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

Факультет прикладной информатики (ФПИН)

Лабораторная работа №7

по дисциплине: Основы кибербезопасности

Автор: доцент практики, кандидат технических наук

Кравчук Алексей Владимирович

Санкт-Петербург
2025

Тема занятия: Web Application Firewall (WAF).

Цель работы:

- Ознакомиться с принципами работы WAF.
- Научиться настраивать и тестировать WAF на примере open-source решений (reverse proxy server Caddy + модуль WAF Coraza .
- Получить практический опыт анализа логов и правил фильтрации htpp(s)-трафика.

Краткие теоретические сведения

Данная лабораторная работа (далее - лаба) является логическим продолжением лабы № 5, в которой вы отработали навыки тестирования уязвимого web-приложения (Juice Shop) с помощью сканеров уязвимостей Acunetix (DAST) и Nuclei.

В этой лабе вам предстоит применить стратегию «виртуального патчинга» (см. лекцию 3, слайд 7) путем проксирования HTTP(S)-трафика через reverse proxy сервер **Caddy** с установленным модулем **WAF Coraza**.

Архитектура стенда, с которым вам предстоит поработать, приведена на рисунке 1.

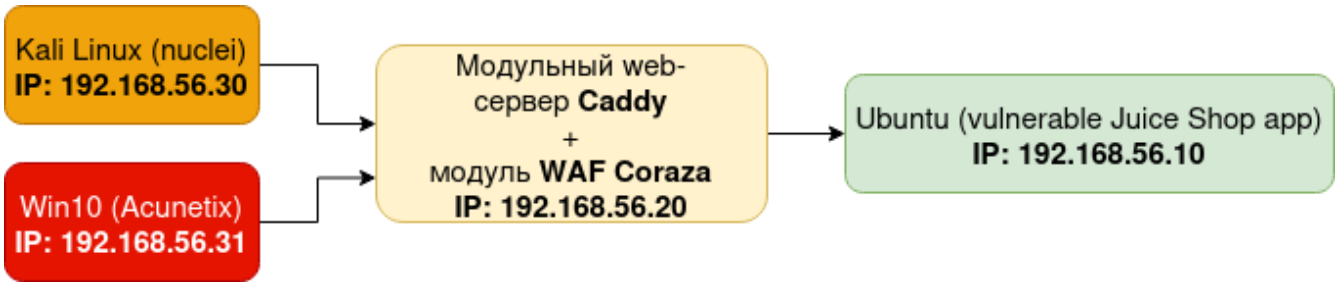


Рисунок 1. Архитектура стенда.

Тему WAF будем раскрывать на примере open-source решений посредством использования reverse proxy сервера **Caddy** с установленным модулем **WAF Coraza**.

Web Application Firewall (WAF) — это программный или аппаратный фильтр, контролирующий HTTP/HTTPS-трафик между клиентом и веб-приложением.

Основные характеристики **WAF Coraza**;

1. новый WAF-движок, написанный на Golang и заменивший ModSecurity (написан на C/C++);
2. быстрее и легче, чем ModSecurity (на Go, без C/C++ зависимостей);
3. совместим с правилами ModSecurity, включая OWASP Core Rule Set (**OWASP CRS**), но проще и быстрее;
4. удобен для использования в DevSecOps-пайплайнах;

В процессе выполнения лабы вы столкнетесь со следующими тремя компонентами open-source WAF, представленными в таблице 1.

Таблица 1 – Используемые в лабе №7 компоненты

Компонент	Описание
Caddy	Веб-сервер и reverse proxy. Управляет маршрутизацией HTTP-трафика.
Coraza	Сам WAF-движок. Анализирует HTTP-запросы и решает: пропустить или заблокировать.
OWASP CRS	Набор правил, по которым Coraza определяет атаки.

WAF Coraza встраивается в веб-сервер **Caddy**, который может работать в двух режимах:

1. как обычный веб-сервер: сам отвечает на HTTP(S)-запросы;
2. как reverse proxy (обратный прокси): принимает запросы от клиентов и пересылает их на другой сервер (в нашем случае – это web-app Juice Shop)

Основные функции **WAF Coraza**:

- Фильтрация HTTP(S) трафика;
- Логирование подозрительных запросов.

Практическая часть

1. Установка WAF (Caddy + Coraza)

Обновляем список доступных пакетов и устанавливаем необходимые пакеты:

sudo apt update && sudo apt install -y curl git wget unzip build-essential

1.1. Установка Go

Go устанавливаем вручную, т.к. в официальных репозиториях Ubuntu, как правило, более старые версии. Для установки Caddy v2.10.x нам нужна версия Go ≥ 1.25:

cd
wget https://go.dev/dl/go1.25.3.linux-amd64.tar.gz
sudo tar -C /usr/local -xzf go1.25.3.linux-amd64.tar.gz

Проверяем командой ***go version***, что golang нужной нам версии установлен.

```
user@waf:~$ go version
go version go1.25.3 linux/amd64
user@waf:~$
```

Создадим каталог, куда пакетный менеджер Go по умолчанию будет устанавливать исполняемые файлы (когда ставим их через **go install**):

mkdir -p ~/go/bin

Добавляем 2 пути к переменной PATH, чтобы система знала, где искать установленные утилиты:

echo "export PATH=/usr/local/go/bin:$HOME/go/bin:$PATH ">> ~/.bashrc
source ~/.bashrc

/usr/local/go/bin	здесь у нас размещен исполняемый файл **go**
/home/user/go/bin	здесь у нас будет лежать исполняемый файл **caddy** (а также все утилиты, которые мы устанавливаем через **go install**)

Проверяем, что PATH обновился:

echo $PATH

1.2. Установка xcaddy и сборка Caddy с модулем WAF Coraza

Предварительно вам потребуется установить утилиту xcaddy.

xcaddy – это утилита от разработчиков Caddy, предназначенная для сборки Caddy с нужными плагинами (модулями).

Без **xcaddy** вы получите «чистую» сборку Caddy, в которой есть только базовые модули без сторонних плагинов. Нам же нужен Caddy с модулем WAF Coraza.

Устанавливаем xcaddy (без sudo она установится в **~/go/bin**):

go install github.com/caddyserver/xcaddy/cmd/xcaddy@latest

Собираем Caddy с модулем Coraza (по умолчанию пакет будет собран в текущем каталоге: /home/${USER}/caddy/):

xcaddy build --with github.com/corazawaf/coraza-caddy/v2

Для удобства использования перемещаем собранный исполняемый файл в системный каталог. По правилам **FHS** (Filesystem Hierarchy Standard) локально собранные исполняемые файлы следует размещать в /usr/local/bin

sudo mv ./caddy /usr/local/bin/caddy

Устанавливаем суперпользователя (root) и его группу владельцами файла в целях безопасности.

sudo chown root:root /usr/local/bin/caddy

Проверим версию и наличие модуля:

caddy version

Для убедительности можно проверить список модулей:

caddy list-modules | grep waf # в выводе должен быть http.handlers.waf

```
user@waf:~$ caddy version
v2.10.2 h1:g/gTYjGMD0dec+UgMw8SnfmJ3I9+M2TdvoRL/Ovu6U8=
user@waf:~$
user@waf:~$
user@waf:~$ caddy list-modules | grep waf
http.handlers.waf
user@waf:~$
```

Создадим сервисного (системного) пользователя **caddy**, под которым можно безопасно запускать web-сервер, а также создадим структуру директорий для Caddy (чтобы запускать наш WAF как сервис):

sudo useradd \
--system \
--gid *nogroup* \
--create-home \
--home-dir */var/lib/caddy* \
--shell */usr/sbin/nologin* \
--comment *"Caddy web server"* \
caddy

Разберём параметры запуска команды:

sudo	Выполняем команду с правами root.
useradd	Низкоуровневая утилита для создания пользователей
--system	Создаём системного пользователя: ● его UID берётся из диапазона системных пользователей (обычно <1000); ● такой пользователь не предназначен для интерактивного входа, пароля у него нет; ● используется для запуска сервисов/демонов.
--gid nogroup	Назначаем первичную группу nogroup (на Debian/Ubuntu это группа с минимальными правами, аналог «nobody» для группы).
--create-home	Явно создаём домашний каталог для пользователя (по умолчанию для **system** он может не создаваться).
--home-dir /var/lib/caddy	Указываем путь к домашнему каталогу: /var/lib/caddy.
--shell /usr/sbin/nologin	Запрещаем интерактивный вход (SSH/tty).
--comment “Caddy web server”	Комментарий к учетной записи. Виден в выводе команды ***getent passwd caddy***.
caddy	Имя пользователя, которого создаём.

В выводе команды ***getent passwd caddy*** вы должны увидеть:

```
user@waf:~$ getent passwd caddy
caddy:x:995:65534:Caddy web server:/var/lib/caddy:/usr/sbin/nologin
```

Мы запускаем Caddy под отдельной учетной записью с минимальными правами (**principle of least privilege**), а не под root.

Если в процессе работы веб-сервера что-то пойдёт не так (RCE/эксплойт), злоумышленник получит минимальные права только пользователя **caddy**.

Выдаем права на каталоги, куда Caddy будет писать, изолируя доступ к остальной части системы:

sudo mkdir -p /var/log/caddy
sudo chown -R caddy:nogroup /var/lib/caddy /var/log/caddy

1.3. Установка OWASP CRS и настройка WAF Coraza

1.3.1. Создаём рабочую структуру и скачиваем CRS

Цель этого шага – установить OWASP Core Rule Set (CRS), подготовить конфигурацию Coraza и подключить правила CRS к WAF Coraza, чтобы WAF мог детектировать и блокировать типичные веб-атаки (SQLi, XSS, LFI, RCE и др.).

Создайте каталог /etc/caddy/coreruleset и перейдите в него:
sudo mkdir -p /etc/caddy/coreruleset
cd /etc/caddy/coreruleset

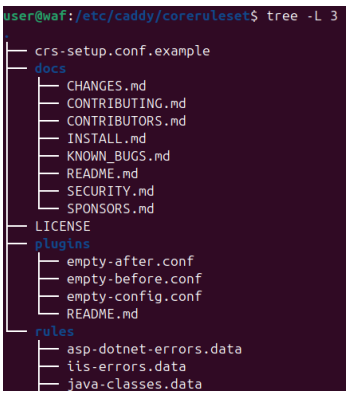
Перейдите на официальный сайт OWASP CRS Project https://coreruleset.org/ и скачайте последнюю версию набора общих правил для обнаружения атак, распакуйте их и разместите в ***/etc/caddy/coreruleset/***

Также вы можете скачать CRS с помощью wget (из каталога ***/etc/caddy/coreruleset/***):

***sudo wget -O coreruleset-4.19.0-minimal.tar.gz ***
https://github.com/coreruleset/coreruleset/releases/download/v4.19.0/coreruleset-4.19.0-minimal.tar.gz

Распаковываем архив и помещаем его содержимое в ***/etc/caddy/coreruleset/ :***
sudo tar -xzf coreruleset-4.19.0-minimal.tar.gz
sudo rm coreruleset-4.19.0-minimal.tar.gz
sudo mv coreruleset-4.19.0/* ./
sudo rm -rf coreruleset-4.19.0/

Убедитесь, что структура файлов и каталогов похожа на следующую:



1.3.2. Настраиваем WAF-движок Coraza

Для WAF Coraza можно скачать базовый конфигурационный файл **coraza.conf-recommended**.

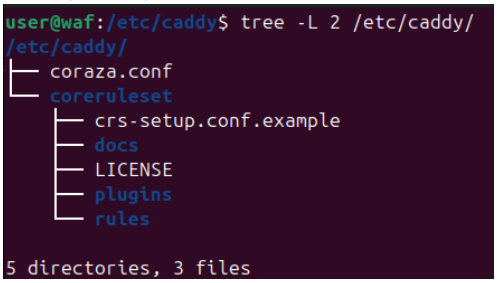
Для его скачивания воспользуемся доменом **githubusercontent.com**, который GitHub использует для раздачи файлов, а не для отображения веб-страниц репозитория.

cd /etc/caddy/

***sudo wget -O coraza.conf ***

https://raw.githubusercontent.com/corazawaf/coraza/v2/master/coraza.conf-recommended

В результате структура файлов и каталогов в /etc/caddy должна быть похожа на следующую:



Откройте файл **coraza.conf** в текстовом редакторе (например, **nano**) и посмотрите на его ключевые параметры.

Этот конфигурационный файл определяет:

- как WAF обрабатывает входящие данные;
- какая информация будет помещаться в log-файлы;
- какие действия по умолчанию будут применяться (deny/pass/log).

1.3.3 Настраиваем Caddy как Reverse Proxy + WAF

Определим конфигурацию веб-сервера Caddy в файле **/etc/caddy/Caddyfile**.

Этот файл определяет:

- какие порты слушать (:3000);
- куда перенаправлять запросы (reverse_proxy)ж
- когда и как включать WAF (coraza_waf);

- какие конфигурации WAF загружать (directives, Include).

Создадим конфигурационный файл **/etc/caddy/Caddyfile**:
sudo touch /etc/caddy/Caddyfile

Откроем его в текстовом редакторе (например, nano) и вставим в него следующее содержимое:

```
{
        order coraza_waf first
}

:3000 {
        coraza_waf {
                load_owasp_crs
                directives <<EOT
            Include /etc/caddy/coraza.conf
            Include /etc/caddy/coreruleset/crs-setup.conf
            Include /etc/caddy/coreruleset/rules/*.conf

            SecRuleEngine On
            SecRequestBodyAccess On
            SecAuditLog /var/log/caddy/waf_audit.log
            SecAuditLogFormat JSON
            SecAuditLogParts ABIJDEFHZ
EOT
        }
        reverse_proxy 192.168.56.10:3000
}
```

Разберём этот файл:

В секции { ... } заданы глобальные параметры Caddy.

order coraza_waf first означает, что middleware-плагин Coraza WAF должен применяться к запросам перед другими обработчиками (здесь перед прокси). Это обязательная опция, без неё Coraza не перехватит запросы.

:3000 { ... } – здесь мы настраиваем виртуальный хост, прослушиваемый порт 3000 на всех интерфейсах (так как IP не указан, по умолчанию на 0.0.0.0:3000). Таким образом, WAF будет также доступен по адресу 192.168.56.20:3000 – как и Juice Shop, только на другом IP.

Директива **coraza_waf** включает WAF. Параметр **load_owasp_crs** активирует режим использования CRS.

Строки конфигурации, передаваемые движку Coraza, используют синтаксис **heredoc** (<<EOT ... EOT).

Include /etc/caddy/coraza.conf подключает базовую конфигурацию Coraza (рекомендуемый конфиг). В нём определены общие настройки WAF.
По умолчанию для опции **SecRuleEngine** используется значение DetectionOnly, но непосредственно при выполнении лабы нужно будет переключить **SecRuleEngine** в значение On.

Include /etc/caddy/coreruleset/crs-setup.conf подключает настройки CRS.

Include /etc/caddy/coreruleset/rules/*.conf подключает все правила из набора CRS. Это более 100 файлов правил, покрывающих различные атаки.

SecRuleEngine On – ключевая строка, включающая режим блокировки правил. Без неё движок останется в режиме DetectionOnly и не будет блокировать запросы.

SecRequestBodyAccess On заставляет WAF анализировать не только заголовки, но и тела запросов. Это нужно для анализа POST-запросов к Juice Shop, поскольку в POST-запросах параметры (а это может быть и вредоносный код) передаются в теле запроса.

Примечание: Juice Shop – это single-page application (SPA), в котором вся логика общения с сервером идёт через JSON-API и если SecRequestBodyAccess будет иметь значение Off, то 90% атак на Juice Shop пройдут незамеченными.

SecAuditLog /var/log/caddy/waf_audit.log – файл, куда Coraza будет писать детальные логи по срабатываниям правил.

SecAuditLogFormat JSON для использования формата JSON.

SecAuditLogParts ABIJDEFHZ определяет что именно будет записываться в WAF-аудит лог, т.е. то, на основании чего вы будете производить анализ атак на Juice Shop. В директиве **SecAuditLogParts ABIJDEFHZ** каждая буква соответствует определенной **секции лог-файла** (приведены в таблице 2).

Таблица 2 – Расшифровка **SecAuditLogParts**

Буква	Имя секции	Что включает	Зачем полезно
A	AuditLog Header	Метаданные запроса (IP, timestamp, UID)	Понимание источника атаки
B	Request Headers	Все заголовки HTTP-запроса	Анализ Client/UA/Cookies
I	Request Body	Тело запроса: JSON/form-data	Выявление XSS/SQLi внутри данных
J	Response Headers	Заголовки ответа сервера	Анализ поведения приложения
D	Response Body (фрагмент)	Фрагмент HTML/JSON ответа	Определение реакции backend
E	Matched Rules	Список сработавших правил и их ID	Идентификация типа атаки
F	Matched Data	Фрагмент данных, вызвавший срабатывание	Понимание причины блокировки
H	Debug/Stats	Время обработки, фаза проверки	Оптимизация и расследования
Z	EOF Marker	Маркер конца записи	Корректный парсинг логов

Таким образом, лог-файл WAF будет содержать:

- IP, время;
- метод/URL/заголовки;
- тело запроса;
- сработавшие правила, их ID и описания;
- какая часть запроса вызвала срабатывание;
- фрагмент ответа;
- завершение записи, чтобы журнал можно было парсить.

Теперь проверим правильность синтаксиса в файле **/etc/caddy/Caddyfile**:
caddy validate --config /etc/caddy/Caddyfile

Применим автоформатирование для ***Caddyfile***:
sudo caddy fmt --overwrite /etc/caddy/Caddyfile

Создайте каталог и файл для ведения логов аудита, назначьте следующие права доступа:
- владелец: сервисный пользователь **caddy**;
- группа: группа **nogroup**:

Создаем каталог для ведения логов
sudo mkdir -p /var/log/caddy

Создаем пустой файл аудита:
sudo touch /var/log/caddy/waf_audit.log

Настраиваем права доступа:
sudo chown -R caddy:nogroup /var/log/caddy
sudo chmod 750 /var/log/caddy
sudo chmod 640 /var/log/caddy/waf_audit.log

Проверим, что теперь Caddy может писать в лог-файл:
if sudo -u caddy test -w /var/log/caddy/waf_audit.log; then echo OK; else echo FAIL; fi

1.3.4 Настраиваем Caddy как сервис

systemd –это система, которая управляет службами (демонами) в Linux.

Если «на пальцах», то **systemd** это «диспетчер автозапуска программ» в Linux.

Функционал systemd при запуске/работе операционной системы:

- systemd запускает сервисы (сеть, ssh, nginx, caddy и т.д.);
- перезапускает их при сбоях;
- контролирует логи сервисов;
- позволяет удобно запускать/останавливать/перезапускать сервисы вручную

Настройка сервисов (демонов) осуществляется с помощью **unit-файлов**.

Unit-файл – это конфигурационный файл, который говорит systemd:

- ***какую*** программу запускать;
- от какого пользователя;
- куда писать логи;

- когда перезапускать;

В зависимости от типа службы unit-файлы располагаются обычно по следующим путям:

для **системных** сервисов	для **пользовательских** сервисов
/etc/systemd/system/	***~/.config/systemd/user/***

Caddy является системным сервисом, поэтому мы будем размещать unit-файлы в **/etc/systemd/system/**

Создайте unit-файл ***caddy.service***. Для этого откройте файл ***/etc/systemd/system/caddy.service*** в текстовом редакторе:

sudo nano /etc/systemd/system/caddy.service

и вставьте текст конфигурации:

```
[Unit]
Description=Caddy with Coraza WAF (reverse proxy for Juice Shop)
Documentation=https://caddyserver.com/docs/
After=network-online.target
Wants=network-online.target

[Service]
# Запуск а от служебного пользователя с минимальными правами
User=caddy
Group=nogroup

WorkingDirectory=/var/lib/caddy

# Основная команда: запуск с указанным конфигом
ExecStart=/usr/local/bin/caddy run --environ --config /etc/caddy/Caddyfile

# Мягкая перезагрузка конфига без простоя
ExecReload=/usr/local/bin/caddy reload --config /etc/caddy/Caddyfile

# Перезапускать при сбоях
Restart=on-failure
RestartSec=2s

# Увеличим лимит открытых файлов (для обслуживания большого количества соединений)
LimitNOFILE=1048576

# Определяем уровень защиты (hardening). Можно ослаблять при необходимости.
NoNewPrivileges=true
ProtectSystem=full
ProtectHome=true
PrivateTmp=true
LockPersonality=true
ProtectKernelTunables=true
ProtectKernelModules=true
ProtectControlGroups=true
RestrictSUIDSGID=true
RestrictRealtime=true
MemoryDenyWriteExecute=true

# Если будете слушать 80/443, нужно раскомментировать:
# AmbientCapabilities=CAP_NET_BIND_SERVICE
# CapabilityBoundingSet=CAP_NET_BIND_SERVICE
```

```
# Если используете переменные окружения (например, прокси), можно подключить файл:
# EnvironmentFile=-/etc/default/caddy

# Журналы пишем в journald, WAF-аудит — в /var/log/caddy/waf_audit.log

[Install]
WantedBy=multi-user.target
```

ПРИМЕЧАНИЕ по уровням информационной безопасности (**hardening**): директивы Protect… и пр. делает сервис более безопасным. Если из-за них что-то перестаёт работать (например, чтение конкретной директории), временно ослабьте соответствующую опцию и разберитесь, чего не хватает.

Регистрируем и запускаем сервис:
перечитать новые unit-файлы
sudo systemctl daemon-reload

включить автозапуск при загрузке
sudo systemctl enable caddy

стартовать сейчас
sudo systemctl start caddy

посмотреть статус
sudo systemctl status caddy --no-pager

В статусе сервиса Caddy ожидаем увидеть active (running):

```
user@waf:~$ sudo systemctl status caddy --no-pager
● caddy.service - Caddy with Coraza WAF (reverse proxy for Juice Shop)
     Loaded: loaded (/etc/systemd/system/caddy.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-10-27 15:50:35 MSK; 58s ago
       Docs: https://caddyserver.com/docs/
   Main PID: 2957 (caddy)
      Tasks: 7 (limit: 9432)
     Memory: 51.0M (peak: 51.3M)
        CPU: 250ms
     CGroup: /system.slice/caddy.service
             └─2957 /usr/local/bin/caddy run --environ --config /etc/caddy/Caddyfile
```

1.4. Склеротичка – схема компонентов Caddy + Coraza + OWASP CRS

Компонент	Путь	Описание
бинарник caddy (с модулем coraza)	/usr/local/bin/caddy	Reverse proxy сервер с модулем WAF Coraza
рабочий каталог caddy	/var/lib/caddy/	Рабочий каталог сервиса
каталог логов caddy / coraza	/var/log/caddy/	Хранение логов WAF.
файл аудит-логов waf	**/var/log/caddy/waf_audit.log**	Записи о блокировках и срабатываниях правил CRS.
конфигурация caddy	**/etc/caddy/Caddyfile**	Определяет Reverse proxy, включает coraza и подключает CRS.
конфигурация coraza	**/etc/caddy/coraza.conf**	Базовые настройки работы движка WAF.
конфигурация CRS	**/etc/caddy/coreruleset/crs-setup.conf**	Это не сами правила WAF. Это файл настроек, который: - определяет глобальные параметры поведения CRS; - задаёт значения по умолчанию; - включает режимы (например, на сколько «строгим» будет WAF); - **не** содержит самих сигнатур атак.
каталог правил CRS	**/etc/caddy/coreruleset/rules/*.conf**	Сами правила обнаружения атак: xss, sqli, rce, lfi и др.
юнит-файл systemd	/etc/systemd/system/caddy.service	Описывает, как Caddy запускается и управляется systemd.

Порядок загрузки компонентов:

1. systemctl запускает **/usr/local/bin/caddy**.
2. Caddy читает **/etc/caddy/Caddyfile.**
3. Caddy находит блок **coraza_waf** и загружает конфигурационные файлы (directives).
4. Загружается основная конфигурация Coraza WAF: **/etc/caddy/coraza.conf**.
5. Загружается конфигурация правил CRS: **/etc/caddy/coreruleset/crs-setup.conf**.
6. Загружаются сами сигнатуры атак: **/etc/caddy/coreruleset/rules/*.conf**

2. Задание

Общие требования: в ходе выполнения задания по всем пунктам необходимо вставлять в отчет скриншоты и давать краткие пояснения.

ПРИМЕЧАНИЕ: при выполнении задания все пакеты отправляем на наш WAF-прокси по адресу **http://192.168.56.20:3000**. При использовании сканеров Acunetix/nuclei, соответственно, изменяем цель (target) на http://192.168.56.20:3000.

2.0. Подготовка к работе.

Скачайте виртуальную машину Ubuntu с развернутым WAF Coraza по ссылке: https://disk.yandex.ru/d/C2FVk77wXNvU6A. Также вы можете самостоятельно установить все необходимые компоненты, руководствуясь разделом 1 «Установка WAF (Caddy + Coraza)».

2.1. Настройка стенда.

Завести все виртуальные машины в единую внутреннюю сеть VirtualBox и настроить IP-адресацию в соответствии рисунком 1 «Архитектура стенда».

2.2. Проверка работоспособности стенда.

2.2.1. Убедитесь, что WAF отвечает (с Kali Linux):

curl -I http://192.168.56.20:3000

Ожидаемый результат: заголовки ответа, среди них должен быть «HTTP/1.1 200 OK». Это показывает, что WAF слушает и возвращает ответ (проксирует на Juice Shop).

2.2.2. Скачайте главную страницу Juice Shop, отправив запрос с Kali Linux на WAF:

```
curl -sS http://192.168.56.20:3000/ \
  | awk 'BEGIN{done=0} { if(!done && tolower($0) ~ /<head>/) { sub(/<head>/, "<head><base href=\"http://192.168.56.20:3000/\">"); done=1 } print }' \
  > ~/juice_index.html
```

и откройте её с помощью браузера.

Если попробовать скачать главную страницу так:

```
curl -sS http://192.168.56.20:3000/ -o ~/juice_index.html
```

то она будет нечитабельной для браузера.

2.2.3. Протестируйте работоспособность WAF Coraza.

Выполните на Kali Linux ниже следующие запросы с помощью утилиты curl. После выполнения каждого теста нужно отрыть лог-файл WAF'a: ***/var/log/caddy/waf_audit.log*** зафиксировать и пояснить в отчете наблюдаемые результаты.

Открывать JSON-файл ***waf_audit.log*** удобно с помощь утилиты **jq**, универсального инструмента для чтения, форматирования и выборки из JSON в терминале:

sudo tail -n 50 /var/log/caddy/waf_audit.log | jq .

После того, как вы отразили в отчете результаты текущего теста, удобно сделать резервную копию лог-файла waf_audit.log, например, так:

sudo cp /var/log/caddy/waf_audit.log /var/log/caddy/waf_audit.log-$(date +"%d.%m_%H:%M:%S")-testX

В результате получим структуру файлов, похожую на приведенную ниже:

```
root@waf:/var/log/caddy# ls -al
итого 16
drwxr-x---  2 caddy nogroup 4096 окт 28 12:40 .
drwxrwxr-x 17 root  syslog  4096 окт 28 12:26 ..
-rw-r-----  1 root  root     785 окт 28 12:37 waf_audit.log-28.10_12:37:47-test1
-rw-r-----  1 root  root     785 окт 28 12:39 waf_audit.log-28.10_12:39:24-test2
```

Собственно тесты для запуска с Kali Linux:

2.2.3.1 Тест на блокирование XSS

curl -I "http://192.168.56.20:3000/?q=<script>alert(1)</script>"

2.2.3.2 Тест на блокирование SQL-инъекции

Мы не можем непосредственно передать в curl следующую строку **http://192.168.56.20:3000/rest/products/search?q=' OR 1=1--**, поэтому вынуждены использовать дополнительные опции для curl:

-G говорит curl, что параметры должны быть добавлены в строку запроса;

-I запрашивает только заголовки ответа, без тела;

--data-urlencode добавляет параметр q в URL с URL-кодированием;

"q=' OR 1=1--" - это инъекционная строка, которая пытается изменить SQL-запрос на сервере: OR 1=1-- всегда делает условие истинным, а -- комментирует остаток SQL.

Результирующий запрос получится таким:

```
echo "=== TECT SQLi ==="
curl -G -I \
  --data-urlencode "q=' OR 1=1--" \
  http://192.168.56.20:3000/rest/products/search
```

2.2.3.2 Тест на блокирование нагрузки в теле запроса

```
curl -i -X POST -H "Content-Type: application/json" \
  -d "{\"email\":\"admin' OR 1=1 --\",\"password\":\"x\"}" \
  http://192.168.56.20:3000/rest/user/login
```

Для проверки лога WAF Coraza можно использовать следующие команды:

sudo tail -n 50 /var/log/caddy/waf_audit.log | jq .

или

sudo tail -f /var/log/caddy/waf_audit.log | jq . # позволяет следить за наполнением файла

Если нужно прекратить вывод, нажмите Ctrl + C.

Описание команды ***sudo tail -f /var/log/caddy/waf_audit.log | jq .***:
sudo запускает команду от имени администратора.
tail выводит последние строки файла.
-f означает «следить за файлом в режиме реального времени» (показывать новые строки по мере появления).
/var/log/caddy/waf_audit.log – это лог-файл с записями аудита WAF.
| (пайп) передаёт вывод первой команды на вход второй.
jq – это утилита для форматирования и обработки JSON.
Точка (.) указывает jq выводить JSON в красивом формате.

2.3. Сканирование Juice Shop с включенным WAF Coraza.

Запустите полный скан на Acunetix (нужно будет изменить target на наш WAF: http://192.168.56.20:3000). Сравните полученный отчет с отчетом, полученным по результатам 5-ой лабы: отразите в отчете, какие уязвимости исчезли, какие остались (false negatives), какие появились (false positives).

2.4. Способы обхода WAF и противодействие им.

2.4.1. Обход WAF простым URL-энкодированием.

URL encoding – это способ представления специальных символов в адресах URL так, чтобы они корректно передавались по сети и правильно интерпретировались браузерами и серверами. Если в строке встречаются пробелы, русские буквы, знаки пунктуации и т.д., их нужно закодировать. Каждый такой «запрещённый» символ заменяется на знак процента (%) и его представление (код символа) в шестнадцатеричной системе.

Попробуйте XSS в нескольких вариантах:
обычный
curl -i "http://192.168.56.20:3000/?q=<script>alert(1)</script>"

URL-encoded
curl -i "http://192.168.56.20:3000/?q=%3Cscript%3Ealert(1)%3C%2Fscript%3E"

смешанные декодирования (double-encoded)
curl -i http://192.168.56.20:3000/?q=%253Cscript%253Ealert(1)%253C%252Fscript%253E

Отметьте, какие варианты блокируются, какие проходят. Отметьте в отчете, какие атаки прошли/не прошли и какие правила сработали/не сработали.

После этого можно обнулять содержимое лог-файла /var/log/caddy/waf_audit.log и приступать к выполнению очередного теста.

2.4.1. Обход WAF через модификацию заголовков и тела запроса.

Попробуйте поместить вредоносную нагрузку в тело запроса POST (JSON), в заголовок X-Custom-Header, или в cookie.

Проверьте, анализируется ли тело запроса в WAF (SecRequestBodyAccess On).

Результаты теста отразите в отчете.