

MANEJO DEFESA & MONITORAMENTO COM WAF MODSECURITY E DVWA

Evora da Ibéria Leite

Evora da Ibéria Leite

Setembro | 2025

Versão 1.0

Sumário

<i>Sumário Executivo</i>	3
<i>Objetivo</i>	4
<i>Escopo</i>	4
<i>Metodologia</i>	4
<i>Diagrama</i>	5
<i>Evidências</i>	7
Reconhecimento.....	7
Ataques DVWA – MODSEC_RULE_ENGINE=DetectionOnly	7
SQLi	8
XSS DOM	10
Command Injection.....	12
Ataques DVWA – MODSEC_RULE_ENGINE=On	14
SQLi	14
XSS DOM	15
Command Injection.....	16
<i>NIST IR</i>	17
Detecção.....	17
Contenção	17
Erradicação	17
Recuperação	17
Lições Aprendidas	17
<i>Recomendações</i>	18
Hardening da aplicação.....	18
Fine tuning das regras do WAF	18
Monitoramento contínuo.....	18
Treinamento	18
<i>Plano 80/20</i>	19
<i>Conclusão</i>	20

Sumário Executivo

O ambiente usado é um laboratório virtual. Nele, temos como foco principal duas máquinas virtuais, uma que atua como servidor da aplicação **DVWA** em modo *low security* e uma máquina contendo o *firewall ModSecurity* que está protegendo a aplicação.

A aplicação **DVWA** é basicamente um *website* com níveis de vulnerabilidade a ataques, para este projeto foi usado apenas o nível de vulnerabilidade *low security*. Cada ataque gera um log pelo *firewall ModSecurity*, o qual foi lido em uma aplicação de leitura de logs chamado **Dozzle**.

O *firewall* pode ser configurado em dois modos, um apenas para **detecção** dos ataques sem atuar no bloqueio, e outro modo para **detecção e bloqueio**. Foram testados os dois modos.

No modo de detecção, todos os ataques à aplicação foram bem-sucedidos, com logs detalhando as falhas e confirmando a criticidade destas vulnerabilidades. No modo **detecção e bloqueio**, os mesmos ataques foram bloqueados, evidenciando o papel do *firewall* de aplicações em mitigar ameaças, principalmente quando o código da aplicação e o tratamento de informações de input são os pontos fracos.

Pode-se concluir que uma aplicação web com riscos críticos e sem controles de segurança pode se beneficiar de um *firewall* de aplicações como o **ModSecurity**. Percebe-se que mesmo com regras padrão, o *firewall* já oferece uma barreira significativa contra ataques. Ainda assim, é importante avaliar a aplicação e mitigar os riscos ligados à programação da aplicação, entregando assim uma estratégia de defesa em profundidade.

Objetivo

Analizar os logs de um WAF configurado em uma aplicação web vulnerável a ataques.

Escopo

Ambiente virtualizado com Kali Linux com ambiente Docker simulando quatro máquinas:

- Firewall de aplicações **WAF ModSecurity**;
- Aplicação web **DVWA** low security;
- Visualizador de logs **Dozzle**;
- Kali Linux.

Metodologia

As máquinas estão em um ambiente Docker. Após a inicialização do ambiente, foi feita uma varredura utilizando a ferramenta Nmap utilizando a máquina **Kali Linux**:

```
nmap -sS -sV waf_modsec
```

A configuração do DVWA, onde, após o login, foi configurado o banco de dados (**create/reset database**) e o nível de segurança (**low**).

Os testes com o firewall WAF-ModSecurity foram feitos em dois modos:

- - MODSEC_RULE_ENGINE = **DetectionOnly**
- - MODSEC_RULE_ENGINE = **On**

Estes parâmetros foram mudados no arquivo docker-compose.yml do ambiente. Para cada modo, foram realizados ataques do tipo SQLi, XSS DOM e Command Injection, após os ataques os logs gerados foram acessados na aplicação Dozzle, disponível em localhost:9999.

Os ataques foram realizados diretamente na aplicação web, acessível em localhost:8080.

Diagrama

Os IPs:

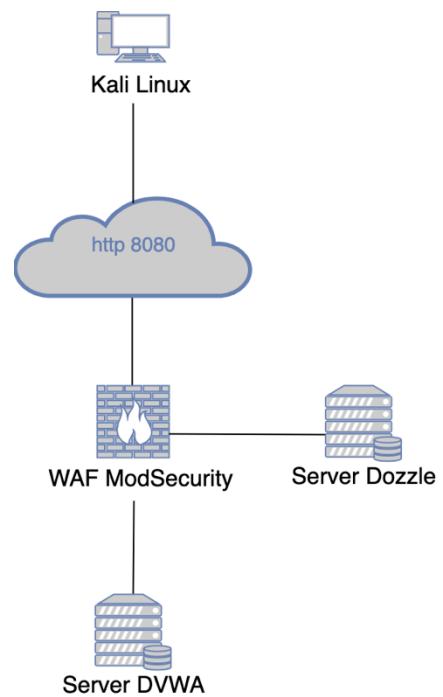
WAF ModSecurity - 192.168.35.30

DVWA - 192.168.35.40

Dozzle - 192.168.35.50

Kali Linux - 192.168.35.11

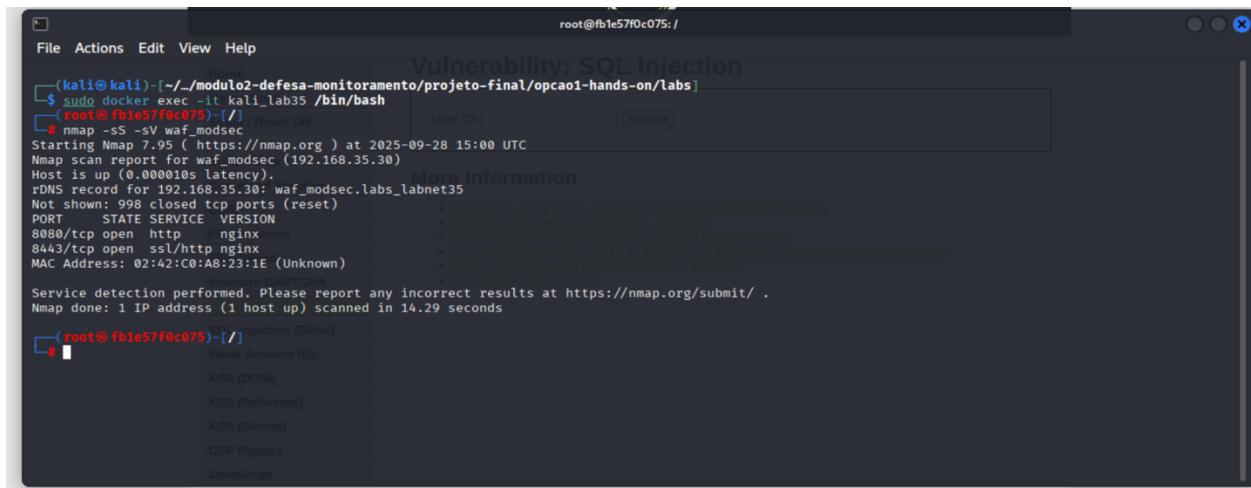
A máquina “atacante”, Kali Linux, tem acesso ao servidor DVWA em localhost:8080. O WAF está entre o DVWA e a máquina Kali pois ela tem os ataques bloqueados ao injetar os ataques via CLI (no caso do modo **On** do WAF). O Dozzle está conectado ao WAF para receber os logs.



Evidências

Reconhecimento

Primeiro foi feito o reconhecimento do ambiente utilizando a ferramenta Nmap, as portas 8080 e 8443 estão abertas, como mostra a imagem abaixo:



```
(kali㉿kali)-[~/../modulo2-defesa-monitoramento/projeto-final/opcao1-hands-on/labs]
$ sudo docker exec -it kali_lab35 /bin/bash
root@fb1e57f0c075:/#
# nmap -sS -sv waf_modsec
Starting Nmap 7.95 ( https://nmap.org ) at 2025-09-28 15:00 UTC
Nmap scan report for waf_modsec (192.168.35.30)
Host is up (0.000010s latency).
rDNS record for 192.168.35.30: waf_modsec.labs_labnet35
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
8080/tcp  open  http   nginx
8443/tcp  open  ssl/http nginx
MAC Address: 02:42:C0:AB:23:1E (Unknown)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.29 seconds
```

Ataques DVWA – MODSEC_RULE_ENGINE=DetectionOnly

Os ataques na aplicação DVWA foram feitos no modo *low security*, com o modo do WAF-MosSecurity – MODSEC_RULE_ENGINE = DetectionOnly, ou seja, teremos apenas os logs de detecção, os ataques não serão bloqueados.

SQLi

Este tipo de ataque ocorre através da injeção de queries SQL em um input de uma aplicação web. Quando este ataque é bem sucedido, é possível ler e modificar dados de um banco de dados, além também de executar operações a nível administrativo, também no banco de dados.

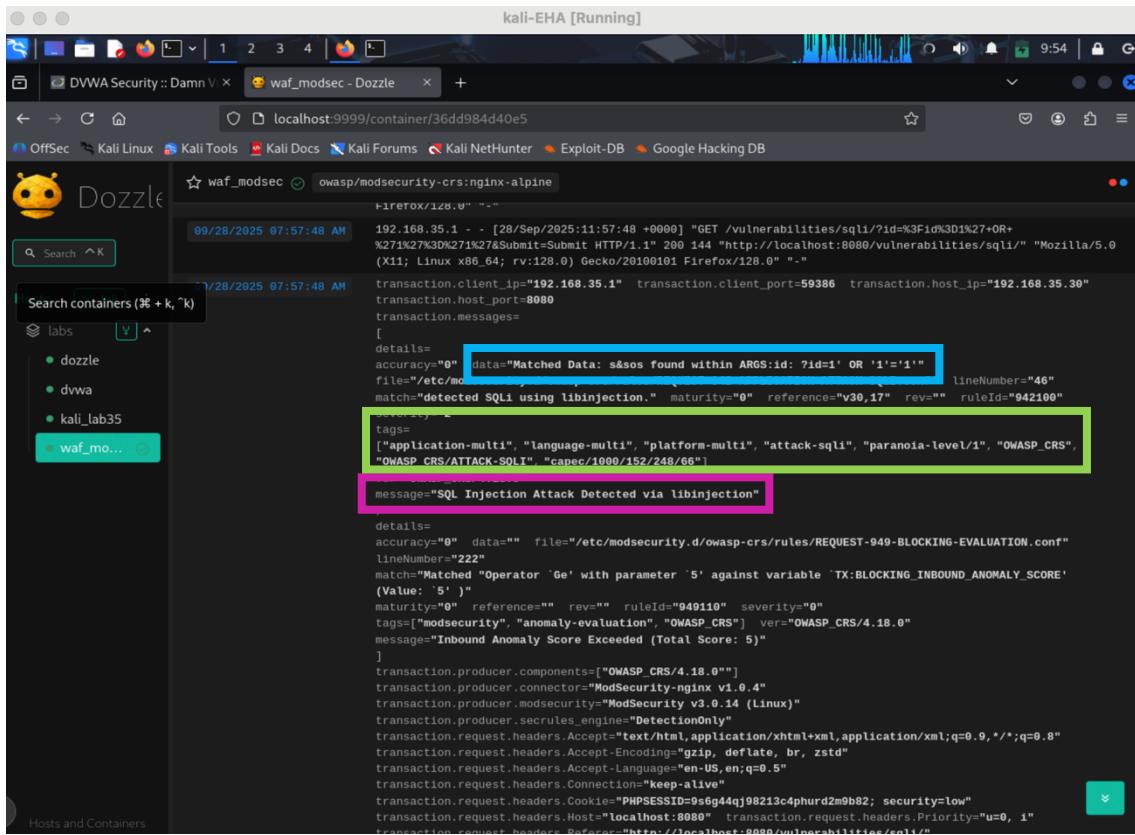
No caso da aplicação DVWA – low security, este tipo de ataque é possível. No campo de input do **User ID:**, foi feita a seguinte tentativa bem sucedida:

```
1' OR '1'='1'
```

The screenshot shows the DVWA application's SQL Injection vulnerability page. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection (the current tab), SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, DVWA Security, PHP Info, About, and Logout. The main content area has a title "Vulnerability: SQL Injection". It contains a form with "User ID:" and a "Submit" button. Below the form, several database rows are displayed, each showing a user's first name and surname. All rows have "First name" set to "admin" and "Surname" set to "admin", except for one row where "First name" is "Gordon" and "Surname" is "Brown". Further down, there are more rows with "First name" as "Hack" and "Surname" as "Me". The bottom of the page includes links for "More Information" and navigation buttons for "View Source" and "View Help". At the very bottom, it shows the user information: "Username: admin", "Security Level: low", and "PHPIDS: disabled".

Na imagem podemos ver dados sensíveis de outros usuários. **Podemos concluir que a aplicação é bastante vulnerável e que a função usada não possui tratamento de input.**

Observamos **Dozzle** o log abaixo. Em azul podemos ver o comando usado no input para se ter acesso ao banco de dados.



The screenshot shows a browser window titled "kali-EHA [Running]" displaying the "waf_modsec - Dozzle" page. The URL is "localhost:9999/container/36dd984d40e5". The log entry is from "waf_modsec" and "owasp/modsecurity-crs:nginx-alpine". The timestamp is "09/28/2025 07:57:48 AM". The log details a SQL injection attack where the user submitted "id=1 OR 1=1" to the database. The transaction information includes client IP "192.168.35.1", client port "59386", host IP "192.168.35.30", and host port "8080". The log also shows the rule ID "942100" and the file "/etc/modsecurity.d/owasp-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf". The message field contains the redacted command "id=1 OR 1=1". The tags field lists various categories, including "application-multi", "language-multi", "platform-multi", "attack-sqli", "paranoia-level/1", "OWASP CRS", and "OWASP CRS/ATTACK-SQLI". The message field also contains the redacted command "id=1 OR 1=1".

Como informações importantes temos o IP do cliente, a porta usada e principalmente as tags que confirmam o tipo de ataque:

```
tags=[“application-multi”, “language-multi”, “platform-multi”,  
“attack-sqli”, “paranoia-level/1”, “OWASP CRS”, “OWASP CRS/ATTACK-SQLI”, “capec/1000/152/248/66”]
```

Além das tags, também temos a mensagem:

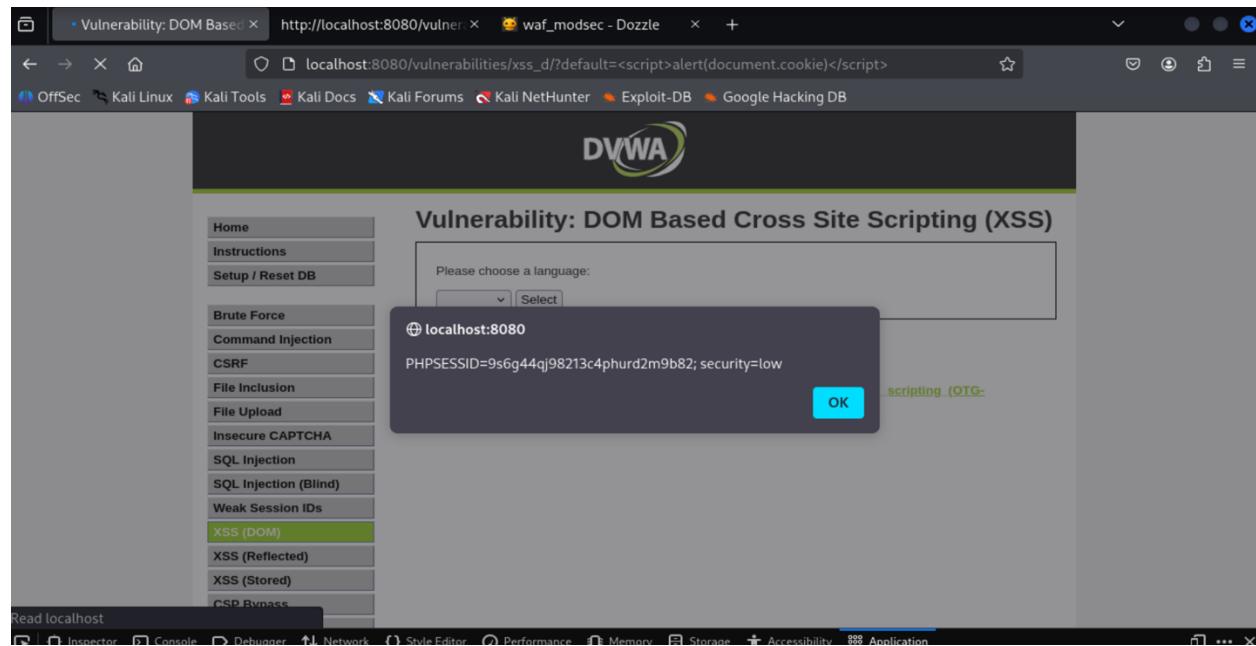
```
message=“SQL Injection Attack Detected via libinjection”
```

XSS DOM

Um ataque XSS DOM ocorre quando o atacante manipula o DOM do website no input da página, podendo assim ter acesso a dados sensíveis, como por exemplo cookies de sessão. O comando usado foi:

```
<script>alert(document.cookie)</script>
```

Na imagem abaixo podemos ver o resultado deste comando, quando colocado na URL.



Podemos concluir que esta vulnerabilidade é grave, pois podemos usar o cookie acessado para fazer um sequestro de sessão.

No **Dozzle**, confirmamos este tipo de ataque também pelas tags e mensagem:

```
tags=["modsecurity", "application-multi", "language-multi", "platform-multi", "attack-xss", "xss-perf-disable", "paranoia-level/1", "OWASP CRS", "OWASP CRS/ATTACK-XSS", "capec/1000/152/242"]  
  
message="NoScript XSS InjectionChecker: HTML Injection Injection"
```

Neste caso, para além das linhas **tag** e **message**, também podemos observar a linha **data**, que possui informações sobre o ataque. O WAF detectou a execução do script no request:

```
data="Matched Data: <script> found within REQUEST_HEADERS:Referer:  
https://localhost:8080/vulnerabilities/xss_d/?default=<script>aler(doc  
ument.cookie)</script>"
```

Abaixo, a imagem do log:

The screenshot shows a log entry from the Dazzle interface. The log details a detected XSS attack on a host named 'kali'. The log entry includes the following information:

```
transaction.client_ip="192.168.35.1" transaction.client_port=45176 transaction.host_ip="192.168.35.30"
transaction.host_port=8080
transaction.messages=
[
  {
    details:
      "data"="Matched Data: <script> found within REQUEST_HEADERS:Referer: http://localhost:8080/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>"
      "match"="Matched "Operator 'Rx' with parameter '(?i)<script[^>]*>[\s\S]*?' against variable REQUEST_HEADERS:Referer' (Value: 'http://localhost:8080/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>')"
      "maturity"="0"
      "reference"="o53,8v218,100t:utf8tounicode,t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDecode,t:removeNulls"
      "rev"="" ruleId="044110" severity="2"
    tags:
      ["modsecurity", "application-multi", "language-multi", "platform-multi", "attack-xss", "xss-perf-disable", "paranoia-level/1", "OWASP CRS", "OWASP CRS/ATTACK-XSS", "capec/1000/152/242"]
    message="XSS Filter - Category 1: Script Tag Vector"
  }
]
details=
accuracy="0"
data="Matched Data: <script found within REQUEST_HEADERS:Referer: http://localhost:8080/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>"
file="/etc/modsecurity.d/owasp-crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf" lineNumber="205"
match="Matched "Operator 'Rx' with parameter '(?i)<script[^>]*>[\s\S]*?' against variable REQUEST_HEADERS:Referer' (Value: 'http://localhost:8080/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>')"
maturity="0"
reference="o53,7v318,100t:utf8tounicode,t:urlDecodeUni,t:htmlEntityDecode,t:jsDecode,t:cssDecode,t:removeNulls"
rev="" ruleId="044110" severity="2"
tags:
  ["modsecurity", "application-multi", "language-multi", "platform-multi", "attack-xss", "xss-perf-disable", "paranoia-level/1", "OWASP CRS", "OWASP CRS/ATTACK-XSS", "capec/1000/152/242"]
ver="OWASP CRS/4.18.0"
message="NoScript XSS InjectionChecker: HTML Injection"
```

Command Injection

A imagem abaixo mostra um ataque bem sucedido de **Command Injection**. Este tipo de ataque é feito adicionando outro comando ao final do input, neste exemplo pude acessar um arquivo sensível do servidor (**/etc/passwd**) usando o seguinte input:

```
localhost ; cat /etc/passwd
```

Na imagem abaixo temos o arquivo na própria página web:

The screenshot shows a browser window for DVWA (Damn Vulnerable Web Application) at localhost:8080/vulnerabilities/exec/. The title bar says "localhost:8080/vulnerabilities/exec/#". The main content area is titled "Vulnerability: Command Injection" and contains a form titled "Ping a device" with a text input field containing "localhost ; cat /etc/passwd" and a "Submit" button. Below the form, the page displays the output of the command: a series of system logs and file contents. The logs include entries like "PING localhost (127.0.0.1): 56 data bytes", "64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.252 ms", and "64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.084 ms". The output also shows the contents of the "/etc/passwd" file, which includes entries for root, daemon, bin, sys, sync, games, man, mail, backup, and mysql users, along with their respective home directories and shell paths.

Nos logs, podemos confirmar o tipo de ataque:

```
tags=[“modsecurity”, “application-multi”, “language-shell”, “platform-unix”, “attack-rce”, “paranoia-level/1”, “OWASP CRS”, “OWASP CRS/ATTACK-RCE”, “capec/255/152/248/88”]
```

Além disso, também em vermelho em **data** o comando usando para fazer o ataque, em azul na figura da próxima página.

The screenshot shows a browser window titled "kali-EHA [Running]" displaying the Dozzele WAF interface. The URL is "localhost:59999/container/36dd984d40e5". The log entry is as follows:

```

star waf_modsec oowasp/modsecurity-crs:nginx-alpine
9/28/2025 07:57:08 AM 192.168.35.1 - [28/Sep/2025:11:57:08 +0000] "POST /vulnerabilities/exec/ HTTP/1.1" 200 1913 "http://localhost:8080/vulnerabilities/exec/" "Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0"
transaction.client_ip="192.168.35.1" transaction.client_port=59386 transaction.host_ip="192.168.35.30"
transaction.host_port=8080
transaction.messages=
[
  {
    details=
      accuracy="0" data="Matched Data: etc/passwd found within ARGS:ip: localhost ; cat /etc/passwd"
      file="/etc/modsecurity.d/owasp-crs/rules/REQUEST-932-APPLICATION-ATTACK-RCE.conf"
      match="Matched 'Operator $FromFile' with parameter 'lfi-os-files.data' against variable 'ARGS:ip'"
      (Value: 'localhost : cat /etc/passwd' )
      maturity="0" reference="o17_10v665_27t:uft8toUnicode,t:urlDecodeUni,t:normalizePathWin" rev=""
      ruleId="930120" severity="2"
      tags=
        ["modsecurity", "application-multi", "language-multi", "platform-multi", "attack-lfi",
        "paranoia-level/1", "OWASP CRS", "OWASP CRS/ATTACK-LFI", "capec/1000/255/153/126"]
      ver="OWASP CRS/4.18.0"
      message="OS File Access Attempt"
    ,
    details=
      accuracy="0" data="Matched Data: etc/passwd found within ARGS:ip: localhost cat /etc/passwd"
      file="/etc/modsecurity.d/owasp-crs/rules/REQUEST-932-APPLICATION-ATTACK-RCE.conf"
      match="Matched 'Operator $FromFile' with parameter 'unix-shell.data' against variable 'ARGS:ip'"
      (Value: 'localhost : cat /etc/passwd' )
      maturity="0" reference="o14_10v665_27t:cmdline,:normalizePath" rev="" ruleId="932160"
      tags=
        ["modsecurity", "application-multi", "language-shell", "platform-unix", "attack-rce", "paranoia-level/1",
        "OWASP CRS", "OWASP CRS/ATTACK-RCE", "capec/1000/152/248/88"]
      message="Remote Command Execution: Unix Shell Code Found"
    ,
    details=
      accuracy="0" data="" file="/etc/modsecurity.d/owasp-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"
      lineNum="222"
      match="Matched 'Operator $Ge' with parameter '5' against variable 'TX:BLOCKING_INBOUND_ANOMALY_SCORE'
      (Value: '10' )"
      maturity="0" reference="o10_10v665_27t:rule,:normalizePath" rev="" ruleId="949100" severity="0"
  }
]

```

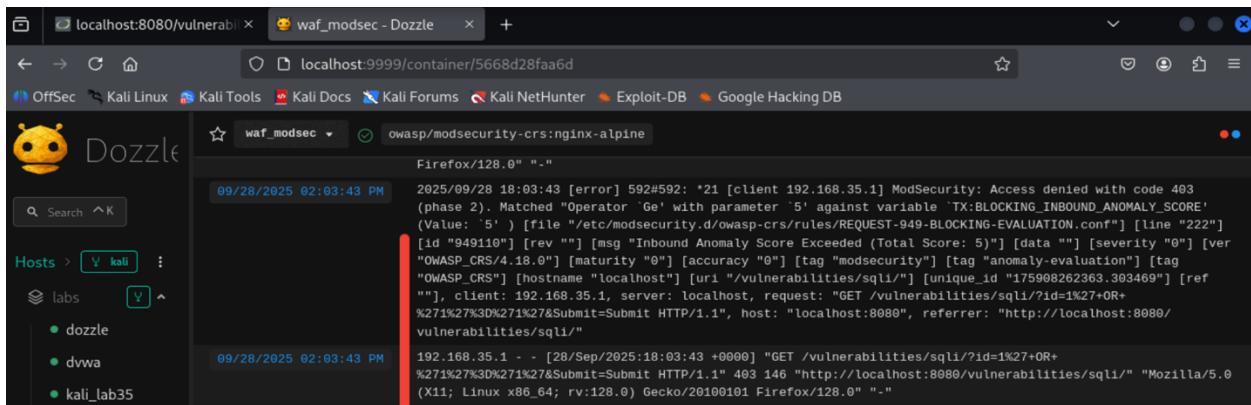
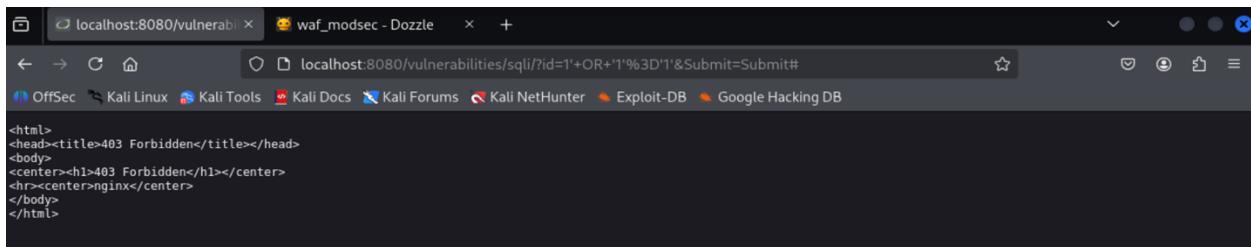
Ataques DVWA – MODSEC_RULE_ENGINE=On

Neste modo, o firewall bloqueia os ataques. Então percebemos que, mesmo sem um tratamento no *backend* da aplicação, os ataques podem ser bloqueados.

SQLi

No caso do SQLi, temos as mesmas informações de antes, porém agora temos a mensagem do WAF:

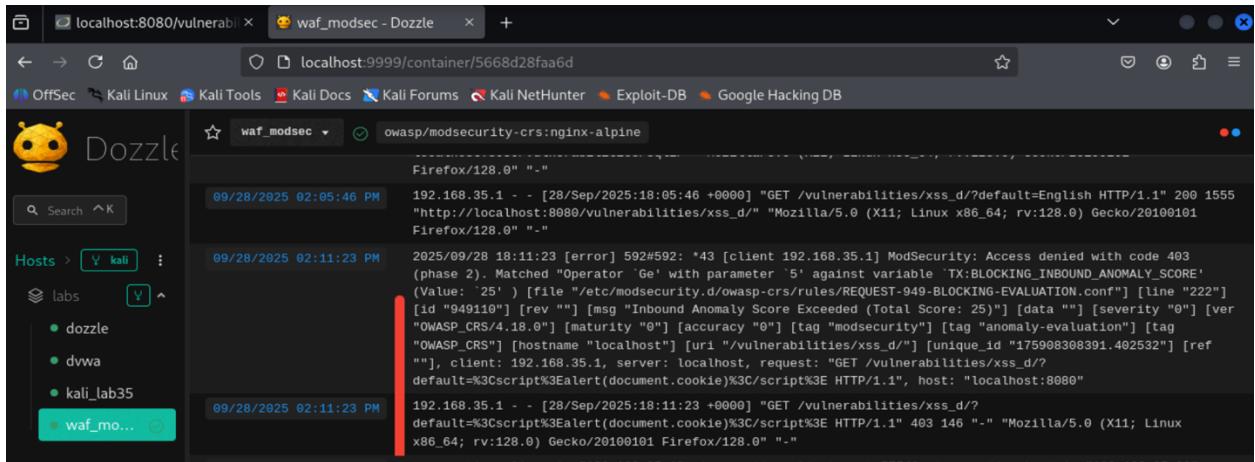
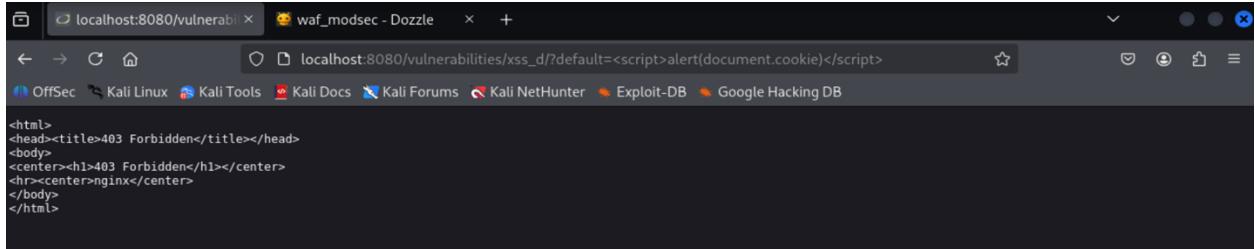
ModSecurity: Access denied with code 403



XSS DOM

No caso do XSS, temos as mesmas informações de antes, porém agora temos a mensagem do WAF:

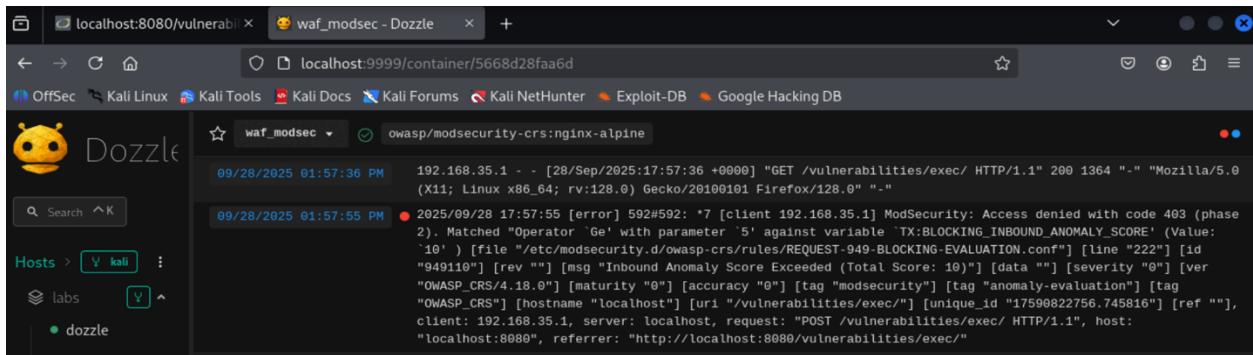
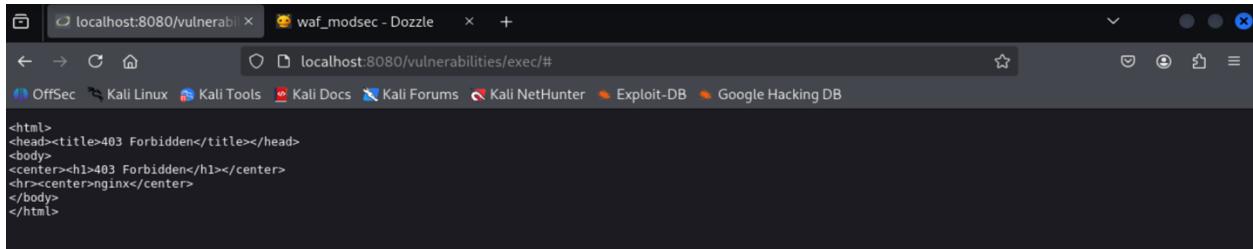
ModSecurity: Access denied with code 403



Command Injection

No caso do Command Injection, temos as mesmas informações de antes, porém agora temos a mensagem do WAF:

```
ModSecurity: Access denied with code 403
```



NIST IR

Detecção

Ataques web do tipo SQLi, XSS e Comman Injection foram detectados pelo WAF ModSecurity no modo **DetectionOnly**, no modo **On** foram detectados e bloqueados.

Contenção

Os ataques puderam ser contidos ajustando o WAF ModSecurity para o modo **On**. A contenção foi imediata, sem precisar alterar o código da aplicação.

Erradicação

Recomenda-se corrigir o backend da aplicação com sanitização de inputs, parametrização de queries e escaping. Atualizar continuamente o WAF.

Recuperação

Manter o WAF ativo, gerar logs e guardá-los para auditoria.

Lições Aprendidas

Sem controles de input a aplicação permanece vulnerável, mesmo com firewall ativo, pois o WAF deveria atuar como linha de defesa extra, não como mitigação para a não implementação de práticas seguras de desenvolvimento.

Recomendações

Hardening da aplicação

Aplicar sanitização e validação de inputs no *backend* e parametrização de queries, reduzindo dependência exclusiva do *firewall*.

Fine tuning das regras do WAF

Aumentar nível de paranoia do *firewall*, ajustar falsos positivos e monitorar continuamente os logs.

Monitoramento contínuo

Monitorar continuamente os logs gerados.

Treinamento

Reforçar práticas seguras de desenvolvimento e uso de ferramentas de *pentest* para validar controles.

Plano 80/20

AÇÃO	RISCO TRATADO	IMPACTO	FACILIDADE	PRIORIDADE
Ativar e manter o ModSecurity em modo On e atualizado	Ataques contra a aplicação	Alto	Alta	Alta
Implementar validação de input no backend	SQLi, Command Injection e XSS	Alto	Média	Média
Monitorar e integrar logs com alertas centralizados em um SIEM	Detecção tardia, falta de visibilidade de alertas importantes	Médio	Alta	Baixa

Conclusão

Os testes confirmam que a DVWA em *low security* é totalmente explorável por SQLi, XSS DOM e Command Injection. No modo **DetectionOnly**, o **ModSecurity** registra o ataque, mas não o bloqueia. Já no modo **On**, o WAF bloqueia os ataques, reduzindo de imediato o risco de exploração. É importante frisar que o WAF bloqueia os ataques porque detecta seus padrões, mas como o código da aplicação não segue os padrões de desenvolvimento seguro, a insegurança é inerente à aplicação.

Sem correção no *backend*, a vulnerabilidade continua existindo e pode ser explorada caso o WAF seja bypassado ou mal configurado. Portanto, recomendo validar os inputs no *backend*, parametrizar as queries e sanitizar os inputs para evitar escaping.