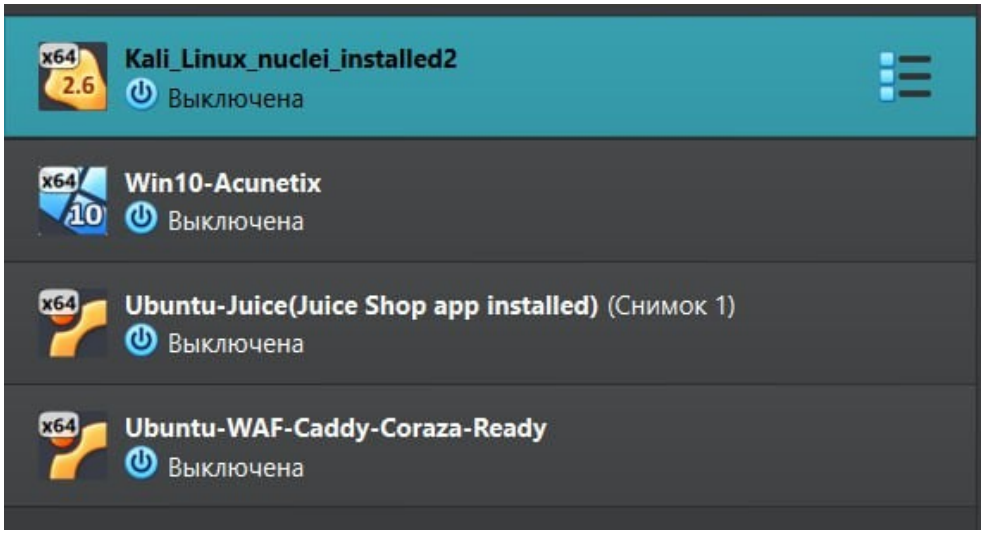


Рисунок 1 — Виртуальные машины, используемые в лабораторной работе

2.2 Проверка работоспособности стенда

После этого была выполнена проверка работоспособности стенда. Для этого на виртуальной машине с Kali Linux с помощью утилиты curl был отправлен HTTP-запрос в WAF. Как видно на рисунке 2, WAF вернул 200 HTTP-код, что означает, что стенд был поднят и настроен верно.

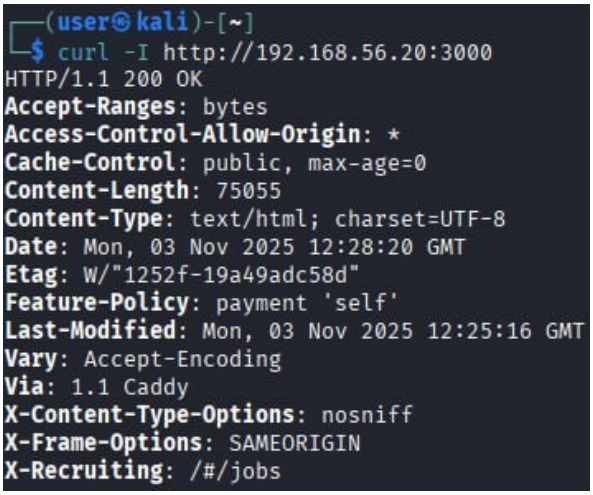


Рисунок 2 — Тестовый HTTP-запрос в WAF

После этого с помощью утилиты curl была загружена главная страница Juice Shop. Также с помощью утилиты awk в загруженном HTML-документе был переопределен базовый URL-адрес, чтобы ссылки посылали запросы через WAF, а не напрямую к Juice Shop. Этот процесс представлен на рисунке 3.

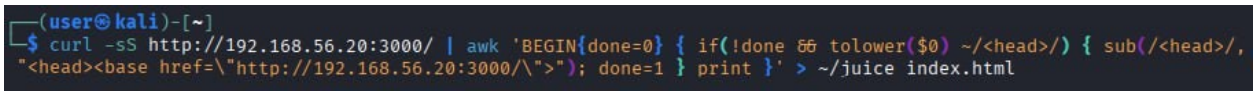


Рисунок 3 — Загрузка и изменение главной страницы Juice Shop

После этого загруженная страница была открыта в браузере. Как видно на рисунке 15, страница смогла частично загрузится: на ней видны основные элементы управления.

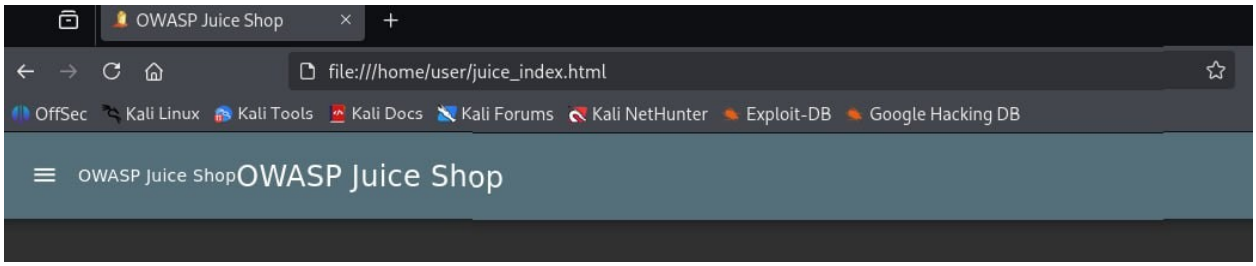


Рисунок 4 — Загруженная страница

В случае, если загрузить главную страницу и не подменить базовый URL-адрес, как это сделано на рисунке 5, то в таком случае страница не

сможет загрузится. Это видно на рисунке 6.

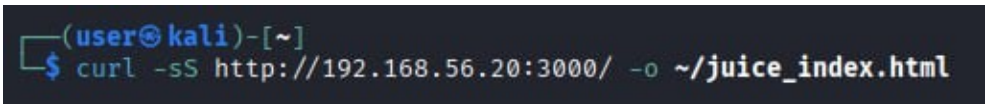


Рисунок 5 — Загрузка главной страницы Juice Shop без изменений

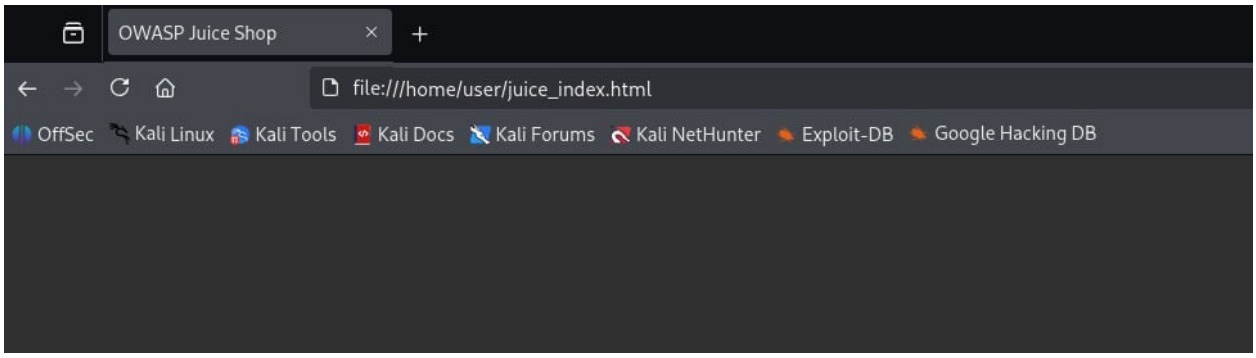


Рисунок 6 — Не загружающаяся главная страница Juice Shop

После этого с помощью утилиты curl была предпринята попытка встраивания XSS. Как видно на рисунке 7, WAF не пропустил такой запрос.

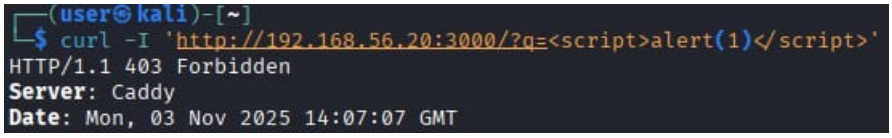


Рисунок 7 — Попытка встраивания XSS

После выполнения запроса с XSS информация о данном запросе также появилась в логах Caddy. Как видно на рисунке 8, в логе содержится достаточно подробная информация о запросе, а именно:

1) дата и время, когда совершен запрос;

2) IP-адрес клиента;

3) содержимое запроса: заголовки, параметры, тело, метод запроса;

4) набор правил, в соответствии с которыми был отклонен запрос.

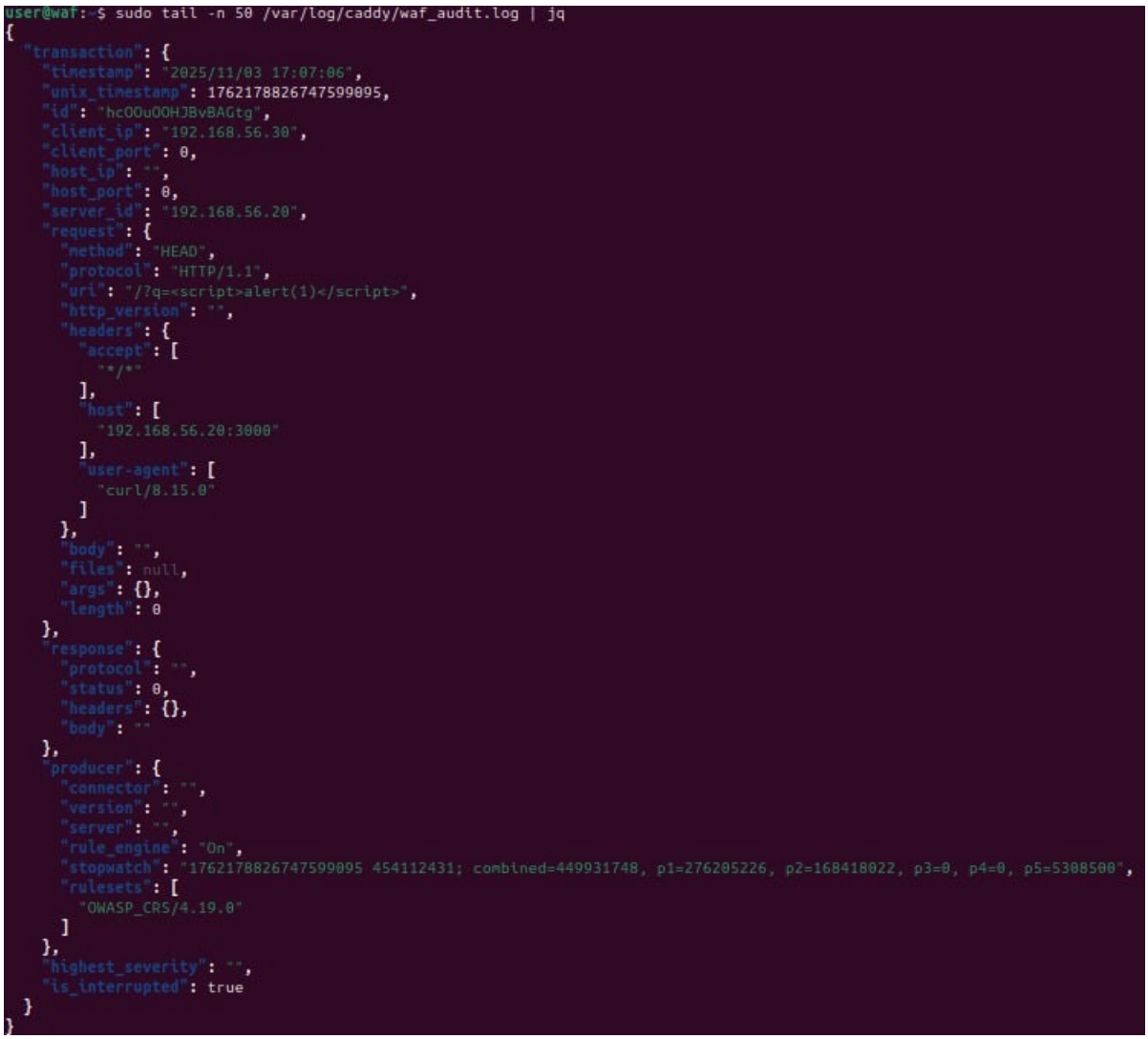


Рисунок 8 — Лог заблокированного запроса с XSS

После этого была выполнена попытка реализовать SQL-инъекцию. Как видно на рисунке 9, данный запрос также был заблокирован WAF.

Рисунок 9 — Запрос с SQL-инъекцией

Как видно на рисунке 10, в логах Caddy также видна информация о данном запросе, в т. ч. и сама SQL-инъекция, которая была отправлена клиентом.

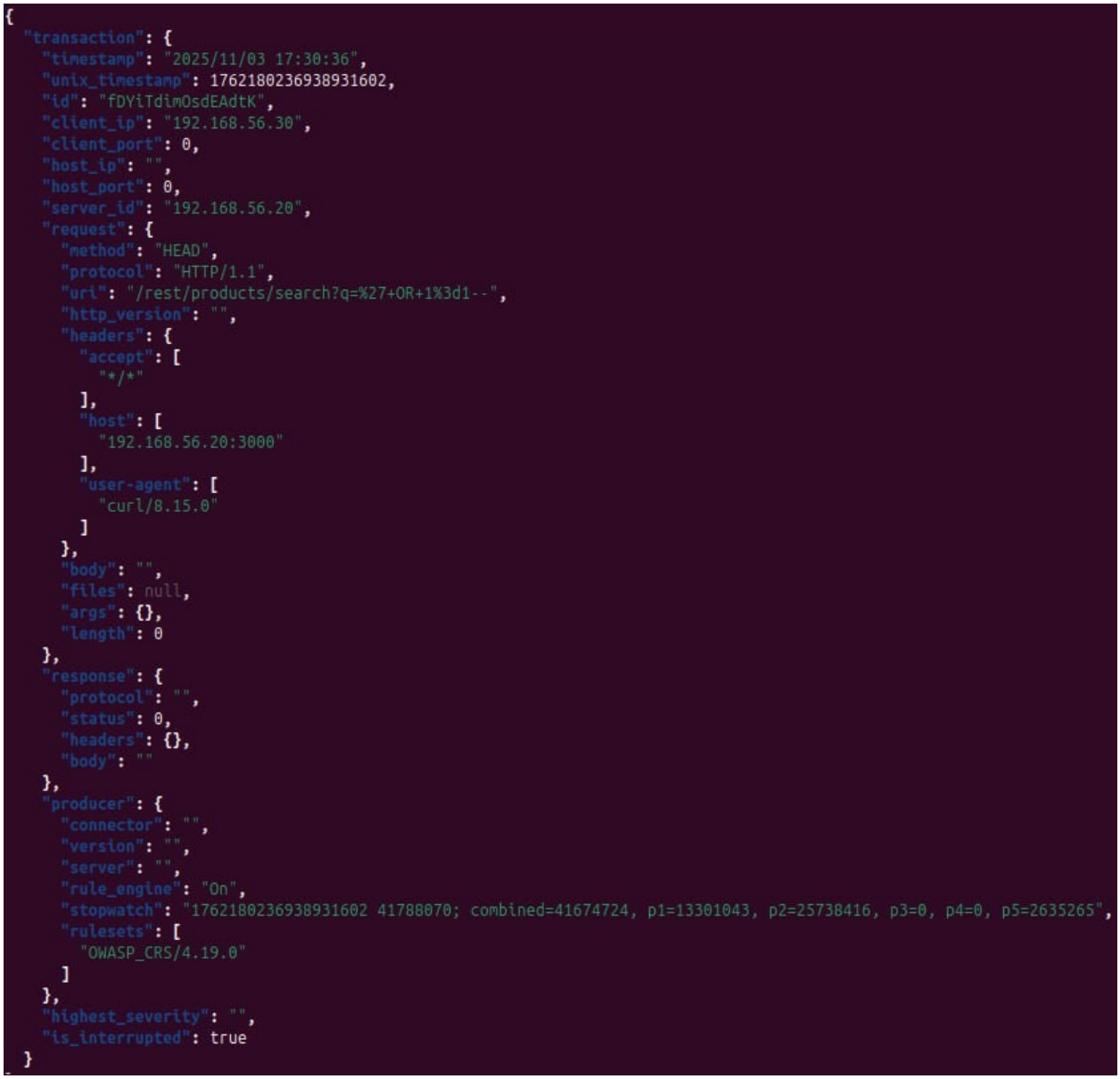


Рисунок 10 — Лог заблокированного запроса с SQL-инъекцией

После этого была выполнена попытка внедрить SQL-инъекцию не в параметры, а в тело запроса. Как видно на рисунке 11, подобный запрос был также отклонен WAF.

Рисунок 11 — Запрос в SQL-инъекцией в теле запроса

2.3 Сканирование Juice Shop с включенным WAF Coraza

После этого на виртуальной машине с Windows был запущен Acunetix. В Acunetix была создана новая цель, IP-адрес которой соответствует IP-адресу виртуальной машины с WAF. Это видно на рисунке 12.

Рисунок 12 — Создание цели с IP-адресом WAF

После создания цели было запущено сканирование. Как видно на рисунке 13, Acunetix смог найти 1 критичную, 9 средний, 2 низких и 6 информационных уязвимостей. Из них наиболее интересными являются:

1) Acunetix удалось реализовать SQL-инъекцию: теперь сложно или невозможно напрямую поменять данные в БД, однако все еще можно вызвать ошибки при работе с БД;

2) Juice Shop все еще позволяет доступ по HTTP вместо HTTPS;

3) Acunetix смог обнаружить устаревшие JavaScript библиотеки, потенциально допускающие XSS;

4) токены и секреты все еще доступны без необходимости обладать какими-либо привилегиями (например, LinkedIn и GitLab токены);

5) у злоумышленников все еще есть доступ к Prometheus метрикам;

6) Juice Shop все еще не использует некоторые HTTP-заголовки, без которых злоумышленник может эксплуатировать Juice Shop для проведения атак (например, фишинговых).

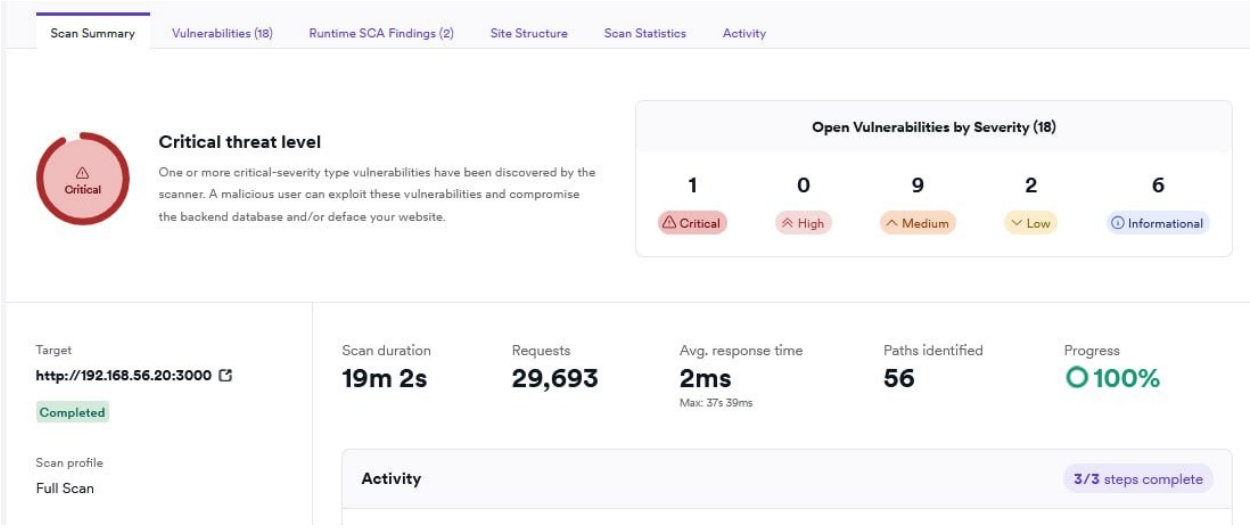


Рисунок 13 — Результаты сканирования Juice Shop с WAF

Таким образом можно сделать вывод, что WAF смог сильно уменьшить поверхность атаки, но он все еще не защищает от всех видов атак. Некоторые из уязвимостей сложно или невозможно закрыть с помощью WAF, т. к. данные уязвимости можно исправить только в самом приложении Juice Shop.

2.4 Способы обхода WAF и противодействие им

После этого на виртуальной машине с Kali Linux была выполнена попытка обойти WAF с помощью использования URL кодирования. В первую очередь был отправлен запрос без URL кодирования. Как видно на рисунке 14, WAF не пропустил данный запрос.

```
┌──(user㉿kali)-[~]
└─$ curl -i "http://192.168.56.20:3000/?q=<script>alert(1)</script>"
HTTP/1.1 403 Forbidden
Server: Caddy
Date: Tue, 04 Nov 2025 09:24:02 GMT
Content-Length: 0
```

Рисунок 14 — Запрос с XSS без URL кодирования

Затем в запросе специальные символы, такие как «<», «>», «/» были заменены на соответствующие им представления в шестнадцатеричной системе. Как видно на рисунке 15, WAF также не пропустил данный запрос.

Рисунок 15 — Запрос с XSS с использованием URL кодирования

После этого в запросе было использовано смешанное URL кодирование, однако как и прежде WAF не пропустил данный запрос. Это видно на рисунке 16.

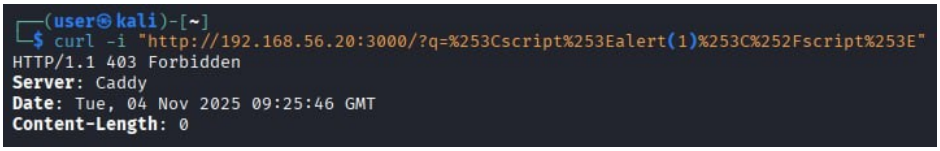
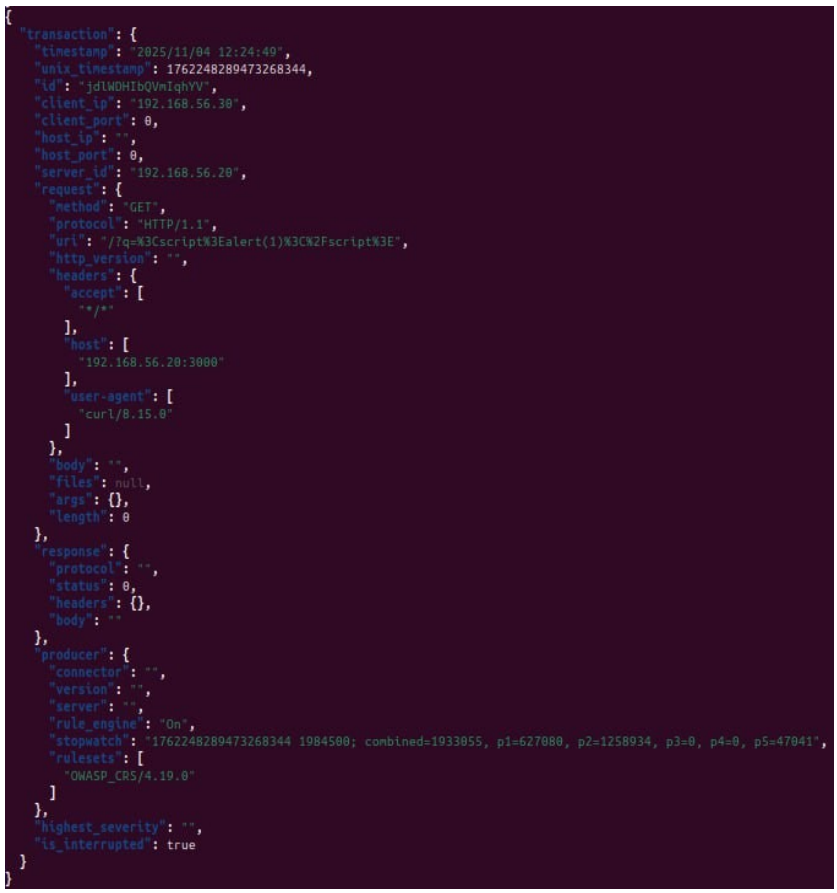


Рисунок 16 — Запрос с XSS с использованием смешанного URL кодирования

В логах Caddy также видно, что эти запросы были заблокированы в соответствии с набором правил OWASP_CRS. Логи с данной информацией представлены на рисунках 17-19.

Рисунок 17 — Лог с запросом без использования URL кодирования



```
{
  "transaction": {
    "timestamp": "2025/11/04 12:24:49",
    "unix_timestamp": 1762248289473268344,
    "id": "jdlWDHIbQVmIqhYV",
    "client_ip": "192.168.56.30",
    "client_port": 0,
    "host_ip": "",
    "host_port": 0,
    "server_id": "192.168.56.20",
    "request": {
      "method": "GET",
      "protocol": "HTTP/1.1",
      "uri": "/?q=%3Cscript%3Ealert(1)%3C%2Fscript%3E",
      "http_version": "",
      "headers": {
        "accept": [
          "*/*"
        ],
        "host": [
          "192.168.56.20:3000"
        ],
        "user-agent": [
          "curl/8.15.0"
        ]
      },
      "body": "",
      "files": null,
      "args": {},
      "length": 0
    },
    "response": {
      "protocol": "",
      "status": 0,
      "headers": {},
      "body": ""
    },
    "producer": {
      "connector": "",
      "version": "",
      "server": "",
      "rule_engine": "On",
      "stopwatch": "1762248289473268344 1984500; combined=1933055, p1=627080, p2=1258934, p3=0, p4=0, p5=47041",
      "rulesets": [
        "OWASP_CRS/4.19.0"
      ]
    },
    "highest_severity": "",
    "is_interrupted": true
  }
}
```


Рисунок 18 — Лог с запросом с использованием URL кодирования

Рисунок 19 — Лог с запросом с использованием смешанного URL кодирования

После этого была предпринята попытка поместить XSS в тело запроса. Как видно на рисунке 20, WAF не пропустил подобный запрос.

Рисунок 20 — XSS в теле запроса

Как видно из лога, представленного на рисунке 21, запрос также был заблокирован с соответствии с набором правил OWASP_CRS.

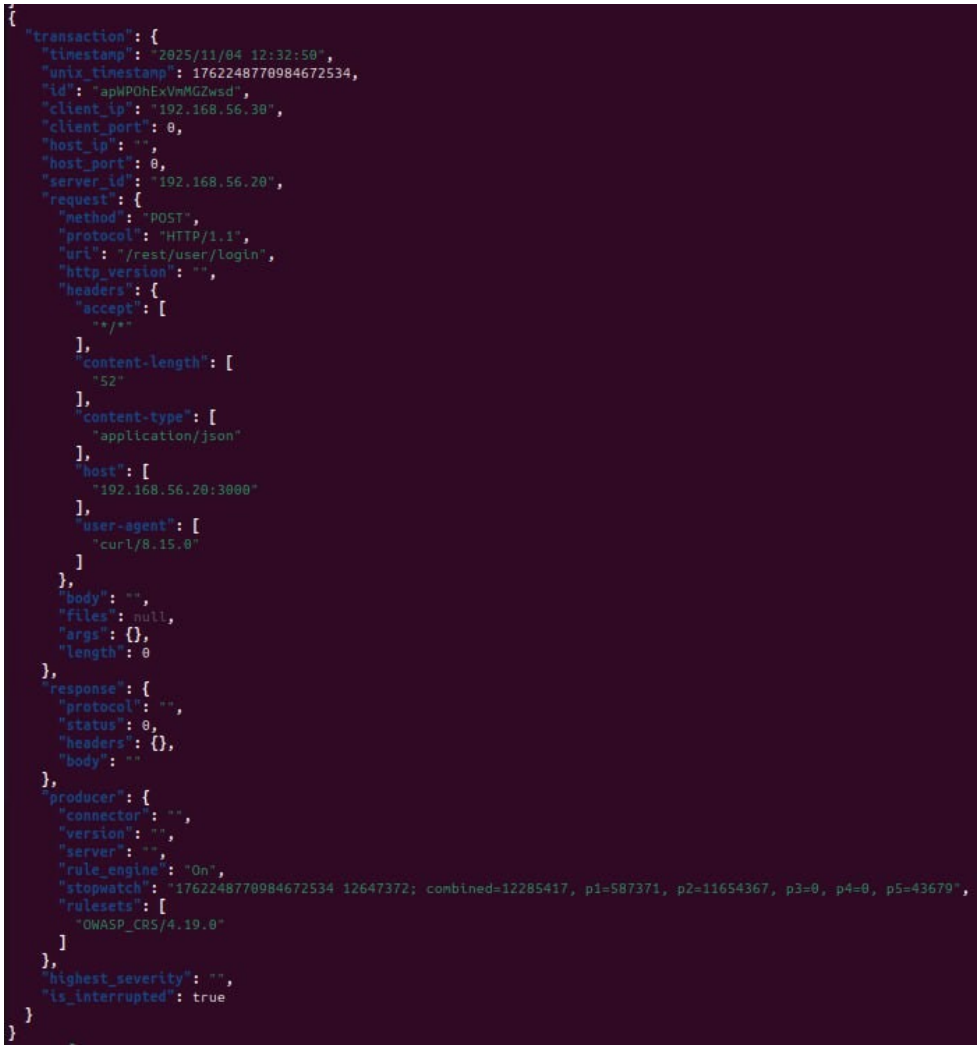


Рисунок 21 — Лог с запросом с использованием XSS в теле запроса

После этого была предпринята попытка отправить запрос с XSS в теле запроса с использованием URL кодирования. Однако как видно на рисунке 22, данный запрос также был не пропущен WAF.

Рисунок 22 — XSS в теле запроса с URL кодированием

После этого была предпринята попытка отправить запрос с XSS в заголовке. Сначала XSS был встроен в заголовок User-Agent. Как видно на рисунке 23, WAF не пропустил такой запрос.

Рисунок 23 — XSS в заголовке User Agent

Также на рисунке 24 видно, что в данном случае также был применен набор правил OWASP_CRS.

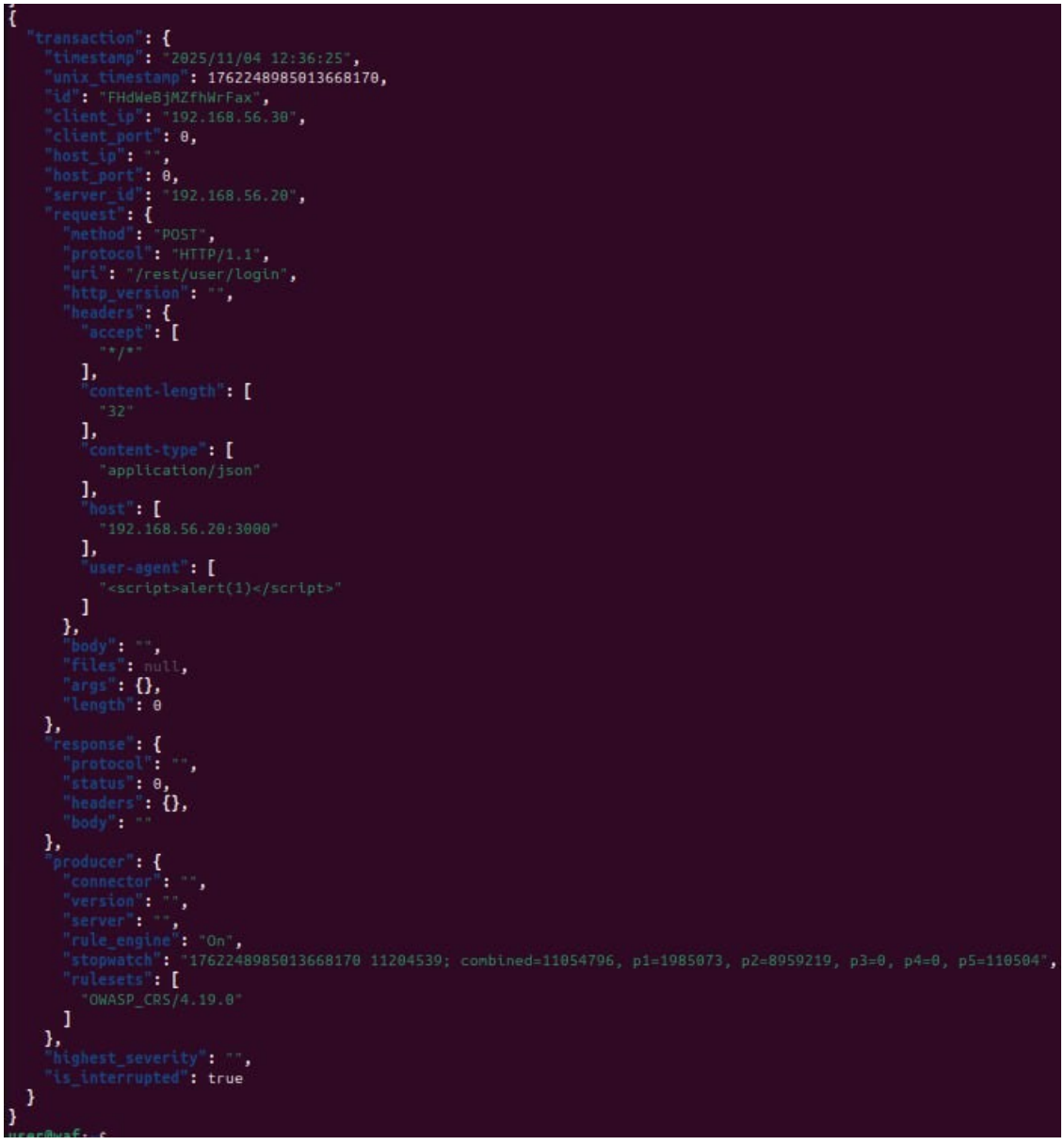


Рисунок 24 — Лог с запросом с использованием XSS в заголовке User-Agent

После этого была выполнена попытка встроить XSS через заголовок

User-Agent с использованием URL кодирования. Однако, как видно на рисунке 25, этот запрос был также заблокирован WAF.

Рисунок 25 — XSS в заголовке User Agent с использованием URL кодирования

Затем была выполнена попытка встроить XSS в другой заголовок, т. е. в заголовок X-Custom-Header. Как видно на рисунке 26, XSS был успешно пропущен WAF. Из этого можно сделать вывод, что Caddy проверяет не все заголовки и допускает XSS в некоторых из них (в частности, в пользовательских заголовках).

Рисунок 26 — XSS в заголовке X-Custom-Header

После этого была выполнена попытка встроить XSS в куки. Как видно на рисунке 27, WAF не пропустил такой запрос.

Рисунок 27 — XSS в куки

Заблокированный запрос с XSS в куки также виден в логах Caddy, представленных на риснуке 28.

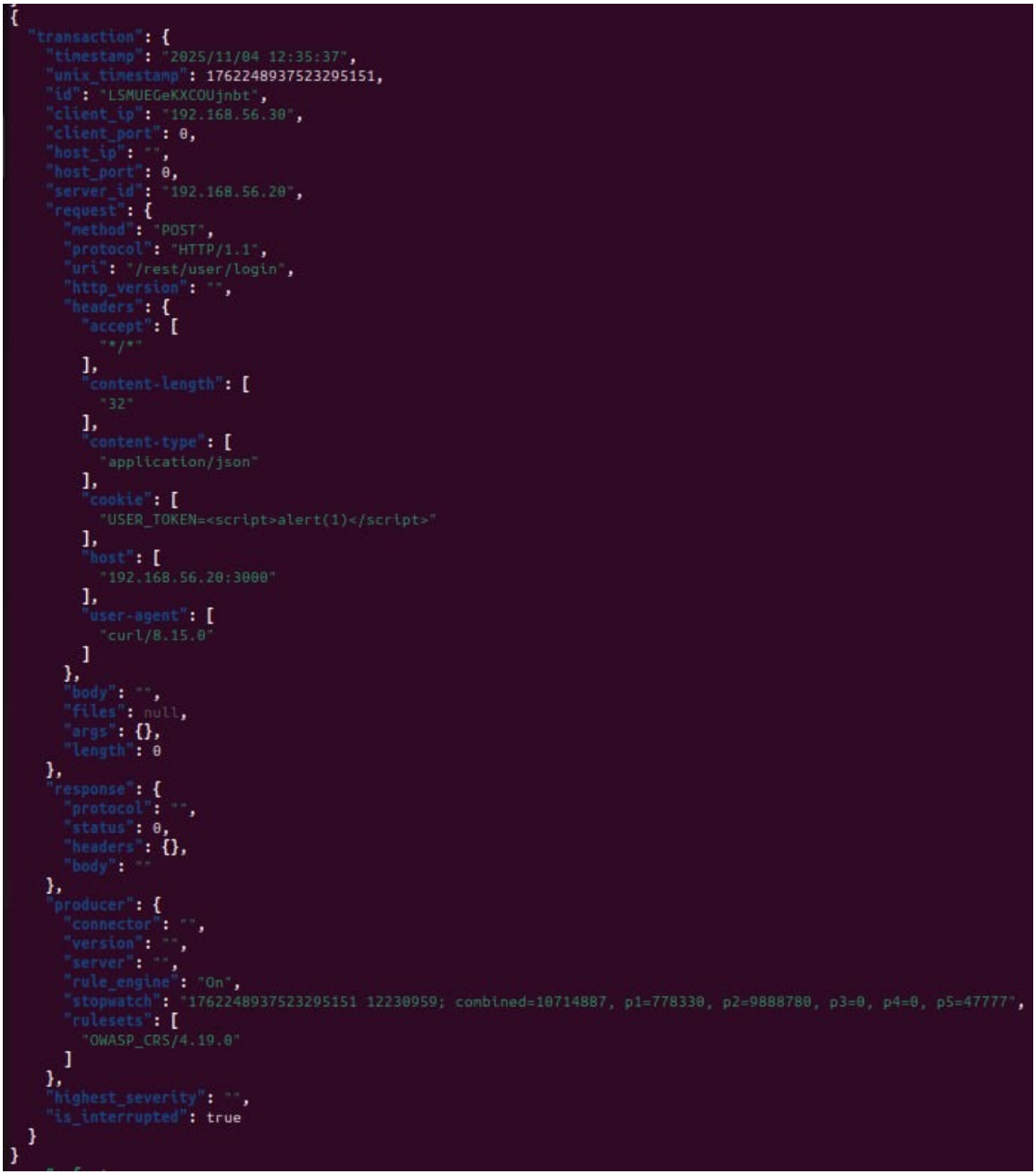


Рисунок 28 — Лог с запросом с использованием XSS в куки

Затем была предпринята попытка встроить XSS с использованием URL кодирования. Однако, как видно на рисунке 29, и такой запрос был отклонен WAF.

Рисунок 29 — XSS в куки с использованием URL кодирования

3 Вывод

В ходе выполнения данной лабораторной работы были получены навыки работы с WAF, навыки настройки и тестирования WAF на примере open-source решений, практический опыт анализа логов и правил фильтрации http(s)-трафика.