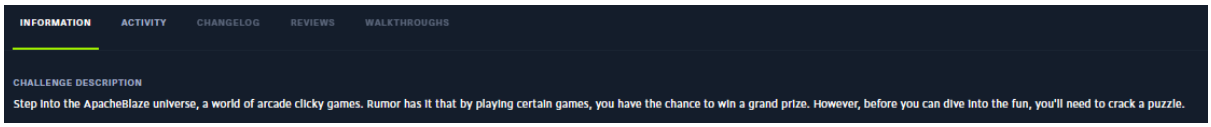
Erel Regev

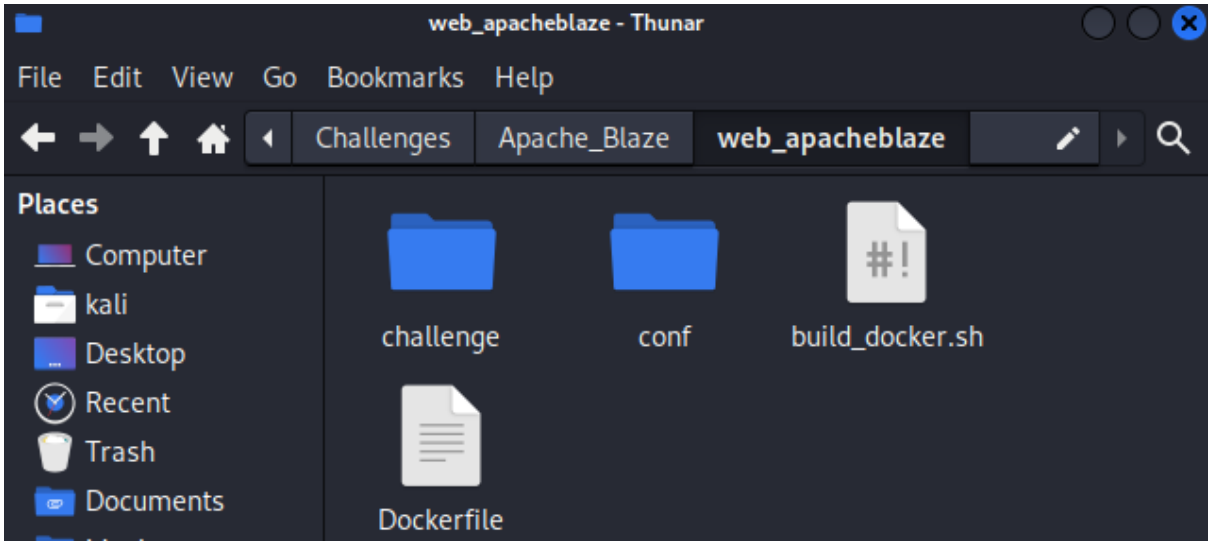
Table of Contents

Intro 1
Testing Functionality 2
conf/httpd.conf 3
backend/app.py 4
Apache & mod_proxy module 5
Exploiting 6

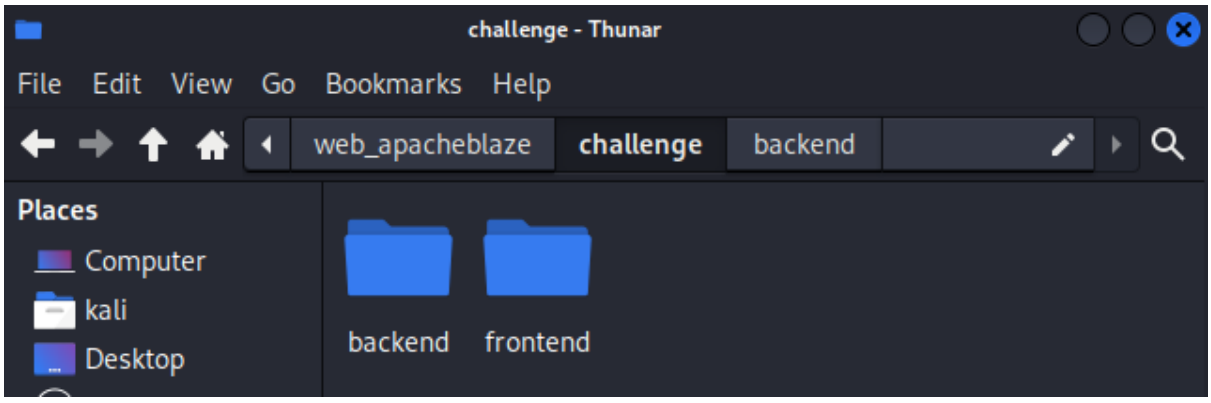
Intro



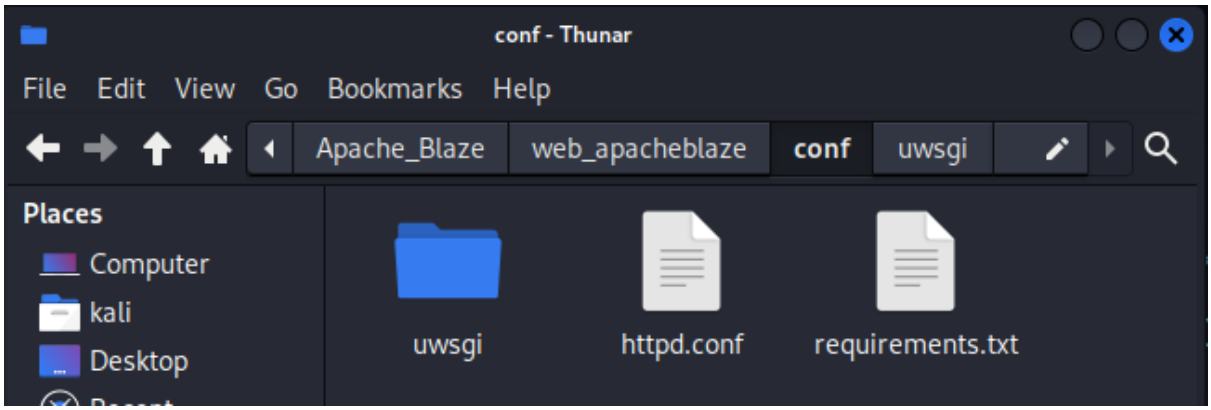
Received files:



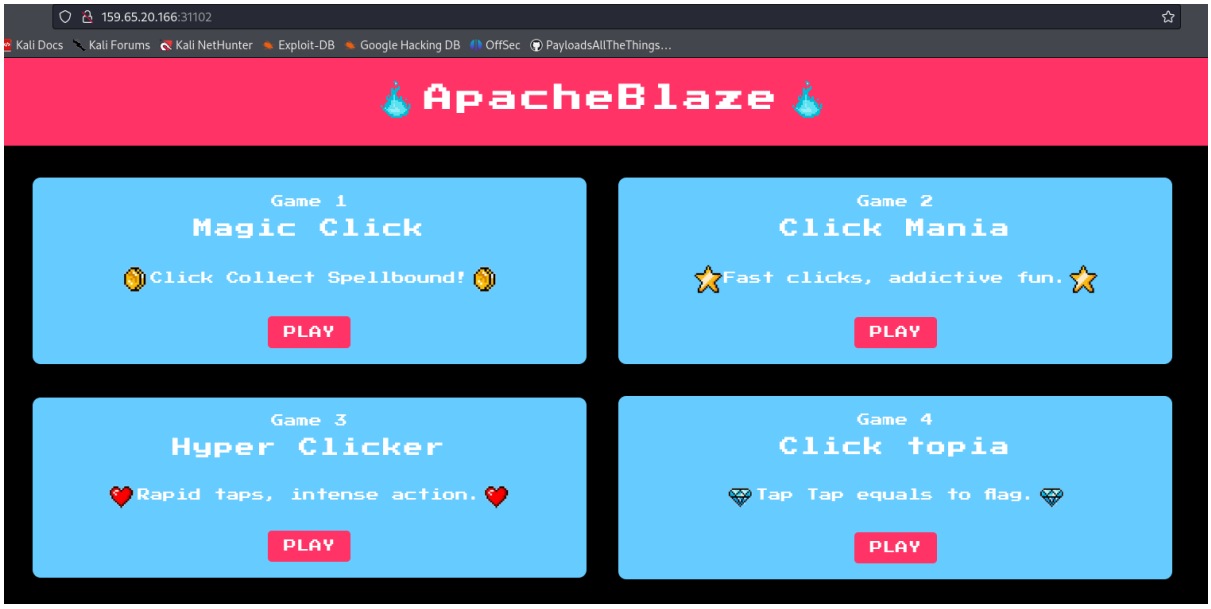
➔ Challenge directory



➔ conf directory

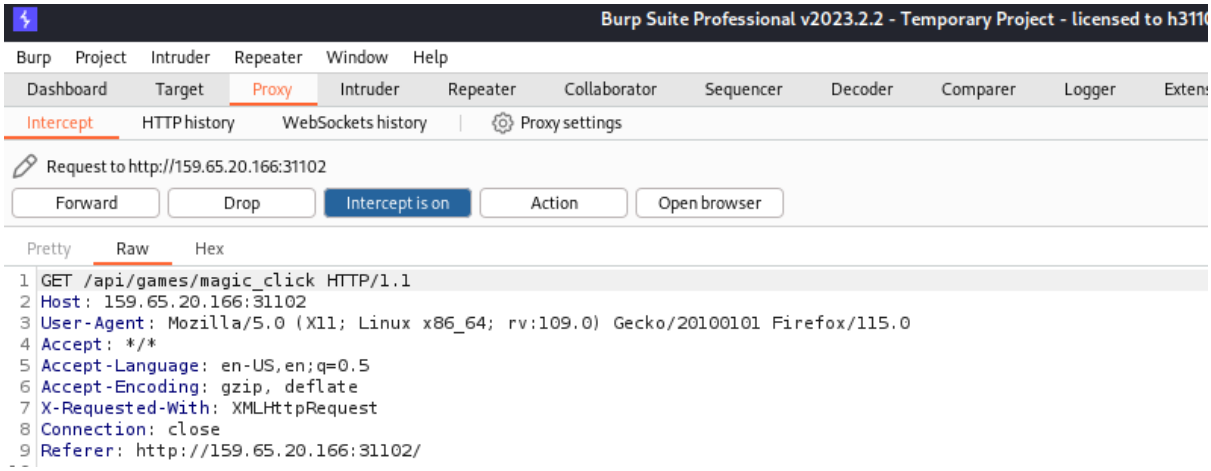


Accessing the website:



Testing Functionality

Capture request – Clicking on “Magic Click”:



Erel Regev

So this is a simple website that basically does nothing. While clicking on any of the available games on the website, nothing happens. Except:

Therefore, I will move on to the received files.

conf/httpd.conf

app.py × conf1.ini × conf2.ini × start_uwsgi.sh × httpd.conf ×

```
1   ServerName _
2   ServerTokens Prod
3   ServerSignature Off
4
5   Listen 8080
6   Listen 1337
7
8   ErrorLog "/usr/local/apache2/logs/error.log"
9   CustomLog "/usr/local/apache2/logs/access.log" common
10
11  LoadModule rewrite_module modules/mod_rewrite.so
12  LoadModule proxy_module modules/mod_proxy.so
13  LoadModule proxy_http_module modules/mod_proxy_http.so
14  LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
15  LoadModule slotmem_shm_module modules/mod_slotmem_shm.so
16  LoadModule lbmethod_byrequests_module modules/mod_lbmethod_byrequests.so
17
18  <VirtualHost *:1337>
19
20      ServerName _
21
22      DocumentRoot /usr/local/apache2/htdocs
23
24      RewriteEngine on
25
26      RewriteRule "^/api/games/(.*)" "http://127.0.0.1:8080/?game=$1" [P]
27      ProxyPassReverse "/" "http://127.0.0.1:8080:/api/games/"
28
29  </VirtualHost>
30
31  <VirtualHost *:8080>
32
33      ServerName _
34
35      ProxyPass / balancer://mycluster/
36      ProxyPassReverse / balancer://mycluster/
37
38      <Proxy balancer://mycluster>
39          BalancerMember http://127.0.0.1:8081 route=127.0.0.1
40          BalancerMember http://127.0.0.1:8082 route=127.0.0.1
```

This is an Apache HTTP Server configuration file (httpd.conf). It's specifying how the server should behave and handle requests.

While investigating the code there are two interesting things to note:

LoadModules

These lines load necessary modules for Apache, such as mod_rewrite, mod_proxy, mod_proxy_http, mod_proxy_balancer, etc.

VirtualHosts

Configuration for two virtual hosts: one on port 1337 and the other on port 8080.

This configuration seems to be setting up a reverse proxy and load balancing for requests, with specific rules for routing API requests. The load balancing is done between two servers, and stickysession is used to maintain session persistence.

HTB Challenge: ApacheBlaze - Difficulty: Easy - Web

Erel Regev

Stickysession is a feature in Apache's mod_proxy module that enables session persistence when load balancing. When you have multiple backend servers (BalancerMembers) in a load-balanced setup, stickysession ensures that a user's requests are consistently directed to the same backend server for the duration of their session.

** Load balancing is a technique used in computing and networking to distribute incoming network traffic or workload across multiple servers. The primary goal of load balancing is to optimize resource utilization, maximize throughput, minimize response time, and ensure that no single server becomes overwhelmed with too much demand.

```
38      <Proxy balancer://mycluster>
39          BalancerMember http://127.0.0.1:8081 route=127.0.0.1
40          BalancerMember http://127.0.0.1:8082 route=127.0.0.1
41          ProxySet stickysession=ROUTEID
42          ProxySet lbmethod=byrequests
43      </Proxy>
```

In our example, stickysession=ROUTEID means that the session will be sticky based on the ROUTEID parameter. Requests from the same client with the same ROUTEID will consistently be directed to the same backend server.

backend/app.py

app.py

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

app.config['GAMES'] = {'magic_click', 'click_mania', 'hyper_clicker', 'click_topia'}
app.config['FLAG'] = 'HTB{f4k3_fl4g_f0r_t3stIng}'

@app.route('/', methods=['GET'])
def index():
    game = request.args.get('game')

    if not game:
        return jsonify({
            'error': 'Empty game name is not supported!.'
        }), 400

    elif game not in app.config['GAMES']:
        return jsonify({
            'error': 'Invalid game name!'
        }), 400

    elif game == 'click_topia':
        if request.headers.get('X-Forwarded-Host') == 'dev.apacheblaze.local':
            return jsonify({
                'message': f'{app.config["FLAG"]}'
            }), 200
        else:
            return jsonify({
                'message': 'This game is currently available only from dev.apacheblaze.local.'
            }), 200

    else:
        return jsonify({
            'message': 'This game is currently unavailable due to internal maintenance.'
        }), 200
```

This is a simple application written in python.

Erel Regev

Conditions:

The code contains conditional statements to handle different scenarios based on the 'game' parameter.

- If 'game' is empty, it returns an error response.
- If 'game' is not in the supported games, it returns an error response.
- If 'game' is 'click_topia', it checks the 'X-Forwarded-Host' header. If it matches 'dev.apacheblaze.local', it returns the fake flag; otherwise, it indicates that the game is only available from a specific domain.

For other games, it returns a message indicating that the game is currently unavailable due to internal maintenance.

So obviously, we are going to focus on the following condition:

- If 'game' is 'click_topia', it checks the 'X-Forwarded-Host' header. If it matches 'dev.apacheblaze.local', it returns the fake flag; otherwise, it indicates that the game is only available from a specific domain.

Apache & mod_proxy module

We have to understand the following things in order to understand the point of this challenge:

Apache HTTP Server

Apache is one of the most widely used open-source web servers. It serves as the foundation for many web environments, delivering web content to users. Apache supports a modular architecture, allowing users to extend its functionality through modules.

mod_proxy Module

mod_proxy is one such module for Apache that provides proxying capabilities. Proxying involves forwarding client requests to other servers and returning the responses to the clients. The mod_proxy module, along with its related modules like mod_proxy_http, mod_proxy_balancer, and others, enables Apache to act as a reverse proxy or a load balancer.

Reverse Proxy

In a reverse proxy setup, Apache sits between the client and one or more backend servers.

When a client makes a request, Apache forwards the request to the appropriate backend server.

After receiving the response from the server, Apache sends it back to the client.

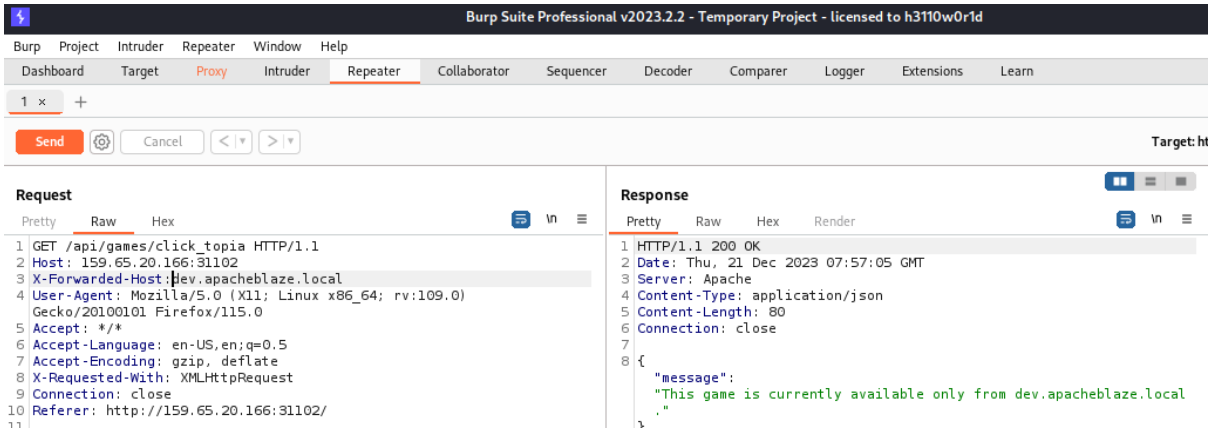
Load Balancer

Apache, with the help of mod_proxy_balancer, can distribute incoming requests among multiple backend servers to balance the load.

Erel Regev

Exploiting

I used burpsuite again to capture requests, this time for exploitation. I Clicked on ‘click_topia’ and sent rhe request to the repeater:



Seems not to be working.

What is the reason?

In the provided configuration, there are two virtual hosts—one for reverse proxy on port 1337 and another for load balancing on port 8080. Both of these virtual hosts rely on the X-Forwarded-Host header during request processing.

To exploit this setup, we need to manipulate the X-Forwarded-Host header to include only the desired value. Fortunately, we can leverage a known vulnerability: HTTP Request Smuggling Attack (CVE-2023–25690).

This attack allows us to craft a malicious request that, when processed by the server, results in a discrepancy between the interpretations of the request by the front-end and back-end servers. In this case, it could lead to the X-Forwarded-Host header being set to our desired value.

By exploiting this vulnerability, we can control the X-Forwarded-Host header and gain the ability to interact with the reverse proxy and load balancer in a way that suits our goals.

See the following:

https://github.com/dhmosfunk/CVE-2023-25690-POC/tree/main#internal-http-request-smuggling-via-header-injection

By employing the HTTP Request Smuggling Attack with the \r\n\r splitting method, we discover a technique to conceal a second request within the first one sent to the target. This allows us to initiate the second request directly from the reverse proxy without having additional elements appended to the end of the X-Forwarded-Host header.

I'll be dispatching two requests simultaneously, with the second request serving as a facilitator for the first one. The second request's purpose is to enable the initiation of the first request directly from the reverse proxy, allowing us to choose any desired route:

```
1  GET /api/games/click_topia HTTP/1.1
2  Host: dev.apacheblaze.local
3
4
5  GET / HTTP/1.1
6  Host: localhost:1337
```

Erel Regev

Use URL encoding:

The request should look like this when following the above rules: