Una mirada en profundidad

HTTP Request Smuggling

PhiloCyber 2024

Indice

1. Introducción
2. ¿Qué es HTTP Request Smuggling?
3. Tipos de Ataques
4. Automatización... ¿vale la pena?
5. Prevención
6. HTTP 2.0 - Se acabó el juego?

1. Introducción

Un poco de historia, un recorrido desde la HTTP/0.9 hasta HTTP/2.0

HTTP 0.9

- HTTP (HyperText Transfer Protocol), creado en 1991.
- Protocolo extremadamente simple.
- Una conexión TCP por solicitud.
- Solo un método (GET) y una única cabecera señalando la ruta del archivo.


```
# The full URL wasn't included as the protocol, server, and
# port weren't necessary once connected to the server.
GET /mypage.html

# The response was extremely simple too and only in HTML:
# it only consisted of the file itself, not even response
# headers (one TCP request per connection witho).
<html>
  A very simple HTML page
</html>
```

HTTP 1.0

- Creado en 1996.
- Fuerte transición desde la obtención de simples documentos hasta aplicaciones web más dinámicas.
- Más HTTP métodos, como: POST, HEAD, PUT and DELETE.
- Múltiples solicitudes HTTP bajo la misma conexión TCP.
- Agrega metadatos y más contexto en varios encabezados nuevos para facilitar tipos de interacciones más especializados.

Nuevos Headers en HTTP 1.0

Content-Length

Indicando el tamaño del cuerpo de la entidad en bytes.

Connection

En esta versión, podría incluir “keep-alive" para solicitar una conexión persistente.

Date

Indicando la fecha y hora en que se originó el mensaje.

Content-Type

Especificar el entity-type de la solicitud enviada al servidor (text/html, application/json, etc).

Content-Encoding

Especificar la codificación aplicada al cuerpo de la entidad, permitiendo la compresión o el cifrado.

Nuevos Headers en HTTP 1.0

Content-Language

Describir el(los) lenguaje(s) natural(es) de la audiencia prevista para la entity-body.

Host

Especifica el nombre de dominio del servidor.

Accept

Informa al servidor sobre los tipos de contenido que el cliente puede entender.

User-Agent

Proporciona información sobre el cliente que realiza la solicitud, como el tipo de navegador o agente de usuario.

Authorization

Se utiliza para enviar credenciales (como usuario y contraseña) para acceder a recursos protegidos.

HTTP 1.1

- Se lanza apenas un año después de la versión anterior, 1997.
- Más centrado en mejoras de eficiencia.
- Más métodos HTTP.
- Mantener viva la conexión por defecto (keep-alive).
- Nuevas características (Pipelining y Range Request).

Nuevos headers y cambios en HTTP 1.1

Transfer-Encoding
Específica la forma de codificación utilizada para transferir de forma segura el payload al usuario.

Host
*Cambia a **obligatorio** para distinguir entre diferentes hosts virtuales que comparten la misma dirección IP.*

Cache-Control
Permite un control sofisticado de la caché con directivas como “no-cache, no-store, max-age, etc".

ETag
Porporciona un identificador único para una versión específica de un recurso, lo que ayuda en la validación de la caché.

Ranged
Permite a los clientes solicitar porciones específicas de recursos, como se mencionó anteriormente.

Nuevos headers y cambios en HTTP 1.1

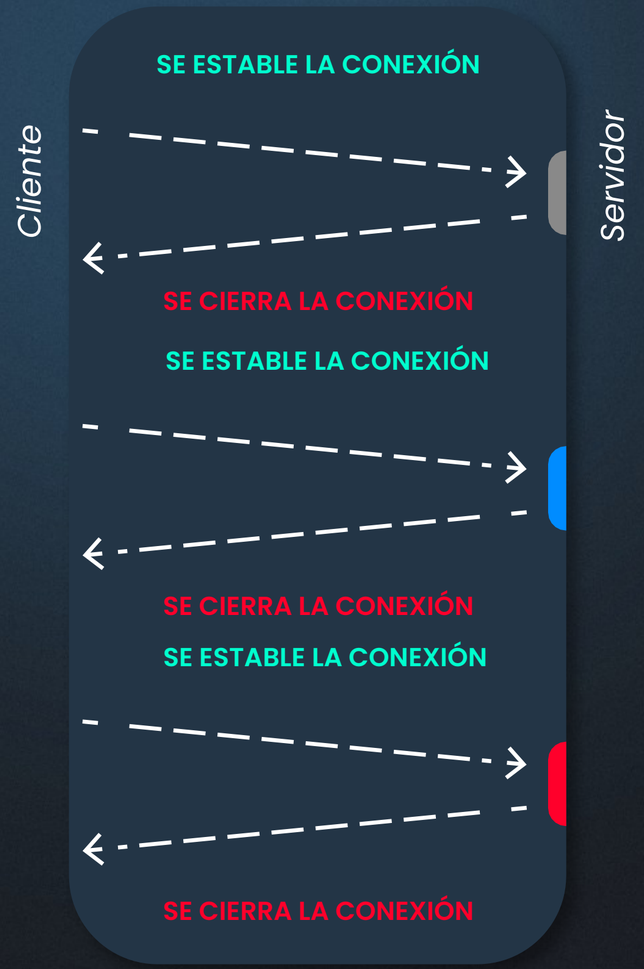
If-Range
Permitir solicitudes condicionales basadas en una ETag de recursos, lo que permite a los clientes recuperar el recurso solo si se ha modificado.

Content-Range
Enviado por el servidor en respuestas parciales para especificar el rango de bytes que se devuelven.

Age
Clientes informados sobre la hora en que un recurso se almacenó en caché en un proxy.

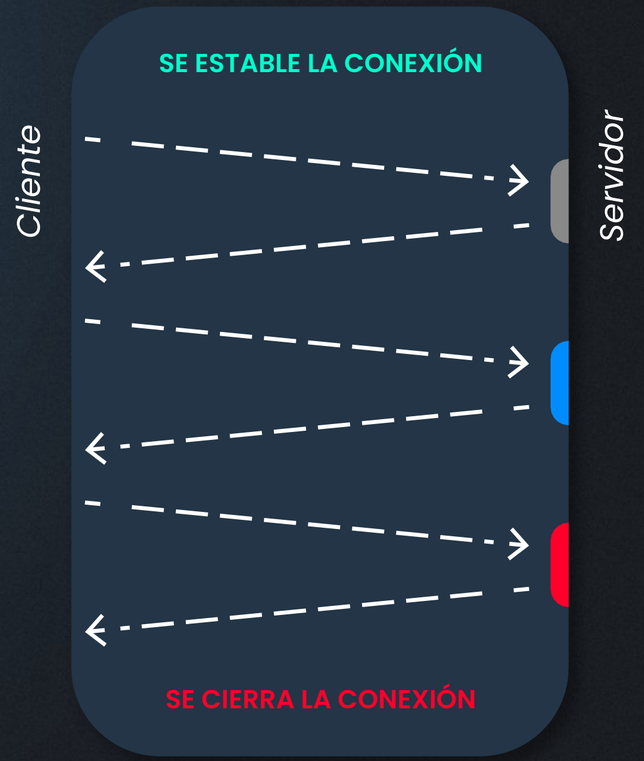
HTTP/0.9

Conexiones de corta duración



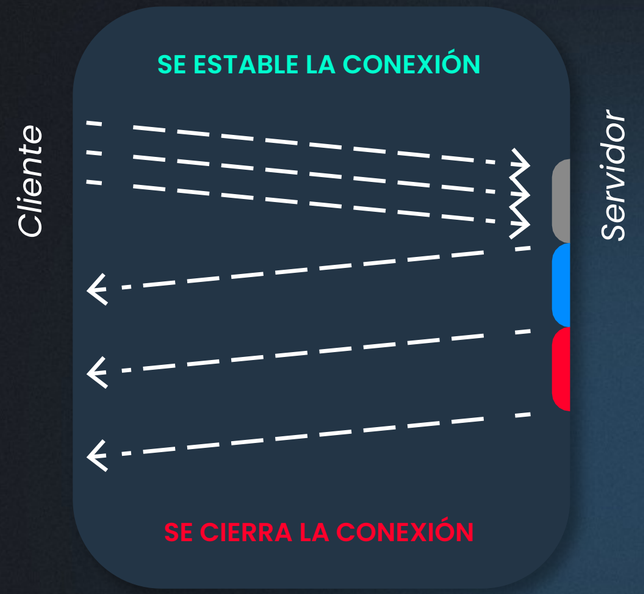
HTTP/1.0-1.1

Conexiones persistentes



HTTP/1.1

HTTP Pipelining



Pipelining

Podemos enviar diferentes solicitudes HTTP en la misma conexión sin tener que esperar la primera respuesta de la primera solicitud (para este ejemplo, rojo).

Además, podemos ver cómo podemos delimitar la longitud de nuestra carga útil utilizando dos encabezados diferentes, Longitud del contenido y codificación de transferencia.

```
POST / HTTP/1.1
Host: example.local
Content-Length: 5

HELLOGET /index HTTP/1.1
Host: example.local
Content-Length: 23

This is another exampleGET /admin HTTP/1.1
Host: example.local
Transfer-Encoding: chunked

5
HELLO
0
```

Transfer-Encoding sobre Content-Length

"Si se recibe una solicitud con un header Transfer-Encoding y un otro header Content-Length, este último debe ignorarse"

Para más información sobre esto visitá el siguiente link:

https://www.rfc-editor.org/rfc/rfc2616#section-4.4

HTTP/2.0… Ahora qué?

Introducido en 2015, tiene como objetivo reducir la latencia y mejorar el rendimiento del tráfico HTTP.

Multiplexing

Actualización importante, que permite múltiples solicitudes y respuestas simultáneas dentro de una única conexión TCP, lo que permite enviar y recibir recursos de forma asincrónica, mejorar la eficiencia de la transferencia de datos y reducir la latencia.

Protocolo Binario y Compresión de Headers

Se adoptó una capa de marco binario que reemplazó el formato de texto plano utilizado en HTTP/1.1 y anteriores versiones. Básicamente, mejora la eficiencia del parsing que hacen los diferentes componentes (proxy y server), lo que conduce a un procesamiento más rápido y un uso reducido del ancho de banda.

Server Push

Permite que los servidores envíen proactivamente recursos al caché del cliente antes de que sean solicitados. Esto optimizó la carga de páginas web al proporcionar de forma preventiva los recursos necesarios, reduciendo la necesidad de solicitudes posteriores.

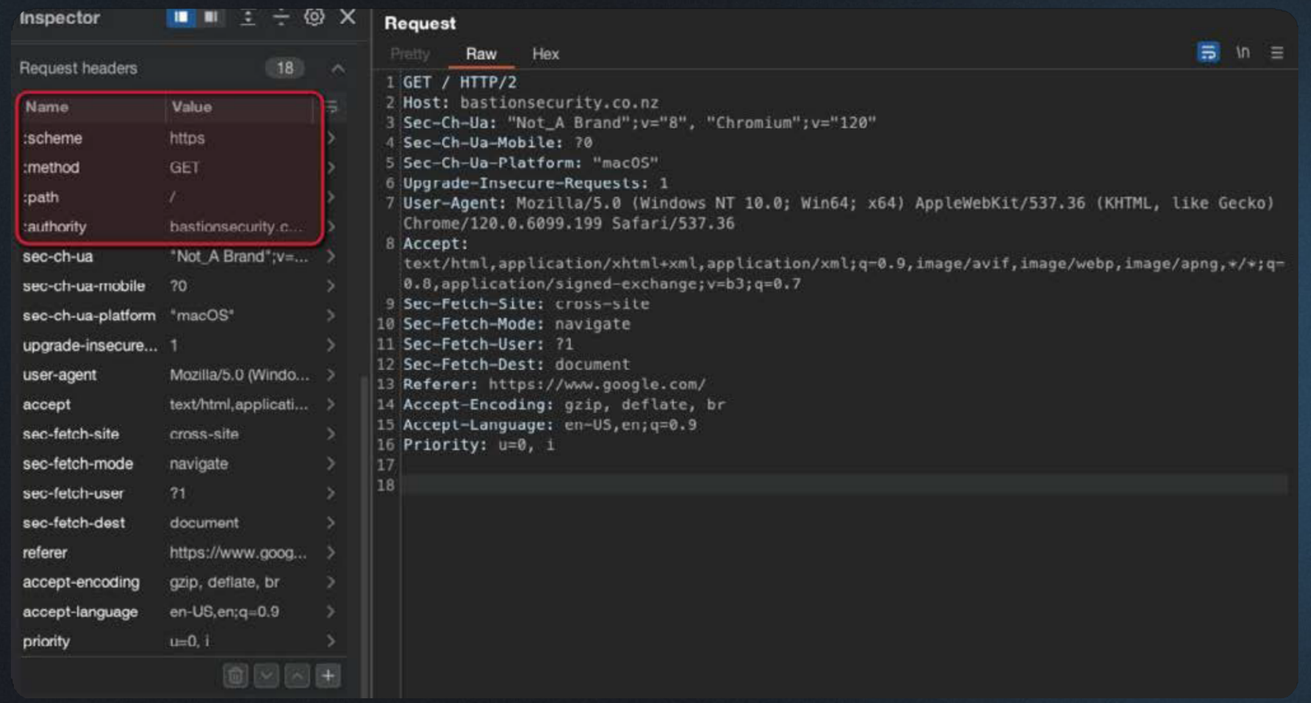
Priorización y Dependencia de flujos

Usar prioritisation of streams, permite priorizar los recursos críticos para una entrega más rápida, mejorar la experiencia general del usuario. La información de dependencia entre diferentes recursos ayudó a optimizar los datos.

Nueva versión, nuevas formas de manejar datos

Los siguientes pseudo headers se definen en una solicitud HTTP/2:

```
:method GET
:path /index.php
:authority bastionsecurity.co.nz
:scheme https
```



Compatibilidad total con las versiones anteriores (los métodos HTTP, los encabezados y las rutas de consulta aún existen, pero los datos tienen un formato diferente en tránsito).
Del texto claro al binario.
La codificación de transferencia ya no es compatible.

2. ¿Qué es HTTP Desync?

HTTP Request Smuggling

HTTP Request Smuggling (RS)

El ataque de desincronización, también conocido como HTTP Request Smuggling (RS), se considera un vector de ataque avanzado que explota la discrepancia entre los sistemas frontend y backend en el análisis de las solicitudes HTTP entrantes, forzando un desacuerdo en los límites de las solicitudes entre los dos sistemas, provocando así una **desincronización**.

¿Cómo funciona?

Paso 1

La solicitud del usuario enviada al servidor

Paso 2

Solicitudes procesadas por el servidor

Paso 3

Respuesta enviada nuevamente al usuario

Impacto de la Vulnerabilidad

Dependiendo del tipo específico de desacuerdo entre los sistemas, esta vulnerabilidad puede tener un impacto diferente, incluida la explotación masiva de XSS, el robo de datos de otros usuarios e incluso la posibilidad de bypassear algunas protecciones de un Web Application Firewall (WAF).
Para ver en más detalle los ataques de HTTP Request Smuggling, visita la siguiente publicación de blog de James Kettle.

- *https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn*

3. Tipos de Ataques y Técnicas

Ejemplos en vivo

CL.TE

- Es el tipo más común de ataque de HTTP Request Smuggling.
- El Reverse Proxy toma la longitud del contenido como referencia para el límite de la solicitud HTTP (no admite Transfer-Encoding).
- En cambio, el Web Server utiliza Transfer-Encoding para la delimitación.

```
POST / HTTP/1.1
Host: philocyber.com
Content-Length: 10
Transfer-Encoding: chunked

0
HELLO
```

Interpretación del Reverse Proxy

```
POST / HTTP/1.1
Host: philocyber.com
Content-Length: 10
Transfer-Encoding: chunked

0
HELLO
```

Interpretación del Web Server

HACKTHEBOX

Identificación y Explotación

DEMO CL-TE

TE.TE

- Pasado por alto por varias herramientas de automatización.
- Hay una diferencia en cómo los comentarios siguen el estándar RSA; pero ambos siguen admitiendo el encabezado Transfer-Encoding.
- Necesitamos forzar la mala interpretación manipulando el encabezado se la siguientes maneras:

Description	Header
Substring Match	Transfer-Encoding: **test chunked**
Space in Header name	Transfer-Encoding: **chunked**
Horizontal Tab Separator	Transfer-Encoding: [\x**09**]**chunked**
Vertical Tab Separator	Transfer-Encoding: [\x**0b**]**chunked**
Leading Space	Transfer-Encoding: **chunked**

Nota: Las secuencias [\x09] y [\x0b] no son secuencias de caracteres literales utilizadas en la ofuscación. Más bien indican el carácter de tabulación horizontal (ASCII 0x09) y el carácter de tabulación vertical (ASCII 0x0b).

Se trata de cómo los diferentes sistemas interpretan el encabezado TE, es posible que solo verifiquen la presencia de chunked, mientras que el otro podría estar verificando una coincidencia exacta.

HACKTHEBOX

Identificación y Explotación

DEMO TE-TE

TE.CL

- El más complejo de identificar y explotar manualmente.
- El Reverse Proxy toma la codificación de transferencia como referencia para el límite de la solicitud HTTP (no admite CL).
- En cambio, el servidor de aplicaciones web utiliza la longitud del contenido para la delimitación.

```
POST / HTTP/1.1
Host: philocyber.com
Content-Length: 3
Transfer-Encoding: chunked

5
HELLO
0
```

Interpretación del Reverse Proxy

```
POST / HTTP/1.1
Host: philocyber.com
Content-Length: 3
Transfer-Encoding: chunked

5
HELLO
0
```

Interpretación del Web Server

HACKTHEBOX

Identificación y Explotación

DEMO TE-CL

4. Automatización

Vale la pena?

Escenarios

- La automatización puede valer la pena para testing en scopes que son extenso, pero la disponibilidad de herramientas gratuitas de GitHub para este tipo de pruebas es bastante limitada en comparación a otras vulnerabilidades.

- En términos de herramientas/automatización la mejor actualmente, en mi opinión, es el ***HTTP Request Smuggler*** creado por James Kettler, pero aún así puede devolver falsos positivos.

- Creo que debemos intentar evitar la dependencia en herramientas automatizadas, más aún si sabemos que no hay herramientas maduras y flexibles como nmap, sqlmap, etc.

5. Prevenciones

Posibles mitigaciones

La mitigación de este tipo de ataques puede ser desafiante ya que las vulnerabilidades que lo causan generalmente residen dentro del software del servidor web, haciendo la prevención desde la aplicación web un desafío imposible.

Prevención

- Debemos asegurarnos de que el servidor web y el reverse proxy estén actualizados y con parches de seguridad instalados lo antes posible.
- Asegurar de que las vulnerabilidades client-side que puedan parecer "*inexplotables*" se encuentren parcheadas y monitoreadas ya que podrían volver explotables en un escenario de HTTP Desync.
- Configurar al servidor web para que por defecto cierre las conexiones TCP cuando se produzcan excepciones o errores en el nivel del servidor web durante el maneja y análisis de solicitudes.
- De ser posible, configurar el uso de HTTP/2 entre el cliente y el servidor. Asegurando que versiones previas de HTTP estén deshabilitadas.

6. HTTP/2.0, se acabó el juego?

Sigamos de cerca los nuevos cambios

Características de HTTP/2.0

- Ya no se soporta/admite la cabecera chunked.
- HTTP/2 transmite el cuerpo de la solicitud en formato binario que consta de Marcos de datos (data frames).
- No se requiere ningún campo de longitud explícito para determinar la longitud del cuerpo de la solicitud.
- Los marcos de datos contienen un campo de longitud incorporado que cualquier sistema puede utilizar para calcular la longitud del cuerpo de la solicitud.
- Buenas noticias... los ataques de desincronización son casi imposibles si HTTP/2 se implementa y configura correctamente!

Pero espera un segundo, y... ¿qué pasa con la HTTP/2 Downgrading?

Gracias por su tiempo!!

Ricardo Prieto

Penetration Tester en

BASTION

@PhilocyberWithRichie

https://philocyber.com

PhiloCyber 2024