TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE

- Qué es JWT
- Por qué debemos agregar seguridad a nuestros servicios
- El modelo stateless
- Seguridad con JWT
- Firma de token
- Clave del servidor
- Proceso de autenticación
- Tiempo de vida y caducidad de token
- JWT vs. variables de sesión y cookies

¿QUÉ ES JWT?

A JSON Web Token (JWT) por sus siglas en inglés, es un estándar utilizado para compartir información de forma segura entre dos partes — un cliente y un servidor. Cada JWT contiene objetos JSON codificados, incluido un conjunto de notificaciones. JWT puede ser utilizado para un sistema de autenticación y también se puede utilizar para el intercambio de información.

Los JWT se firman digitalmente con un algoritmo criptográfico, utilizando un par de claves secretas o públicas/privadas, para garantizar que los notificaciones no se puedan modificar después de que se emita el token.

JWT es un estándar que define una forma compacta y autónoma de transmitir información. El tamaño compacto hace que los tokens sean fáciles de transferir a través de una URL, un parámetro POST o dentro de un encabezado HTTP. La información en un JWT se firma digitalmente.

¿POR QUÉ DEBEMOS AGREGAR SEGURIDAD A NUESTROS SERVICIOS?

Actualmente, existe una infinidad de servicios disponibles en Internet a través de una innumerable variedad de dispositivos, siendo todos estos el blanco de los llamados piratas informáticos. Sitios web que dejan de ofrecer su contenido, hurto de información confidencial, filtración de direcciones de correo electrónico,

autenticación de aplicaciones y bases de datos vulnerables, además de un sin fin de inconvenientes que se pueden afrontar si los servicios se encuentran desprotegidos o no están lo suficientemente protegidos.

La seguridad de los servicios disponibles a través de Internet se considera de vital importancia, pues no existe una red que sea completamente inmune a los ataques cibernéticos. Proteger los datos de los clientes y los de la propia organización con un sistema de seguridad de red, aplicaciones estable y con el máximo de eficiencia es verdaderamente esencial.

Hoy día los estándares y protocolos abiertos que se utilizan para intercambiar datos entre aplicaciones son los servicios web. REST es uno de los estándares más eficientes y habituales en la creación de APIs para servicios de Internet, por tanto es común que para la creación de servicios de comunicación e intercambio de datos se utilicen API REST.

No obstante, REST también tiene vulnerabilidades similares a las de una aplicación web. Datos de tarjetas de crédito, información financiera, información comercial y otras categorías necesitan protección. Establecer métodos de autenticación de usuarios para acceder a recursos tecnológicos es una característica clave para la seguridad de la información.

Generalmente los ataques informáticos en la web se dan en la capa de aplicación y estos constituyen un número importante de los intentos observados en Internet contra aplicaciones web. La seguridad de la información se considera importante, pues permite gestionar los activos de información y controlar de mejor manera sus riesgos, en relación con el impacto que representan para una organización. Éste es un tema que se ha debatido cada vez con mayor frecuencia en blogs y foros de tecnología en vista de las numerosas infracciones de seguridad de alto perfil han crecido significativamente.

La seguridad informática es relevante, especialmente en el mundo de las API REST. Los problemas de seguridad deben ser un aspecto importante a tener en cuenta cada vez que se diseña, prueba e implementa una API REST. La seguridad de los datos confidenciales ya sea información organizacional o personal, es un factor importante que preocupa a los desarrolladores hoy en día y que por ende no se debe subestimar.

Las API REST forman parte de sistemas esenciales que requieren protección contra amenazas e infracciones de seguridad. Dado que se utilizan comúnmente para intercambiar información y posiblemente se ejecuta en muchos servidores, podría dar lugar a gran cantidad de brechas invisibles y fugas de información. Un atacante podría estar en el lado del cliente (consumidor de API REST) o en el lado del servidor si el atacante logra obtener el control sobre el servidor, donde puede crear una aplicación maliciosa. La víctima, en este caso, es la aplicación que consume recursos de servicios remotos. Para una aplicación que utiliza REST como cliente o servidor, el otro lado generalmente tiene control total sobre la

representación de recursos y podría inyectar cualquier carga útil para atacar el manejo de recursos (como obtener código fuente o ejecución de comandos del sistema).

La implementación de un control de acceso eficiente permite otorgar funcionalidades y contenidos únicamente a aquellas con autorización de hacerlo. El no poseer esta característica o implementarla de manera inadecuada puede permitir a un atacante obtener el control de las cuentas de otros usuarios, alterar los privilegios de acceso, cambiar los datos, etc. El acceso no autorizado a las aplicaciones de una organización tiende a darse cuando los desarrolladores no configuran correctamente la accesibilidad a nivel de software, lo que genera vulnerabilidades de acceso. El acceso denegado es la consecuencia más conocida de los controles de acceso vulnerados y la explotación del control de acceso es el arte principal de los atacantes. El control de acceso generalmente es productivo si se implementa en una API confiable del lado del servidor, por lo que el atacante no podrá alterar los metadatos del control de acceso.

Subestimar los aspectos de la seguridad en los servicios web puede acarrear muchos problemas que desde un principio se pueden evitar aplicando buenas prácticas de protección. La seguridad en servicios Web REST es un conjunto de mecanismos, procesos, mejoras, modelos, técnicas que ayudan a mitigar el nivel de impacto de los tipos de vulnerabilidad que sufren las organizaciones y personas que manejan servicios Web.

EL MODELO STATELESS

El modelo stateless o modelo sin estado, es aquel que trata cada petición como una transacción independiente, que no tiene relación con cualquier solicitud anterior, de modo que la interacción se compone de pares independientes de solicitud y respuesta.

Bajo este concepto, el servidor no retiene información de la sesión, es decir, no se almacena información sobre las operaciones anteriores ni se hace referencia a ellas. El modelo stateless no depende de un sistema de almacenaje persistente. Cada operación se lleva a cabo desde cero, como si fuera la primera vez. El modelo sin estado proporciona un servicio o una función y usan servidores de impresión, de red de distribución de contenido (CDN) o web para procesar estas solicitudes a corto plazo.

Un ejemplo de una operación sin estado puede ser la búsqueda en línea de la respuesta a una pregunta que se le haya ocurrido. Usted escribe la pregunta en un motor de búsqueda y presiona la tecla Entrar. Si la operación se interrumpe o se cierra por accidente, simplemente inicia una nueva.

PRINCIPALES CARACTERÍSTICAS DE APLICACIONES STATELESS:

- Las aplicaciones o servicios Stateless, únicamente realizan una función.
- En este tipo de arquitecturas o servicios, el servidor únicamente se basa en la petición que se realiza con cada solicitud, sin tener en cuenta información o peticiones anteriores, es decir, el servidor no necesita información de estado de otras solicitudes.
- Al no tener estado, diferentes peticiones pueden ser tratadas por diferentes servidores.
- Por lo general este diseño de servicios o aplicaciones, suele ir ligado a arquitecturas de microservicios y contenedores.

¿QUÉ ES UN TOKEN?

Un token es una cadena alfanumérica con caracteres aparentemente aleatorios, como el siguiente:

```
edJhbGfiOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3
ODkwIiwibmFtZSI6IkpvaG4gRG9lIiaYWRtaW4iOnRydWV9.TJVA95OrM7
E2cBab30RMHrHDcEfxjoYZgeFONFh7HgK
```

O el siguiente:

```
24353689
```

Estas cadenas de texto pueden no aparentar un significado, sin embargo, tiene un significado real para el servidor o institución que lo emite, el cual puede entender y así, validar al usuario que intenta acceder a la información, e incluso, puede tener datos adicionales.

En el caso de los tokens bancarios, no se almacena una información real dentro del Token, sino que simplemente es un valor generado que luego puede ser validado por el banco como un valor real generado por nuestro token. Sin embargo, con JWT podemos enviar cualquier dato del cliente dentro del token para que el servidor pueda obtener mucha más información de nosotros

SEGURIDAD CON JWT

Un JWT es una forma conveniente de representar fragmentos de información de forma segura. Un fragmento o *claim* no es más que un par clave/valor. Por ejemplo el conjunto de declaraciones que representan la identidad del usuario se conoce como *claims y* se ubican en el contenido o *playload* del JWT.

FIRMA DE TOKEN

Los claims en un JWT a menudo se usan para operaciones sensibles a la seguridad. Es esencial evitar la manipulación de claims generados previamente. El emisor de un JWT firma el token, lo que permite al receptor verificar su integridad, siendo estas firmas cruciales para la seguridad.

Además de la firma, un JWT contiene otras propiedades de seguridad las cuales establecen la vida útil en un JWT. También identifican el emisor y el objetivo previsto, es decir la audiencia. El receptor de un JWT siempre debe verificar estas propiedades antes de usar cualquiera de los claims o fragmentos de información. Es de importancia verificar la siguiente información para garantizar la seguridad del JWT en uso:

- Verifique el fragmento `exp` para asegurarse de que el JWT no haya expirado. Por otra parte, verificar el tiempo de vida del JWT utilizando el valor del fragmento `iat` es una buena práctica.
- Verifique el fragmento `nbf` para asegurarse de que el JWT ya puede ser utilizado.
- Verifique el fragmento `iss` con su lista de emisores confiables.
- Verifique el fragmento `aud` para comprobar que el JWT es para usted.

CLAVES DEL SERVIDOR

El uso de claves para firmas y cifrado requiere un manejo cuidadoso. Las claves deben almacenarse en un lugar seguro y deben rotarse con frecuencia. La aplicación en cuestión debe prever una forma de gestionar las claves del JWT. Considere los siguientes aspectos:

- Almacene las claves en un servicio de almacén de claves dedicado. Las claves deben obtenerse dinámicamente, en lugar de estar codificadas.
- Use el fragmento `kid` en el encabezado para identificar una clave específica.
- Las claves públicas se pueden incrustar en el encabezado de un JWT. El fragmento `jwk` puede contener una clave pública con formato de clave web JSON. El fragmento `x5c` puede contener una clave pública y un certificado `X509`.
- Valide una clave pública utilizada con una lista de claves conocidas. Una falla en el uso restringido de las claves puede incurrir en la aceptación de un JWT que provenga de un atacante.
- El encabezado también puede contener una URL que apunte a claves públicas.
- Valide una URL de clave contra una lista segura de `URL/dominios`.

JWT recomienda que la comunicación entre partes se realice con HTTPS para encriptar el tráfico, de forma que, si alguien lo interceptó, el propio tráfico a través de HTTP sobre esos sockets SSL, cifran toda la comunicación, la del token y todo lo demás, y así añadir esa posibilidad de seguridad.

PROCESO DE AUTENTICACIÓN

Autenticación es el proceso que debe seguir un usuario para tener acceso a los recursos de un sistema o de una red de computadoras. Este proceso implica identificación (decirle al sistema quién es) y autenticación (demostrar que el usuario es quien dice ser). La autenticación por sí sola no verifica derechos de acceso del usuario; estos se confirman en el proceso de autorización.

JWT es una tecnología particularmente útil para una API con autenticación y la autorización de servidor a servidor.

Las características de un JWT lo convierten en una excelente opción para la autenticación basada en tokens. Es un paquete ligero, ya que se incluirá en cada solicitud a nuestra API REST. También debe ser a prueba de manipulaciones, de modo que el reclamo de identidad no se pueda alterar en tránsito o falsificar por completo.

Una de las mayores ventajas de este enfoque es que no requiere que el cliente o el servidor REST mantengan sesiones. De hecho, no se requiere ninguna búsqueda en la base de datos para verificar la identidad del usuario solicitante. Debido a que el JWT está firmado con una clave secreta guardada en la API REST, se puede confiar implícitamente que este usuario es quien dice ser.

La última parte de un JWT es la firma, que es un código de autenticación de mensaje. La firma de un JWT solo puede ser producida por alguien en posesión tanto del contenido (más el encabezado) como de una clave secreta determinada.

Así es como se utiliza la firma para garantizar la autenticación:

- El usuario envía el nombre de usuario y la contraseña a un servidor de autenticación, que puede ser nuestro servidor de aplicaciones, pero normalmente es un servidor independiente.
- El servidor de autenticación valida la combinación de nombre de usuario y contraseña y crea un token JWT con una carga útil que contiene el identificador técnico del usuario y una marca de tiempo de vencimiento.
- El servidor de autenticación luego toma una clave secreta y la usa para firmar el encabezado más el contenido y lo envía de vuelta al navegador del usuario.
- El navegador toma el JWT firmado y comienza a enviarlo con cada solicitud HTTP a nuestro servidor de aplicaciones.
- El JWT firmado actúa efectivamente como una credencial de usuario temporal, que reemplaza la credencial permanente que es la combinación de nombre de usuario y contraseña.

Y a partir de ahí, esto es lo que hace nuestro servidor de aplicaciones con el token JWT:

- Nuestro servidor de aplicaciones verifica la firma JWT y confirma que, de hecho, alguien en posesión de la clave secreta firmó este contenido en particular.
- El contenido identifica a un usuario en particular a través de un identificador técnico.

- Solo el servidor de autenticación está en posesión de la clave privada, y el servidor de autenticación solo entrega tokens a los usuarios que envían la contraseña correcta.
- Por lo tanto, nuestro servidor de aplicaciones puede estar seguro de que este token fue entregado a este usuario en particular por el servidor de autenticación, lo que significa que es el usuario, ya que tenía la contraseña correcta.
- El servidor procede a procesar la solicitud HTTP asumiendo que realmente pertenece a ese usuario.

La única forma de que un atacante se haga pasar por un usuario sería robar su nombre de usuario y contraseña de inicio de sesión personal, o robar la clave de firma secreta del servidor de autenticación.

Como podemos observar, la firma es realmente la parte clave del JWT. La firma es lo que permite a un servidor completamente sin estado estar seguro de que una solicitud HTTP determinada pertenece a un usuario determinado, con solo mirar un token JWT presente en la solicitud en sí, y sin forzar que la contraseña se envíe cada vez con la solicitud.

TIEMPO DE VIDA Y CADUCIDAD DE TOKEN

El token generalmente tiene un tiempo de validez, que se establece en el servidor, al momento de generar el token.

Durante el tiempo de validez del token podrá ser utilizado para autorizar el acceso a ciertos recursos por parte del cliente autenticado. Sin embargo, si el token ha caducado, entonces el servidor lo rechazará y el cliente no podrá acceder a aquella información u operación solicitada.

Para establecer un flujo mediante el cual se mitiguen posibles problemas por caducidad del token, se puede proporcionar también un token de refresco.

Ese token servirá para aumentar el tiempo de validez del token o generar uno nuevo más adelante, sin necesidad de obligar al usuario a volver a enviar sus datos de inicio de sesión. El token de refresco se utiliza opcionalmente, por lo que no todas las implementaciones deben controlarlo necesariamente.

JWT VS VARIABLES DE SESIÓN Y COOKIES

Comencemos recordando que el protocolo HTTP es un protocolo "sin estado"(*stateless*), en el que cada llamada HTTP realizada al servidor es como una nueva llamada, pues no hay información almacenada acerca de conexiones establecidas previamente.

En las aplicaciones generalmente es necesario mantener sesiones de inicio de sesión, de tal manera que una vez que el usuario se conecta con sus credenciales, puede continuar y acceder a diferentes partes de la aplicación sin tener que autenticarse una y otra vez por cada solicitud HTTP que se realice al servidor.

Existen principalmente dos enfoques que son fundamentalmente distintos y totalmente completos para gestión de sesiones:

- El enfoque de sesiones y cookies
- El enfoque basado en JWT (JSON Web Tokens)

COMPARACIÓN DE AMBOS ENFOQUES:

Después de una autenticación satisfactoria, (en el caso del enfoque sesión-cookie) el servidor genera una "cookie", mientras que en el caso del enfoque JWT el servidor genera un "accessToken". Una vez el usuario es autenticado por ejemplo el servidor verifica que las credenciales utilizadas correspondan con las almacenadas en la base de datos para el usuario en cuestión.

ENFOQUE BASADO EN SESIÓN-COOKIE:

1. El servidor genera un id de sesión (que es firmada con una clave secreta), y:
   a) Guarda el id de sesión en la base de datos correspondiente.
   b) Envía una cookie con el id de sesión al navegador (cliente)
2. El navegador (cliente) recibe la cookie en la respuesta del servidor y la guarda en el almacenamiento de "cookies".
3. El navegador luego incluye la cookie dentro de cada solicitud posterior al servidor.

ENFOQUE BASADO EN JWT:

1. El servidor genera un "token de acceso", cifrando el `"userId"` y `"expiresIn"`, con `ACCESS_TOKEN_SECRET`, y envía el "token de acceso" al navegador (cliente).
2. El navegador (cliente) recibe el "token de acceso" y lo guarda en el lado del cliente.
3. El "token de acceso" se incluye en cada solicitud posterior al servidor.

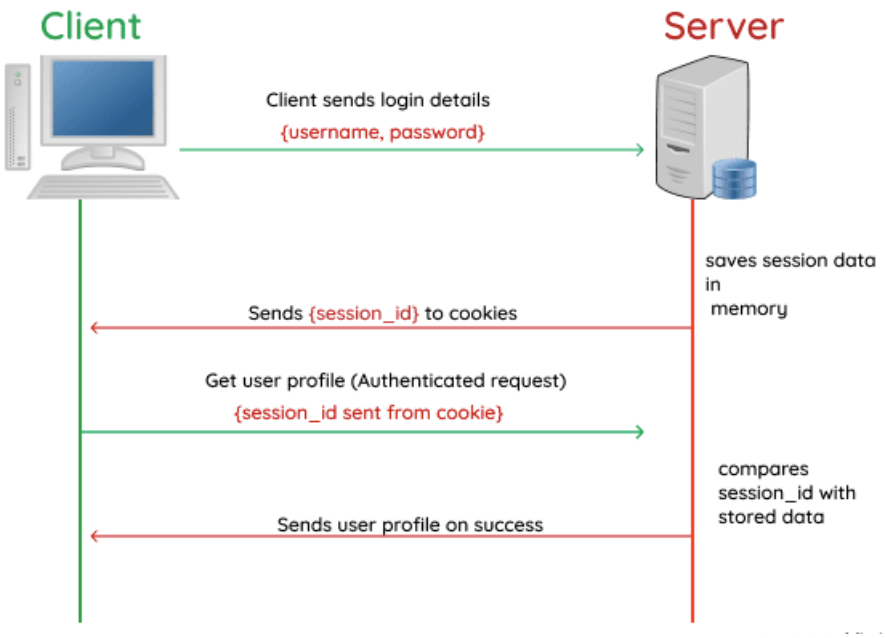
En ambos enfoques, el lado del cliente debe almacenar de forma segura la “cookie” o el “jwt token”. La principal diferencia es que en el caso del enfoque JWT el servidor no necesita mantener una base de datos de identificadores de sesiones.

Para mantener las sesiones, cada solicitud posterior al servidor (en el caso del enfoque sesión-cookie) debe incluir la “cookie (el id de sesión)”, mientras que (en caso del enfoque JWT) debe incluir el “token de acceso”.

Ahora que el usuario es autenticado (y el cliente tiene bien sea una cookie o un token de acceso), digamos que el usuario que inició sesión desea realizar un variedad de acciones, como agregar artículos a un carrito de compras por ejemplo. Para cada solicitud realizada al servidor, éste hace lo siguiente:

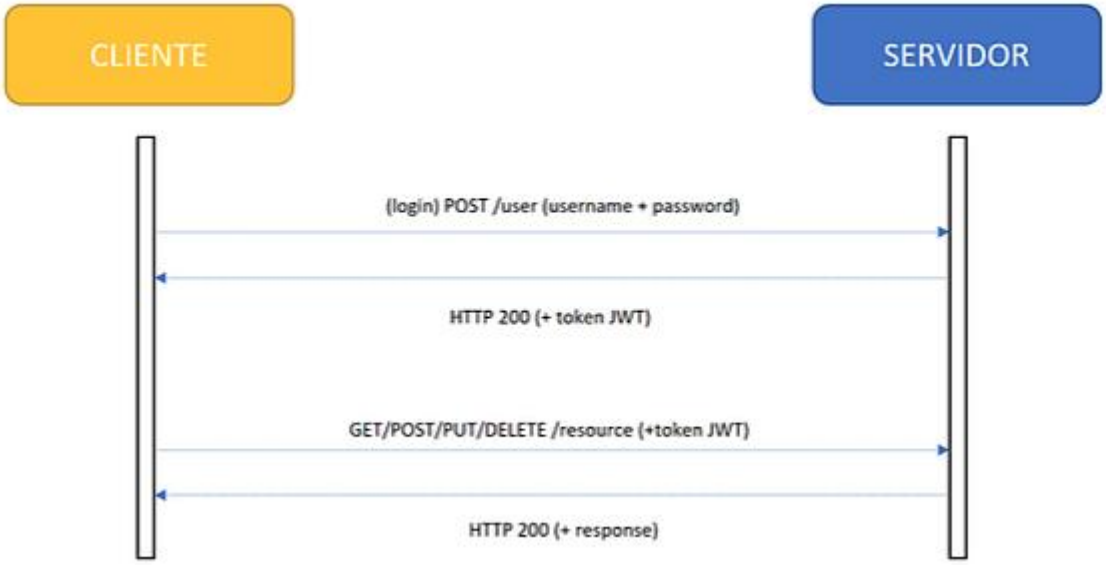
PASOS EN EL CASO DEL ENFOQUE SESIÓN-COOKIE:

1. Obtenga el "ID de sesión" de la solicitud "cookie".
2. Verifique la integridad del "ID de sesión" utilizando la "clave secreta". Luego busque el "ID de sesión" dentro de la base de datos de sesión en el servidor y obtenga el "ID de usuario".
3. Busque el "ID de usuario" dentro de la base de datos del carrito de compras para agregar artículos al carrito para ese ID de usuario, o muestre la información del carrito para ese ID de usuario.



PASOS EN EL CASO DEL ENFOQUE JWT:

1. Obtenga el "token de acceso" del "encabezado" de la solicitud.
2. Descifre el "token de acceso", es decir, el JWT, usando `ACCESS_TOKEN_SECRET` y obtenga el "ID de usuario" ---> (no hay búsqueda en la base de datos).
3. Busque el "ID de usuario" dentro de la base de datos del carrito de compras para agregar artículos al carrito para ese ID de usuario, o muestre la información del carrito para ese ID de usuario.



Es importante recalcar que en el caso del enfoque de cookies de sesión, el servidor necesitaría mantener una lista de todas las sesiones en una base de datos de sesiones. También tenga en cuenta que esto significa que habría una búsqueda en la base de datos para obtener el ID de usuario. Las búsquedas de DB son relativamente más lentas que la acción de descifrado de JWT.

En el caso del enfoque JWT, dado que el token JWT en sí contiene la información del ID de usuario (encriptada), no necesitamos realizar una "búsqueda" para obtener el ID de usuario. En el momento en que el servidor descifra el token JWT, obtendrá el ID de usuario.