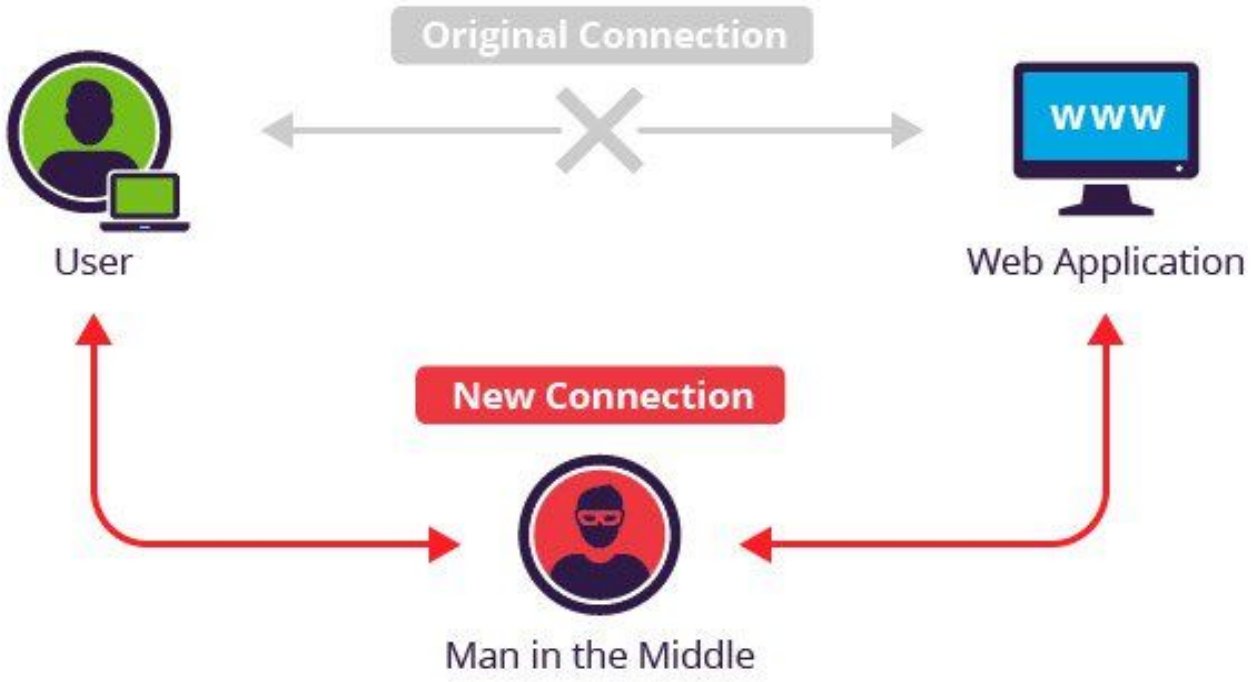
MAN IN THE THE MIDDLE



Óscar Osorio Giráldez
Antonio Martos Rodríguez

ugr Universidad de Granada

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Índice

Man in the middle (MITM)

1. Introducción

Cómo usuarios de la web nos vemos expuestos a todo tipo de amenazas, siempre que estemos conectados pueden haber ataques malintencionados que busquen acceder a nuestros datos aprovechándose de vulnerabilidades en las comunicaciones que establecemos día a día.

Hay muchos tipos de ataques a los que estamos expuestos y no somos conscientes de ello, por esa razón vamos a explicaros uno que es bastante común, el llamado “man in the middle” u “hombre en el medio”.

A lo largo de este trabajo explicaremos en qué consiste este ataque de forma básica pues sus diferentes metodologías y formas lo hacen ampliamente extenso.
Una vez “estando en medio” vamos a centrarnos en ataques a protocolos HTTP y HTTPS de forma directa e indirecta (inferencia). Tras contar de cuántas maneras podemos ser vulnerables nos centraremos en lo que nos atañe, cómo nos podemos proteger ante todo esto y para finalizar las conclusiones que sacamos tras el profundo análisis y comprensión sobre el tema.

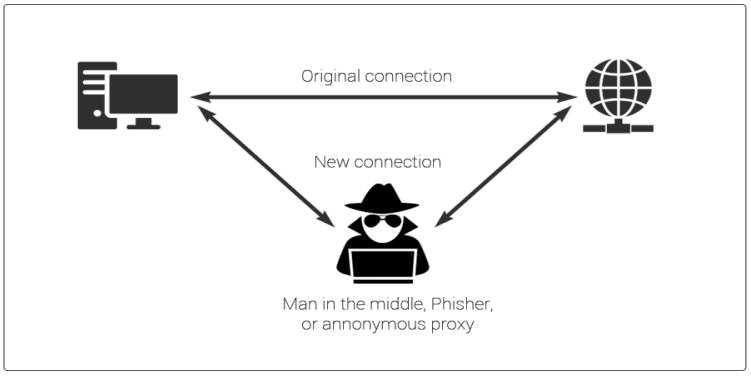
2. ¿Qué es el ataque Man-in-the-Middle?

Un ataque Man-in-the-Middle es un tipo de ataque cibernético en el que el atacante se inserta entre la comunicación entre dos partes (personas o sistemas) sin que ninguno de ellos se dé cuenta y retransmite la comunicación entre ellos. Como el atacante tiene acceso completo a la comunicación, puede **interceptar, espiar o alterar la información, y luego enviar y recibir comunicaciones de las dos partes.**

Como las dos partes creen que se están comunicando entre sí, el atacante podría obtener acceso a la información confidencial que se comparte e inyectar la información que desea.

Los ataques MITM también son posibles en el procesamiento en tiempo real, lo que permitiría a los atacantes comprometer las transacciones financieras, como modificar el número de cuenta del destinatario y la cantidad transferida.

Para este tipo de ataques se necesitan dos máquinas víctima, que bien podría ser el servidor y un equipo de una red empresarial, o bien el router y el equipo de nuestra víctima real, además de la máquina del atacante. Siempre que el atacante pueda autenticarse como los dos lados de la comunicación, tendrá todo el acceso.



3. ¿Cómo se realiza un ataque Man-in-the-Middle?

MITM en LAN

ARP spoofing

Para entender cómo se realiza una de las formas más sencillas de ataque MITM hay que conocer qué es el protocolo ARP.

Cuando se envía un paquete de un host a otro hay que indicar en su cabecera la dirección física (MAC), que es un identificador fijo y único asignado a cada tarjeta de red. Cuando una aplicación se quiere comunicar con otra a través de una red usará el protocolo IP para identificar la máquina de destino, pero teniendo en cuenta que las direcciones IP pueden variar se hace imprescindible asociarlas a las direcciones físicas (MAC). Para ello, se utiliza el protocolo ARP (***Address Resolution Protocol)***, de modo que **cuando un paquete llega a una máquina, esta comprueba que en la cabecera se indique su MAC y si no coincide con la suya, ignorará el paquete**.

Para realizar un ataque MITM hay que envenenar las tablas ARP. El envenenamiento de las tablas ARP o **ARPspoof consiste básicamente en inundar la red con paquetes ARP indicando que nuestra mac address es la asociada a la IP de nuestra víctima y que nuestra MAC está también asociada a la IP del *router*** (puerta de enlace) de nuestra red. De esta forma, **todas las máquinas actualizarán sus tablas con esta nueva información maliciosa**. Así, cada vez que alguien quiera enviar un paquete a través del *router*, ese paquete no será recogido por el *router*, sino por la máquina atacante, ya que se dirige a su dirección MAC, y cada vez que el *router* u otro equipo envíen un paquete a la víctima sucederá lo mismo. Como la máquina atacante sabe que "está envenenando el protocolo ARP" sí conocerá las direcciones MAC reales de todas sus víctimas, por lo que la podremos configurar para que reenvíe esos paquetes a su verdadero destinatario, así **nadie notará que se ha metido en medio**.

Debido a que absolutamente toda la información de la víctima pasa por el equipo del atacante, este es capaz de **leer y modificar en tiempo real absolutamente todos los paquetes**, desde leer y capturar credenciales, como son las de páginas de *e-mails* o bancos, pasando por la modificación de conversaciones de chat, solicitudes a páginas web, inclusive hasta redirigir una consulta a un host que contenga código malicioso, para que una vez ejecutado pueda **tomar control del equipo víctima**.

Simulando un punto de acceso inalámbrico público

Un modelo de ataque dirigido sobre todo a los usuarios de dispositivos móviles se basa en la simulación de un punto de acceso inalámbrico en una red inalámbrica pública, como las de las cafeterías o las de los aeropuertos. En ello, un atacante configura su ordenador de tal manera que este se convierta en una vía adicional para acceder a Internet (probablemente una con una calidad de señal mejor que el propio punto de acceso). De esta manera, si el atacante consigue engañar a los usuarios más ingenuos, este puede acceder y manipular la totalidad de los datos de su sistema antes de que estos se

transmitan al verdadero access point o punto de acceso. Si este requiere autenticación, el hacker recibe para ello los nombres de usuario y contraseñas que se utilizan en el registro. El peligro de convertirse en el blanco de estos ataques man in the middle se da particularmente cuando los dispositivos de salida se configuran de tal manera que se pueden comunicar automáticamente con los puntos de acceso con mayor potencia de señal.

MITM en Internet

Instalando extensiones maliciosas en los navegadores

El atacante instala malware en el navegador de los usuarios de Internet con el objetivo de interceptar sus datos. Si se introducen programas en el navegador de un usuario de forma clandestina, estos registran en un segundo plano todos los datos que se intercambian entre el sistema de la persona que ha sido víctima del ataque y las diferentes páginas web. De esta manera, esta modalidad de ataque hace que los hackers puedan intervenir en una gran cantidad de sistemas con relativamente poco esfuerzo. En ello, el espionaje de datos suele tener lugar, por lo general, antes de que se lleve a cabo una posible codificación del transporte de datos mediante protocolos como TLS o SSL.

Montando un proxy

Es usual que los usuarios busquen proxys para navegar anónimamente por Internet. Los hackers pueden aprovecharse de esto para montar un proxy, normalmente en países en los que tienen una legislación favorable en estos temas, para aprovecharse para robar información o inyectar malware a los usuarios que se conectan a él.

4. Tipos de ataque que se pueden realizar con MITM

Sniffing y recolección de credenciales

Se trata de una técnica por la cual se puede "escuchar" todo lo que circula por una red. Se capturan, interpretan y almacenan los paquetes de datos que viajan por la red, para su posterior análisis (contraseñas, mensajes de correo electrónico, datos bancarios, etc.).

Realizando ataque con comandos

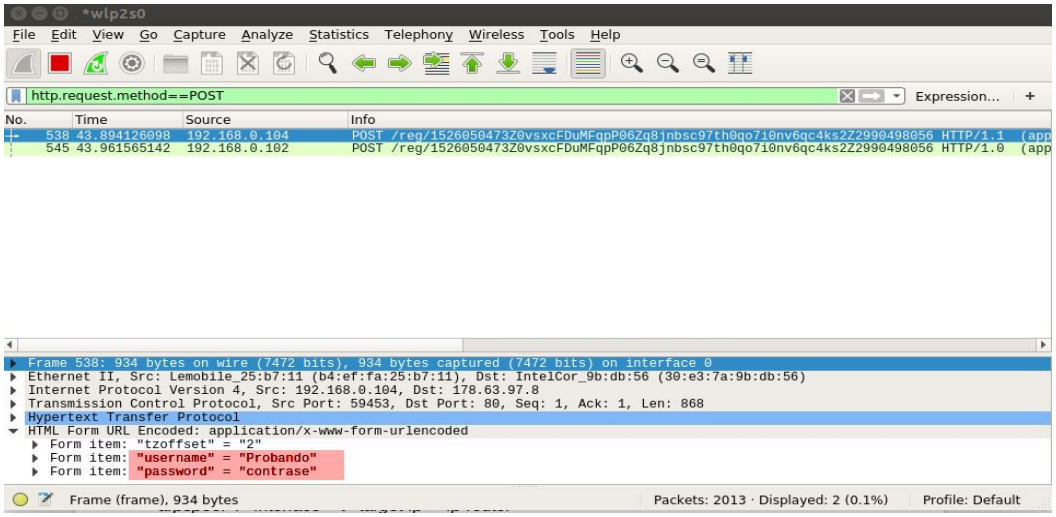
1) Primero activamos el enrutamiento del PC
echo “1” > /proc/sys/net/ipv4/ip_forward

2) Hacemos que el tráfico que vaya al puerto 80 se vaya a otro puerto
iptables -t nat -A PREROUTING -p tcp –destination-port 80 -j REDIRECT –to-port <listenPort>

3) ARPspoof consiste básicamente en inundar la red con paquetes ARP indicando que nuestra mac address es la asociada a la IP de nuestra víctima y que nuestra MAC está también asociada a la IP del router (puerta de enlace) de nuestra red. De esta forma, todas las máquinas actualizarán sus tablas con esta nueva información maliciosa.
arpspoof -i <interface> -t <target-ip> <ip-router>

4) *Utilizamos ssltstrip para intentar escuchar también el tráfico https*
python sslstrip.py -l <listenPort>

5) Ejecutando wireshark veremos los paquetes que envía nuestra víctima:

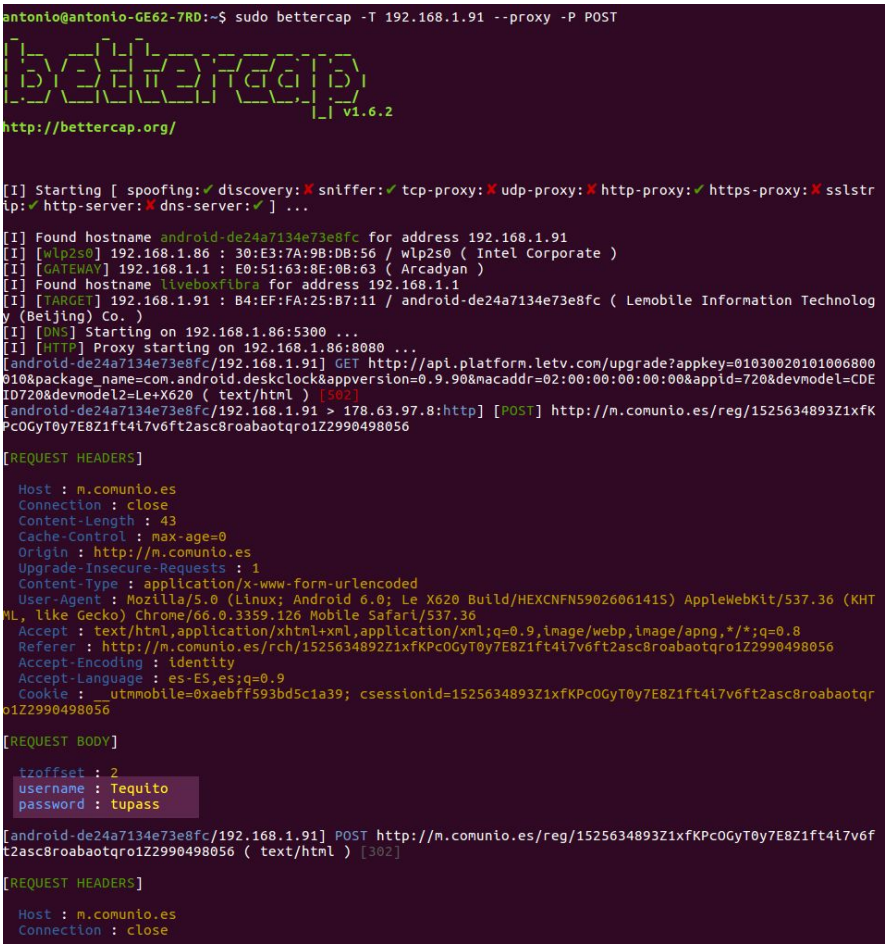


Realizando ataque con herramientas

Pero hay programas que realizan todo lo anterior **automáticamente**:

Una de las herramientas es **Bettercap**:
Con Bettercap lo único que tenemos que hacer para realizar un ataque MITM y conseguir credenciales es atrapar el tráfico con:

sudo bettercap -T <Target-IP> --proxy -P POST



Bettercap cuenta con SSLStrip por lo que las páginas HTTPS que no cuenten con HSTS serán vulnerables, como explicaremos a continuación.

SSL Stripping

Moxie Marlinspike presentó en el Black Hat 2009 una ingeniosa herramienta llamada SSLStrip, dirigida a **hacer creer al usuario que se encuentra en un sitio web con cifrado SSL cuando en realidad todos los datos están siendo transmitidos en abierto**.

El funcionamiento de **SSLStrip** es simple, **reemplaza todas las peticiones HTTPS de una página web por HTTP** y luego hace un MITM (ataque "Man in the Middle") entre el servidor y el cliente. La idea es que la víctima y el agresor se comuniquen a través de HTTP, mientras que el atacante y el servidor, se comunican a través de HTTPS con el certificado del servidor. Por lo tanto, el atacante es capaz de ver todo el tráfico en texto plano de la víctima.

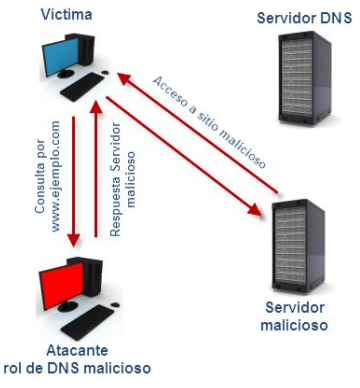
Este método se puede evitar configurando HSTS en la parte del servidor web como explicaremos más adelante (en la sección 5 ¿Cómo podemos protegernos?)

DNS Spoofing

DNS spoofing es un método para alterar las direcciones de los servidores DNS que utiliza la potencial víctima y de esta forma poder tener control sobre las consultas que se realizan.

Los servidores DNS **permiten la resolución de nombres en direcciones IP**. Aprovechando esta dependencia que existe con los servidores de nombres de dominio, **alteran las direcciones IP de los servidores DNS** de la víctima para que apunten a servidores maliciosos. Esta actividad maliciosa es conocida bajo el nombre de DNS Spoofing.

En las siguientes imágenes podemos ver como es el ciclo cuando se realiza una consulta a un servidor DNS frente se realiza cuando hay DNS spoofing.



¿Qué tipos de ataque se pueden realizar con DNS Spoofing?

Uno de los ataques que puede tener más impacto es el de montar sitios falsos que sean réplica de aquellos que el ciberdelincuente desee obtener información sensible por parte de la potencial víctima. De esta forma cuando el usuario intente acceder a ese sitio, será redireccionado al sitio espejo y el atacante obtendrá las credenciales. Este tipo de ataques incluso darán resultado sobre aquellos sitios que cifran la conexión, es decir que utilizan el protocolo HTTPS. En otras palabras, como el usuario atacado accede a un sitio malicioso, el cifrado es inexistente y por la tanto se posibilita el robo de las credenciales.

Otro ataque posible **es la explotación de alguna vulnerabilidad**. El atacante incorpora en el servidor malicioso algún tipo de *exploit* para que cuando el usuario víctima acceda al supuesto sitio legítimo, sea vulnerado.

Side-Channel attack

¿Qué es side-channel?

Cuando tenemos datos secretos a los que queremos acceder, pero están protegidos, por ejemplo, por cifrado ¿Tenemos alguna posibilidad más allá del descifrado? Si, por ejemplo, side-channel, puedo fijarme en el entorno y en lo que sí veo de forma que, mediante alguna analogía, cuando algo cambie en el entorno significará que algo cambia en el contenido y puedo intentar deducir estos cambios y por qué suceden para deducir, finalmente, el contenido que está cifrado sin romper el cifrado en sí. (Por ejemplo, un sistema de autenticación que vaya comprobando carácter a carácter si la contraseña es correcta. El cómputo en realidad es mínimo (para esta comprobación), pero aun así no tardará lo mismo para comprobar el primer dígito (si el primero no coincide deja de comprobar) que para los tres primeros por ejemplo (lo que significaría que nos equivocamos en el tercero, y por ello ha comprobado ya los dos primeros) y esto se traduce en tiempo de respuesta variable y medible por nosotros para deducir la contraseña completa mediante “fuerza bruta” e inferencia.

**Históricamente se utilizaban ataques side-channel físicos para hackear los codificadores de televisión por cable y así poder replicar esas claves criptográficas que te permiten el acceso a dicha televisión. **

En HTTPS:

En HTTPS no podemos ver el contenido (cifrado) pero sí que podemos ver (captando los paquetes obviamente) ciertos datos que no se cifran, como las IPs, el tipo de paquete, protocolo usado, el tiempo de respuesta y el **tamaño del paquete**.

Como hemos dicho para hacer un ataque side-channel necesitamos poder generar estímulos y poder ver las respuestas para monitorizarlas. Una de las maneras más sencillas para esto es estar en medio de la comunicación entre cliente y servidor (Man In The Middle).

Una forma de llevar a cabo este ataque (generar estímulos sin que ninguna de las dos partes se enteren) es, por ejemplo, inyectando javascipt a la víctima, de forma que el navegador de la víctima genere los estímulos que quiero y yo voy viendo las respuestas del servidor (para que ninguno de los dos se dé cuenta de que estoy en medio de la comunicación los estímulos los tiene que producir la propia víctima)(Si por ejemplo le inyecto una imagen (por url) el navegador de la víctima intentará conectarse a esta url para descargar la imagen y aquí, siendo yo el servidor, puedo devolverle la imagen y el javascript (hay múltiples maneras como, si tengo acceso a la wifi, editar DNSs y que al conectarse con cualquier página se descargue el javascript o lo que yo le mande y luego a la página en cuestión, pues yo sería el que especifica la ip a la que conectarse: DNS spoofing))
Este javascript, siguiendo el ejemplo anterior de la contraseña, probará todo el abecedario hasta encontrar el primer dígito "bueno" correspondiente a la contraseña y lo repetirá con el segundo, el tercero, etc hasta conseguir la contraseña entera (fuerza bruta), pero este javascript se encarga de generar estímulos o mandar peticiones, no de monitorizar las respuestas para saber cuándo acierta y cuando no. En este punto nos surge el primer inconveniente: con http/2 el envío y respuesta de paquetes es asíncrono *(esto ocurre porque en http/2 el servidor si recibe 10 peticiones las va procesando y va enviando las que tenga listas, y hay más elementos externos a ti y a la víctima que influyen en el procesado de cada petición(otras peticiones de otros clientes por ej.), esto incrementa la velocidad de carga ya que es mucho más eficiente)* , de forma que si le envío 10 peticiones para 10 dígitos diferentes probablemente me vuelvan las respuestas desordenadas y no pueda con certeza saber qué paquete corresponde a qué dígito por lo que no podría hacer este ataque correctamente a no ser que lo haga de una forma diferente. Para solucionar esto en cuanto a nuestro ataque ejemplo, podemos moderar el envío de paquetes de forma que se envíe un paquete cada vez que recibamos respuesta del anterior, así podremos tenerlos en orden, aunque de una forma mucho más lenta.

Size-Based Side-Channel

Otro "problema" que nos encontramos en este tipo de ataque que hemos propuesto es la correcta medición de tiempo, ya que, como hemos dicho anteriormente, hay muchos factores externos a nosotros (ruido) que no podemos controlar ni conocer exactamente. Estos factores modificarán el tiempo de respuesta percibido por nosotros, que por poco que sea, puede hacer que nos confundamos en nuestro análisis para encontrar la contraseña de la víctima. Por lo que para simplificar y asegurar el ataque vamos a hacerlo fijándonos en el tamaño de los paquetes en vez de en el tiempo de respuesta (*size-based side-channel*). ¿Cómo podemos deducir el contenido de un paquete cifrado por su tamaño? La respuesta es **la compresión.** Si nos fijamos en cómo se comprime una frase ejemplo con gzip podemos observar que para reducir el tamaño intenta reutilizar flujos de bytes o, lo que es lo mismo, intenta buscar partes suficientemente largas que se repitan para dejar solo una y poner como referencia o puntero la otra:

{ ***token=oscar****widowert; Repitiendo el* ***token=oscar*** } podemos ver que el conjunto de caracteres "token=oscar" se repite idénticamente en la frase, por lo que se comprimiría:

{ ***token=oscar****widowert; Repitiendo el* ***(-24,11)*** } este par de números que aparece significan que la palabra que hay en ese lugar se puede encontrar en esa frase 24 caracteres hacia atrás y cogiendo 11 caracteres desde ahí (corresponde al principio de la frase y coger los 11 dígitos desde ahí que sería *token=oscar*).

Si nos fijáramos en el tamaño de las dos frases anteriores, podríamos afirmar que comprimido '(-24,11)' ocupa menos que sin comprimir 'token=oscar', y no solo eso, imaginemos que al segundo token=oscar le añadimos la palabra widowert (como el primer token=), según esta forma de compresión aun añadiendo una palabra entera el tamaño no variaría, sólo variarán los números del puntero al primer 'token=' :

{ ***token=oscarwidowert****; Repitiendo el* ***token=oscarwidowert*** }
{ ***token=oscarwidowert****; Repitiendo el* ***(-24,19)*** } ya que yendo 24 caracteres para atrás y cogiendo 19 caracteres tendríamos 'token=oscarwidowert'.

Visto esto, podemos deducir que si el primer 'token=' contiene la contraseña y el segundo 'token=' lo controlo yo mediante la url por ejemplo, puedo ir probando caracteres y cuando acierte se comprimira de tal manera que ocupará menos que si no es cierto:

{ **token=oscarwidowert**; Repitiendo el **token=o**c }
{ **token=oscarwidowert**; Repitiendo el **(-24,9)**c } como podemos ver no lo comprime porque no coincide con "la contraseña", debería ir una 's' en vez de una 'c' y eso hará que se añada un carácter más en vez de incrementar en uno el número *(y mantener el mismo tamaño)* y eso se refleja en el tamaño *(como un byte más)*.

Estamos en https y estos datos comprimidos se cifran para meterlos en el paquete, pero aun así este efecto explicado se refleja en el tamaño del contenido cifrado aunque hay que tener más precauciones para no confundirse a la hora de llevar a cabo el ataque ya que hay **cifrados de flujo** *(stream cipher)* **y cifrados por bloques** *(block cipher)*. La diferencia es que en el cifrado de flujo se cifra tantos caracteres como haya, es decir, el tamaño cifrado es fiel al tamaño original. En el cifrado por bloques no tiene por qué ser así, ya que *(imaginemos que vamos a cifrar 8 bytes comprimidos)* el tamaño final del cifrado vendrá dado por los bloques *(pongamos 10 bytes el bloque)*, no por los caracteres *(de forma que si tenemos en este caso un bloque y añadimos un byte más (9 bytes) el tamaño final cifrado será el mismo (10 bytes) fallemos (9 bytes originales comprimidos) o acertemos (8 bytes originales comprimidos)).* Como hemos dicho solo hay que tener más cuidado ya que se puede solucionar. Por ejemplo, podemos meter caracteres *(que sepamos que no son parte del secreto)* hasta detectar que se añade un nuevo bloque *(localizando así el número del último byte del bloque)* y empezar las pruebas desde aquí donde sí se verá reflejado el tamaño ya que si fallamos se añadirá un nuevo bloque.

Además nos podemos encontrar con la **codificación Huffman** *(aunque es un* ***algoritmo de compresión*** *dificulta el ataque)*, esta representa a los caracteres más frecuentes con un "código" de bits más pequeña que para los demás y puede ocurrir que al meter un carácter *(en el ataque de fuerza bruta)*, aunque sea erróneo, no se refleje en el tamaño porque la codificación cambie y "comprima por sí solo el propio carácter" en vez de como lo hemos explicado anteriormente, recodificando todos los caracteres. La solución a este problema es la doble verificación (*double check*):

{ ***… token=a####*** } *(1er check)* estamos probando el primer carácter del token (a) (se comprimiría como el ejemplo anterior 'token=a' si esa 'a' coincide, si no, no y #### se quedaría sin comprimir (nos aseguramos de que no están y los usamos de símbolo de prueba)) por lo que debería reflejarse en el tamaño.
{ ***… token=####a*** } *(2o check)* se prueba lo mismo pero con el carácter que queremos comprobar (a) al final de los caracteres "comodín" de prueba (#) que sabemos que no pertenecen. Si el tamaño es el mismo en el primer y segundo check significará que

la ‘a’ no pertenece al token, ya que, aunque el paquete tenga el mismo tamaño antes y durante los checks 1 y 2 podremos afirmar que es culpa de la codificación Huffman ya que el check 2 estamos seguros de que no forma parte de la contraseña. Al contrario, pasaría si tienen tamaños diferentes, ya que el primer check *(sería el de menor tamaño)* habría acertado y se habría comprimido como en el caso anterior y el segundo check no *(sería de mayor tamaño)* al no ser parte de la contraseña.

Esta vulnerabilidad se denominó como **CRIME** ("Compression Ratio Info-leak Made Easy" o filtración de información de ratio de compresión hecho fácil) el cual es una exploit de seguridad que consigue **atacar cookies** sobre protocolos “seguros” HTTPS y SPDY *(protocolo presentado por Google para comunicaciones más rápidas (mayor rendimiento) en capa de sesión, con HTTP/2 se queda obsoleto)* y **con la compresión activada** para poder secuestrar sesiones web autenticadas.

CRIME fue desarrollado por dos investigadores de seguridad Juliano Rizzo y Thai Duong, quienes también desarrollaron BEAST *("Browser Exploit Against SSL/TLS")*. Presentaron su exploit en la conferencia de seguridad informática Ekoparty en 2012. La demostración en esta conferencia mostraba un atacante podría llevar a cabo un ataque para recuperar las cabeceras HTTP donde se guarda la cookie de autenticación en las **peticiones**. Para prevenirlo bastaba desactivar la compresión a nivel TLS/SSL lo cual ya lo hacían unas pocas de empresas y, tras ser anunciado, lo hicieron todas mitigando así esta amenaza específica.

En 2013, en la conferencia Black Hat, salió a la luz una nueva instancia de CRIME llamada **BREACH** (*abreviatura de Reconocimiento de Browser y ex filtración mediante Compresión adaptable de hipertexto)* la cual destapaba secretos HTTPS mediante la compresión de datos HTTP incorporada en las **respuestas**.*(La compresión sobre HTTP era mucho más frecuente que la de a nivel TLS/SSL y más desde la salida de CRIME)*.
Este ataque obvia la versión de TLS/SSL y no requiere compresión a este nivel, adicionalmente trabaja contra cifrado. La única manera de complicarlo un poco es usar cifrado por bloques, pero esto solo necesita de un poco más de esfuerzo por parte del atacante *(como ya hemos explicado*), no soluciona la vulnerabilidad.
Por lo que la subsanación de CRIME no bastaba para mitigar este exploit también. Para solucionarlo los clientes y servidores tuvieron que desactivar completamente la compresión HTTP, lo que reduce el rendimiento y hace “dar un paso hacia atrás” para encontrar nuevas soluciones *(como puede ser automatizar la compresión para que, estando activa, se desactive en entornos o situaciones específicas por ser de riesgo)*.
Condiciones para ser vulnerable a BREACH:

- Conectarte con un servidor que te responda usando compresión a nivel HTTP.
- Reflejar el input del usuario en el cuerpo de la respuesta HTTP.
- Reflejar al mismo tiempo un secreto en el cuerpo de la respuesta HTTP:

(En consecuencia, podemos ver que no debemos mezclar en el mismo paquete o sección entrada o datos de usuario (input) junto a datos secretos como puede ser una cookie, contraseña o cualquier dato privado y más si usamos compresión sobre lo que lo contiene).
Soluciones contra BREACH:

- Desactivar la compresión HTTP.
- Separar datos secretos del input del usuario.
- Aleatorizar los secretos por cada petición.
- Enmascarar los secretos (por ej. cifrarlos por sí solos antes de compresión).

- Esconder el tamaño de los paquetes añadiendo números de bytes aleatorios en las respuestas.
- Proteger páginas vulnerables contra CSRF(*Cross-site request forgery* o falsificación de petición en sitios cruzados, donde las acciones que quiero llevar a cabo como atacante se las hago hacer a la víctima por ejemplo mediante enlaces "maliciosos" que accedan, por el ya estar autenticado, a cierto sitio llevando a cabo cierta acción) (ejemplo: inyecto una imagen con la url : *<img src="http://empresaVictima.com/usuarios/getContraseñas/all">)*
- Limitar el ratios de peticiones (ya que este ataque se basa en fuerza bruta mediante peticiones)

Cross-Sites Searches

Vistos estos ataques de side-channel basado en el tamaño por compresión de los paquetes podemos intentar ampliar el radio de ataque de este tipo de side-channel *(observando el tamaño)* sin usar compresión *(el no usar compresión de ningún tipo, por protección a estos ataques vistos, está ampliamente extendido hoy en día)*. Entonces, nos vamos a fijar sólo en el tamaño y vamos a destinar nuestro nuevo ataque a las búsquedas.
Cuando realizamos una búsqueda mandamos un paquete con lo que queremos y el servidor en cuestión nos responde con los resultados de esa búsqueda, si esta búsqueda no tiene resultados el paquete que nos va a devolver solo va a decir "No he encontrado nada" pero si tiene resultados nos enviará el paquete con los diferentes resultados. Parece lógico que el primer paquete sin resultados será muy liviano, pues no contiene nada *(ningún elemento al no haberlo encontrado)* y el segundo será más pesado al contener los diferentes elementos que haya encontrado. Mediante esta analogía y con motores que permitan búsquedas específicas podríamos, por ejemplo, si se tratase de una agenda de contactos, ir probando número a número para ver los números que contiene la agenda. *(Imaginemos que la agenda contiene sólo el número 633 33 33 33 (asociado a un contacto), buscando por el primer número (resultado para los contactos con número de teléfono que empiece por 'x'), sólo obtendremos el contacto asociado al número (paquete más grande) cuando busquemos el '6', por lo que confirmaremos el primer dígito, probando con los dos primeros dígitos, sólo obtendremos resultados con el '3', es decir, '63' y así sucesivamente hasta completar el número).*
Las páginas que permiten este tipo de búsquedas no solían implementar protección contra CSRF (*Cross-site request forgery)* en las funciones de consulta *(si se protegen ampliamente otras opciones)* porque se supone que con HTTPS nadie puede leer la información devuelta (cifrada), pero como podemos ver no es así.

No tenemos los problemas asociados a la compresión como que esté desactivada, que use codificación huffman, etc porque no necesitamos el uso de compresión para este ataque. Además, para cifrado por bloques tampoco tendremos problemas ya que los bloques suelen ser como mucho de 16 bytes (16 dígitos) y un contacto suele contener entre nombre, apellidos, número, dirección, foto asociada, etc bastante más de 16 bytes.

También se podría aplicar a este ataque midiendo el tiempo en vez del tamaño ya que los paquetes pesados tardan más en llegar (incluso a veces hay que partir la respuesta en varios paquetes) además que se tardaría más en procesar por parte del servidor (ataque de este tipo presentado por Nethanel Gelernter en la Black Hat 2016).

Pero hoy en día este tipo de ataque está bien subsanado, al menos por las grandes empresas, ya que, por ejemplo, Google al realizar una búsqueda te devuelve parte del paquete de forma aleatoria, es decir, para una misma búsqueda y para unos mismos resultados el tamaño varía.

Connection Bassed Side-Channel

Aunque no podemos realizar este ataque por tamaño o por tiempo *(derivado del tamaño, al fin y al cabo)* la idea del ataque cross-site por búsquedas se puede afinar *(esto pensó Jose Selvi al desarrollar su nuevo método)*. Poniendo de ejemplo la agenda de Google, podemos observar que cuando haces una consulta se carga una imagen por cada contacto *(una imagen asociada)* y, por tanto, el navegador tendrá que cargar una imagen diferente por cada contacto pidiéndolas a el servidor de Google donde estén almacenadas. Esto al final se traduce en diferentes conexiones que se hacen para descargar estas imágenes y observando estas conexiones podría deducir, mediante la misma analogía que en cross-side searches, el contenido de la agenda.

El primer problema se presenta con la caché. Ya que los navegadores, cuando descargan una imagen, la cachean por si la vuelves a necesitar durante cierto tiempo *(24 horas usualmente)*. Por lo que si pruebo diferentes consultas sólo podré deducir algo por las conexiones la primera vez que vaya a cargar la imagen en cuestión, la segunda no se conectará a ningún lado y la cojera localmente de su memoria. Pero como se supone que estamos llevando estos ataques estando en medio *(man in the middle)* podemos buscar soluciones para esto.
La primera solución se basa en que todas las conexiones pasan por mí, y soy yo el que permite que se lleve a cabo cada conexión, por lo que, cuando la víctima vaya a conectarse a Google para descargar una imagen *(tras una consulta)* puedo denegarla de forma que su navegador nunca llegue a descargar la imagen *(en este punto yo ya sé que él está haciendo petición de imagen, por lo que sé que hay resultados en la búsqueda hecha)*. De esta forma la próxima vez que necesite la foto que ha intentado descargar, lo volverá a intentar porque aún no la tiene guardada y yo volveré a saber que está intentando conectarse para descargarla.
Otra forma de solucionar esto *(y en combinación con la anterior solución puede servir para cuando empezamos un ataque y ya tiene contenido cacheado)* es, repito, como estamos en medio de la conexión, enrutarlo para que se conecte con otro CDN (*content delivery network o red de distribución de contenidos) (de Google en esta caso)*, de forma que si se conecta en otro país necesitará las imágenes de otro servidor (CDN) y, aunque las tenga cacheadas, las volverá a intentar descargar pues la caché entiende que es una dirección diferente.

Cuando Jose se puso a probar esta vulnerabilidad sobre las grandes empresas tecnológicas ratificó que Google Inbox *(aplicación de Google para acceso a correos Gmail)*, Google Drive, Dropbox *(con la diferencia que carga imagen cuando no encuentra resultados)* y el messenger dentro de la página de Facebook eran vulnerables a este ataque. Aunque les notificó a todas ellas, todas menos Google decidieron solucionar esta vulnerabilidad. Lo que significa que a día de hoy Facebook y Dropbox no pueden sufrir este ataque, pero Google sigue siendo vulnerable y seguirá siéndolo hasta que crean que es realmente una amenaza. De hecho, en su ponencia en Cybercamp 2018, presentó su

herramienta (FIESTA) explotando la agenda de contactos de Google a modo de Demo *(una demo muy real)* y demostró que realmente podía sacar toda la información que quisiera de dicha agenda *(se puede ver la conferencia completa el Youtube, enlace en las referencias 1)*.

La herramienta creada por Jose Selvi (FIESTA) (https://github.com/PentesterES/fiesta)

A modo de *datos interesantes* decir que en la demostración utilizó el primer ataque (size-based) y el último ataque descrito (connection-based). En este último hay varios matices, por ejemplo, es necesario que un script se esté ejecutando en la víctima y (con Google) sólo es posible mediante un tab (más escondido) o una ventana diferente (pop-up) a la que esté usando la víctima (para que no vea el ataque) y donde se vayan ejecutando las diferentes consultas para la extracción de información. Además, implementó un pequeño mecanismo de seguridad que hacía que si la víctima iba a la ventana/pestaña del ataque o simplemente ponía el mouse sobre la pestaña donde está ocurriendo el ataque (se entiende como intención de ir a dicha página) el ataque se pausaba, de forma que si se detectara algo raro visualmente e ibas a comprobarlo no se vea nada raro, pero en cuanto cambias de ventana vuelve a iniciarse el ataque.
También decir que este ataque es un poco lento en comparación con otros, por ello también implementó un auto ajuste de velocidad para la aplicación, ya que para este ataque se tiene que esperar a que se renderice toda la página. Entonces lo que hace la aplicación es, al principio, hacer un cálculo de cuánto tarda en renderizar para diferentes consultas (con y sin resultado devuelto) para reducir al mínimo el tiempo que espera la aplicación Fiesta en volver a hacer una consulta sin perder información de la consulta anterior.

5. ¿Cómo defenderse de los ataques MITM?

Utilizar HTTPS

HTTPS (Protocolo Seguro de Transferencia Hipertexto) utiliza un cifrado basado en SSL/TLS con el fin de crear un canal cifrado entre el cliente y el servidor.

Transport Layer Security (TLS), que es una infraestructura de clave pública, fortalece el Protocolo de control de transmisión contra los ataques MITM. TLS ayuda a autenticar a las partes a través de una autoridad certificadora (CA) mutuamente confiable. Los clientes y servidores adquieren certificados SSL / TLS de CA fiables, por lo que el intercambio de certificados permite la autenticación mutua.

HSTS

HTTP Strict Transport Security o HTTP con Seguridad de Transporte Estricta (HSTS), es una política de seguridad web establecida para evitar ataques que puedan interceptar comunicaciones, cookies, etc. Según este mecanismo un servidor web declara que los agentes de usuario compatibles (es decir, los navegadores), solamente pueden

interactuar con ellos mediante conexiones HTTP seguras (es decir, en HTTP sobre TLS/SSL.

La política HSTS es comunicada por el servidor al navegador a través de un campo de la cabecera HTTP de respuesta denominado "Strict Transport-Security". La política HSTS especifica un período de tiempo durante el cual el agente de usuario deberá acceder al servidor sólo en forma segura.

En otras palabras, si cuando un usuario quiere acceder a un sitio web cómo puede ser ***Gmail*** o ***Facebook***, éste no introduce en la barra de direcciones ***URL*** el protocolo con ***HTTPs***, por ejemplo "***https://gmail.com***" o "***https://facebook.com***", sino que introduce simplemente ***gmail.com*** o ***facebook.com***, entonces el navegador automáticamente fuerza la conexión ***HTTPs***.

Navegadores soportados

Entre los navegadores que soportan HSTS se encuentran:
- Google Chrome desde la versión 4.0.211.0.
- Google Chrome para Android desde la versión 18.
- Firefox y Firefox Mobile desde la versión 4.
- Opera desde la versión 12.
- Safari desde la versión 7.
- Android Browser desde la versión 4.4 de Android.
- Internet Explorer planea implementarlo en la versión 12 de su navegador.

Implementando HSTS

La implementación de HSTS es bastante sencilla, y daremos ejemplos para las tres principales plataformas web del mundo.

-) En el caso de **Apache**, será necesario agregar al archivo **.htaccess** la siguiente línea:

Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"

-) **Nginx** también soporta la utilización de HSTS, declarando la siguiente línea en el archivo **nginx.conf**:

add_header Strict-Transport-Security "max-age=31536000; includeSubDomains";

-) Para sitios alojados en IIS, el servicio web de Windows Server, debemos agregar el siguiente fragmento en el archivo **web.config** del sitio:

```
<system.webServer>
<httpProtocol>
<customHeaders>
<add name="Strict-Transport-Security" value="max-age=31536000"/>
</customHeaders>
</httpProtocol>
</system.webServer>
```

En todos los casos, el parámetro **max-age** indica cuánto tiempo (en segundos) el navegador del usuario deberá comunicarse via HTTPS con el servidor de destino. En todos los ejemplos hemos utilizado como período 1 (un) año y el contador se reiniciará cada vez que el usuario vuelva a ingresar al sitio.
Por último, el parámetro **includeSubDomains** obliga al navegador del usuario a validar que la comunicación sea establecida de manera segura en todos los subdominios del sitio.

Evitar el uso de extensiones sospechosas en los navegadores.

Aunque muchas extensiones para navegador nos sirven de mucha ayuda, como para evitar anuncios molestos, autocompletar información en compras, etc. También hay otras que redirigen el tráfico a un servidor proxy en el que intentan robarnos la información.

Aún así hay otras extensiones que nos ayudan a mantenernos algo más seguros, como por ejemplo HTTPS Everywhere, que es una extensión de navegador gratuita y de código abierto para Mozilla Firefox, Google Chrome y Opera, una colaboración por el proyecto Tor y la Electronic Frontier Foundation. Su propósito es hacer de forma automática que sitios web utilicen la conexión HTTPS, más seguro en lugar de HTTP.

No conectarse a redes wifi públicas.

El peligro de las redes wifi públicas es que cualquier persona puede conectarse a ellas, y por lo tanto cualquiera puede ver los dispositivos que están conectados a ella y realizar un ataque MITM.

Actualizar nuestro software.

Una de las mejores formas de evitar que nos ataquen es actualizar todos los componentes de software de nuestro sistema, ya que así se reducirá el número de vulnerabilidades en nuestro equipo.

6. Conclusiones

Como hemos podido comprobar personalmente, estos tipos de ataques son mucho más sencillos de lo que pueda parecer en un principio, sobre todo de lo que pueda parecer a una persona que no tenga grandes conocimientos sobre informática. Por lo que nos parece de suma importancia que se eduque en el mundo de la ciberseguridad a todo aquel que vaya a hacer uso del “mundo ciber”.
En cuanto a los ataques de side-channel nos hemos dado cuenta que, aunque son ataques lentos (sobre todo si queremos sacar mucha información), son eficaces y lo “peor” de este tipo de ataques es que es muy difícil protegerse sin prescindir de otras funcionalidades como puede ser la eficiencia. Además es difícil protegerse contra ellos

porque salen muchas “ideas” diferentes de como poder inferir información de cualquier dato conocido y para cada modo se necesita una protección específica como hemos visto.

7. Referencias

¿Qué es el ataque Man-in-the-Middle?

1. https://securebox.comodo.com/ssl-sniffing/man-in-the-middle-attack/

¿Como se realiza?

1. https://www.welivesecurity.com/la-es/2014/02/11/como-funciona-arpspoof/
2. https://www.ionos.es/digitalguide/servidores/seguridad/ataques-man-in-the-middle-un-vistazo-general/

Bettercap

1. https://github.com/bettercap/bettercap
2. https://securityhacklabs.net/articulo/bettercap-una-herramienta-completa-modular-y-de-facil-uso-para-realizar-ataques-man-in-the-middle

SSL Stripping

1. https://www.redeszone.net/seguridad-informatica/sslstrip/

DNS Spoofing

1. https://www.welivesecurity.com/la-es/2012/06/18/dns-spoofing/

¿Cómo defenderse de los ataques MITM?

1. https://geekytheory.com/como-funciona-https
2. https://blog.pablofain.com/2014/08/25/que-es-hsts-y-como-implementarlo-para-incrementar-la-seguridad-de-mi-sitio/

SIDE CHANNEL:

FUENTE PRINCIPAL: JOSE SELVI, 15 años en seguridad informática, ingeniero en informatica y telecomunicaciones, consultor principal de seguridad e investigador de seguridad en el NCC group, experto en seguridad GIAC (GSE) y ha dado ponencias en DEFCON, Blackhat (europa), Ekoparty, etc.

1. Conferencias (talleres) CyberCamp 2018, día 27 septiembre (a partir de 06:07:00) https://www.youtube.com/watch?v=-TpUf4Bp5xw, ponencia Jose Selvi.
2. Información sobre CRIME, Wikipedia https://es.wikipedia.org/wiki/CRIME
3. Información sobre BEAST, Wikipedia https://es.wikipedia.org/wiki/Transport_Layer_Security#Ataque_BEAST
4. Información sobre SPDY, Wikipedia https://es.wikipedia.org/wiki/SPDY
5. Información sobre CSRF , Wikipedia https://es.wikipedia.org/wiki/Cross-site_request_forgery
6. Página oficial de BREACH, Wikipedia http://breachattack.com/
7. Información sobre BREACH, Wikipedia https://es.wikipedia.org/wiki/BREACH_(ataque)
8. Conferencia ekoparty 2012, CRIME https://www.youtube.com/watch?v=BysvLotMrwY

9. Conferencia ekoparty 2012, CRIME presentación en PDF https://www.ekoparty.org/archive/2012/CRIME_ekoparty2012.pdf
10. Conferencia black hat 2013, BREACH https://www.youtube.com/watch?v=CoNKarq1IYA
11. Conferencia black hat 2013, BREACH presentación en PDF https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-seconds-A-BREACH-beyond-CRIME-Slides.pdf
12. Conferencia black hat 2016, Nethanel Gelernter cross-site time-based, presentación en PDF https://www.blackhat.com/docs/us-16/materials/us-16-Gelernter-Timing-Attacks-Have-Never-Been-So-Practical-Advanced-Cross-Site-Search-Attacks.pdf
13. Información sobre CDN WIkipedia https://es.wikipedia.org/wiki/Red_de_distribuci%C3%B3n_de_contenidos
14. Github al software de Jose Selvi, FIESTA https://github.com/PentesterES/fiesta