ACERCA DE ALEX SOTO

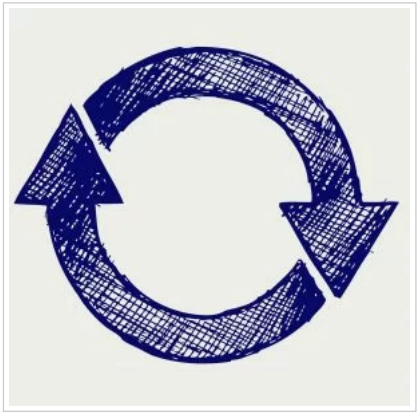
Rotación secreta para tokens JWT

Publicado por: Alex Soto en Enterprise Java 9 de enero de 2018

Cuando usa **JSON Web Token** (**JWT**) o cualquier otra tecnología de token que requiere firmar o encriptar información de carga útil, es importante establecer una fecha de vencimiento para el token, por lo que si el token expira, puede suponer que esto podría se considerará una infracción de seguridad y usted rechazará cualquier comunicación que utilice este token, o si decide habilitar el token actualizándolo con una nueva fecha de caducidad.

Pero también es importante utilizar algún tipo de algoritmo de rotación secreta, por lo que el secreto utilizado para firmar o encriptar un token se actualiza periódicamente, por lo que si el secreto se ve comprometido, los tokens filtrados por esta clave son menores. También de esta manera estás disminuyendo la probabilidad de que se rompa un secreto.

Hay varias estrategias para implementar esto, pero en este post, voy a explicar cómo implementé la rotación secreta en un proyecto que desarrollé hace algunos años para firmar tokens **JWT** con el algoritmo HMAC.



Voy a mostrar cómo crear un token **JWT** en Java.

```
try {

    Algorithm algorithm = Algorithm.HMAC256("secret");
    String token = JWT.create()
        .withIssuer("auth0")
        .sign(algorithm);

} catch (UnsupportedEncodingException exception){
    //UTF-8 encoding not supported
} catch (JWTCreationException exception){
    //Invalid Signing configuration / Couldn't convert Claims.
}
```

Tenga en cuenta que lo que debe hacer aquí es crear un algoritmo de configuración de objeto de algoritmo HMAC y establecer un secreto que se utiliza para firmar y verificar la instancia.

Entonces, lo que necesitamos es rotar esta instancia de algoritmo cada X minutos, por lo que la probabilidad de romper el secreto, y que el secreto roto sigue siendo válido, se vuelve muy bajo.

Entonces para generar el secreto, necesitas una cadena, en el ejemplo anterior era *Secret* String, por supuesto, esto no es tan seguro, entonces la idea es componer esta cadena secreta por una raíz (algo que llamamos la parte big bang) + un tiempo parcial cambiado En resumen, el secreto es *<bigbang> + <timeInMilliseconds>*

La parte de Bing bang no tiene ningún misterio, es solo una parte estática, por ejemplo, *my_super_secret* .

La parte interesante es la parte del tiempo. Supongamos que quiere renovar el secreto cada segundo. Solo necesitas hacer esto:

```
long t = System.currentTimeMillis();

System.out.println(t);
System.out.println((t/1000)*1000);

TimeUnit.MILLISECONDS.sleep(50);

t = System.currentTimeMillis();
System.out.println((t/1000)*1000);
```

Solo pongo la parte de 0 a milisegundos, así que si ejecuto esto obtengo algo así como:

```
<i>1515091335543</i>
<i>1515091335500</i>
<i>1515091335500</i>
```

Tenga en cuenta que, aunque entre la segunda y la tercera impresión, ha pasado 50 milisegundos, la parte de tiempo es exactamente la misma. Y será lo mismo durante el mismo segundo.

Por supuesto, este es un ejemplo extremo en el que el secreto se cambia cada segundo, pero la idea es eliminar la parte del tiempo que desea ignorar y llenarlo con ceros. Por esta razón, primero, divide el tiempo y luego lo multiplica por el mismo número.

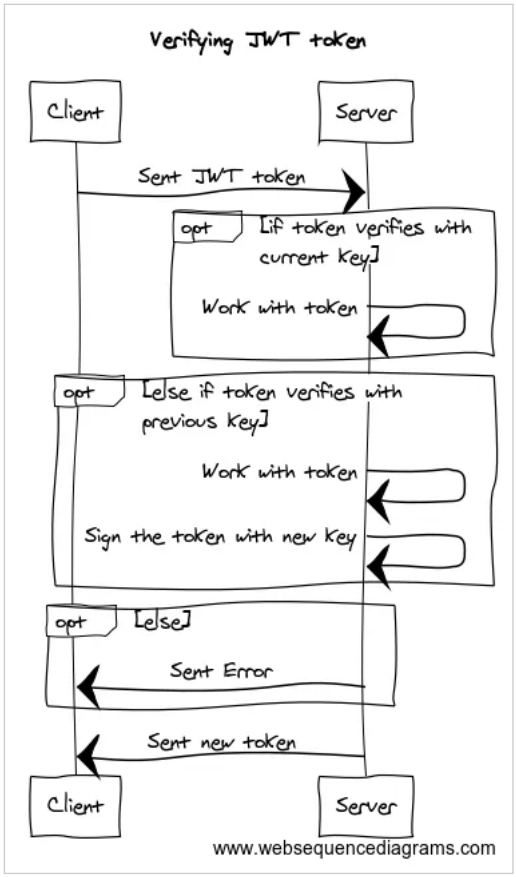
Por ejemplo, supongamos que desea rotar el secreto cada 10 minutos, solo necesita dividirlo y multiplicarlo por 600000.

Hay dos problemas con este enfoque que pueden corregirse, aunque uno de ellos no es realmente un gran problema.

La primera es que, dado que estás truncando la hora si deseas cambiar el secreto cada minuto y, por ejemplo, el primer cálculo ocurre a la mitad de un minuto, entonces solo para este caso inicial, la rotación ocurrirá después de 30 segundos y no 1 minuto No es un gran problema y en nuestro proyecto no hicimos nada para solucionarlo.

El segundo es lo que está sucediendo con los tokens que se firmaron justo antes de la rotación secreta, todavía son válidos y debes poder verificarlos también, no con el nuevo secreto, sino con el anterior.

Para solucionar esto, lo que hicimos fue crear una ventana válida, donde también se mantuvo el secreto válido anterior. Entonces, cuando el sistema recibe un token, se verifica con el secreto actual; si pasa, podemos hacer otras comprobaciones y trabajar con él, de lo contrario, el token se verifica con el secreto anterior. Si pasa, el token se vuelve a crear y se firma con el nuevo secreto, y si no, obviamente este token no es válido y debe ser rechazado.



Para crear el objeto de algoritmo para **JWT** , solo tiene que hacer algo como:

```
long currentTime = System.currentTimeMillis();

```

```
7 }
```

Lo que realmente me gusta de esta solución es:

- Está limpio, no necesita elementos adicionales en su sistema.
- No es necesario que los hilos activados se ejecuten de forma asíncrona para actualizar el secreto.
- Es realmente eficaz, no necesita acceder a un sistema externo.
- Probar el servicio es realmente fácil.
- El proceso de verificación es responsable de rotar el secreto.
- Es realmente fácil de escalar, de hecho, no necesita hacer nada, puede agregar más y más instancias del mismo servicio y todas rotarán el secreto al mismo tiempo, y todas usarán el mismo secreto, por lo que el proceso de rotación es realmente sin estado, puede ampliar o reducir sus instancias y todas las instancias continuarán pudiendo verificar las señales firmadas por otras instancias.

Pero, por supuesto, hay algunos inconvenientes:

- Aún necesita compartir una parte del secreto (la parte del big bang) con cada uno de los servicios de forma segura. Tal vez usando secretos de Kubernetes, Vault de Hashicorp o si no está usando microservicios, puede simplemente copiar un archivo en una ubicación concreta y cuando el servicio esté en funcionamiento, lea la parte Big Bang, y luego simplemente elimínelo.
- Si sus servidores físicos están en diferentes zonas horarias, entonces usar este enfoque podría ser más problemático. Además, necesita que los servidores estén más o menos sincronizados. Como está almacenando el token anterior y el token actual, no es necesario que estén sincronizados en el mismo segundo y todavía es posible un retraso de algunos segundos sin ningún problema.

Así que hemos visto una manera muy simple de rotar secretos para que pueda mantener sus tokens más seguros. Por supuesto, hay otras formas de hacer lo mismo. En este post, acabo de explicar cómo lo hice en una aplicación monolítica que desarrollamos hace tres años, y funcionó muy bien.

Seguimos aprendiendo,

Alex

Publicado en Java Code Geeks con permiso de Alex Soto, socio de nuestro programa JCG . Vea el artículo original aquí: Rotación secreta para tokens JWT

Las opi

Etiquetado con: JWT

1 Comentario sobre "Rotación secreta para tokens JWT"

Notificar de nuevos comentarios de seguimiento Email >

Join the discussion

Ordenar por: más nuevos|más antiguo|el más votado

John Doe

Huésped

1) Desde mi punto de vista, la rotación secreta es importante cuando el secreto de alguna manera se escapa para acortar el período en que su sistema está en peligro. ¿Cómo ayuda este enfoque?

2) Independientemente de la primera pregunta: el token generalmente contiene la marca de tiempo cuando se emitió. ¿No es realmente mejor usar esto para verificar el token en lugar de probar el número arbitrario (2 en su caso) de tokens?

0 RESPUESTA Hace 1 día 19 horas

BASE DE CONOCIMIENTOS

Cursos
Ejemplos
Minilibros
Recursos
Tutoriales

FOGONADURA

Mkyong

LA RED CODE GEEKS

.NET Code Geeks
Java Code Geeks
Geeks del Código del Sistema
Geeks de código web

SALÓN DE LA FAMA

Serie "Tutorial de aplicación completa de Android"
11 sitios web de aprendizaje en línea que debes visitar
Ventajas y desventajas de la computación en la nube - Pros y contras de computación en la nube
Tutorial de Android Google Maps
Android JSON Parsing with Gson Tutorial
Android Location Based Services Application – GPS location
Android Quick Preferences Tutorial
Difference between Comparator and Comparable in Java
GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial
Java Best Practices – Vector vs ArrayList vs HashSet

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused o ultimate Java to Java developers resource center; targeted at the techn technical team lead (senior developer), project manager and junior dev JCGs serve the Java, SOA, Agile and Telecom communities with daily n domain experts, articles, tutorials, reviews, announcements, code snipp source projects.

DISCLAIMER

Todas las marcas comerciales y marcas registradas que aparecen en Ja son propiedad de sus respectivos dueños. Java es una marca comercial registrada de Oracle Corporation en los Estados Unidos y otros países. Code Geeks no está conectado a Oracle Corporation y no está patrocina Corporation.