Capítulo 9: Seguridad

Víctor Iranzo

30 de mayo de 2018

1. Autenticación y autorización

La autenticación es el proceso por el cual confirmamos que una entidad es quien dice ser. Generalmente, un usuario se autentica mediante un nombre de usuario y una contraseña.

La autorización es el mecanismo por el cual relacionamos una entidad autenticada con las acciones que puede llevar a cabo.

En las aplicaciones basadas en microservicios hace falta definir un mecanismo para que los usuarios no tengan que autenticarse en cada uno de los microservicios de forma separada.

1.1. Proveedor de identidades

Una aproximación para ofrecer estas características son las soluciones single sing-on (SSO). Entre ellas podemos mencionar SAML y OpenID Connect. SAML es un estándar basado en SOAP con el que es bastante complejo trabajar. OpenID Connect es un estándar que surge como una implementación específica de OAuth 2.0 que imita la manera en que organizaciones como Google gestionan sus soluciones SSO. Por su facilidad de uso, se espera que OpenID Connect sea en un futuro acogido por cada vez más organizaciones.

Cuando una entidad trata de acceder a un recurso o realizar una operación, es redirigido a un proveedor de identidades para que se autentique. Una vez autenticados, el proveedor de identidades responderá al servicio que lo ha invocado indicando los permisos que tiene la entidad sobre la operación o recurso solicitados.

El proveedor de identidades puede ser un servicio externo, como el de Google. Sin embargo, la mayoría de empresas emplear su propio servicio

de directorio (similar a Active Directory) donde se almacena información sobre las entidades conocidas, sus roles y permisos. Los permisos efectivos para un microservicio no se pueden almacenar de forma centralizada ya que esta información pertenece al servicio en si y puede suponer un punto de acoplamiento. Además, los roles que se modelan deben ser lo más similares a los existentes en el mundo real donde se ejecuta el sistema.

Para centralizar el contacto entre los servicios del sistema y el proveedor de identidades se puede emplear un SSO gateway. Este mecanismo consiste en un proxy situado entre los servicios y el mundo exterior, lo que puede suponer un único punto de fallo. Además, al ser una capa intermedia puede tender a ir aumentando progresivamente en las funcionalidades que ofrece hasta convertirse en un punto de acoplamiento.

2. Seguridad en tránsito

2.1. HTTP y HTTPS

No sólo se deben autenticar los usuarios para realizar una acción, otros servicios también se pueden modelar para saber si se les da acceso a un recurso o funcionalidad.

Una posible opción es hacer que cualquier invocación a un servicio hecha desde el propio servicio se considera autorizada. Esta solución puede ser peligrosa ante atacantes que penetren en la red, pero puede ser adecuada si los datos almacenados no son de alta sensibilidad.

La autorización HTTP permite a un cliente enviar un nombre de usuario y una contraseña en la cabecera del mensaje. Aunque es un protocolo muy extendido, puede ser peligroso porque estos datos son enviados sin ninguna seguridad. Por este motivo se suele emplear el protocolo de HTTPS.

Entre los problemas de usar HTTPS están que el tráfico en este protocolo no puede ser capturado por proxies inversos, aunque en caso de necesidad se puede capturar en alguno de los extremos o en un balanceador de carga. Además, se deben gestionar los certificados SSL, que puede resultar un problema cuando el sistema se distribuye entre múltiples máquinas, y se puede aumentar el tiempo de entrega de los mensajes.

2.2. Certificados de clientes

Con esta aproximación cada cliente emplea un certificado en su interacción con el servidor. Esta solución es aconsejable si la sensibilidad de los datos que se envían es muy alta. Sin embargo, puede traer dificultades a la hora de revocar y emitir certificados.

2.3. Códigos de autentificación de mensajes en clave-hash (HMAC)

El tráfico a través de HTTPS puede afectar al rendimiento del sistema. Una solución a este problema pasa por, en lugar de usar este protocolo, usar mensajes cifrados con claves hash y enviarlos a través de HTTP.

Con HMAC, el cuerpo de una petición es convertido en un hash empleado una llave privada y es enviado junto con la petición. El servidor, cuando recibe una petición, usa la misma llave privada que comparten para generar el hash con el cuerpo del mensaje recibido y compararlo con el hash del mensaje.

Este mecanismo cuenta con 3 desventajas. Primero, entre servidor y cliente tiene que existir un secreto compartido, que debe ser enviada usando un protocolo seguro. Segundo, este mecanismo no es un estándar y existen muchas implementaciones distintas disponibles que deben ser evaluadas. Tercero, un atacante pueda interceptar y leer una petición, pero no puede modificar esta porque si lo hace el hash que genera el servidor y el de la petición no coincidirían.

2.4. Claves API

Muchas APIs públicas emplean este mecanismo para ofrecer sus servicios a terceros. Se pueden emplear para identificar quién hace una llamada y limitar las que puede hacer para que no sobrecargue el servicio. Entre sus ventajas podemos nombrar su facilidad de uso.

2.5. Problema del reemplazo

Una entidad autenticada puede invocar a un servicio que a su vez tenga que invocar a un tercero. En esta segunda petición, si aceptamos la aproximación de que cualquier petición hecha desde dentro del sistema es válida, podemos incurrir en un problema de reemplazo de identidades.

Por ejemplo, un atacante que haya conseguido acceso a la red puede obligar a un servicio a hacer peticiones a otro y obtener los datos del servicio atacado. Este problema es grave de acuerdo a la sensibilidad de los datos. Una posible solución es solicitar que en cada petición sean provistas las credenciales de la entidad que comienza la interacción.

3. Seguridad en los datos

La encriptación de los datos almacenados es necesaria en función de su sensibilidad. Existen muchos algoritmos para este propósito en los principales lenguajes de programación y que han sido ampliamente probados. No obstante, hay que estar siempre actualizado ante parches que solucionen vulnerabilidades detectadas. No hacerlo, al igual que tratar de implementar un algoritmo de encriptación propio, puede darnos una falsa sensación de seguridad que en cierto punto puede llegar a ser incluso más peligroso que no tener seguridad alguna.

Algunas bases de datos como SQL Server ofrecen la encriptación de datos de forma transparente. En otras tecnologías hará falta una llave explícita que se debe almacenar en un lugar seguro o de otra forma el atacante solo necesitaría encontrar la llave para acceder a los datos.

En un sistema basado en microservicios, puede ser una buena idea sólo encriptar las bases de datos o tablas de los servicios más sensibles. La encriptación debe realizarse la primera vez que se ven los datos y sólo debe almacenarse el resultado en un único sitio. La desencriptación debe realizarse bajo demanda para entidades reconocidas. La encriptación en las copias de seguridad también es fundamental.

4. Seguridad en profundidad: otras medidas de protección

La seguridad en profundidad es clave para interponer más barreras entre nuestro sistema y un atacante. Se pueden tomar medidas en diferentes niveles:

- Cortafuegos: puede consistir simplemente en limitar el acceso a cierto tipo de tráfico o de puertos. Tener varios cortafuegos puede ser interesante para diferenciar perímetros de seguridad.

- Logging: no es una medida de prevención, pero puede resultar útil para detectar y recuperarse de un ataque. No se debe almacenar información sensible en los logs.
- Sistemas de detección y prevención de intrusos (IDS e IPS): los primeros, se emplean para monitorizar las redes y hosts en busca de comportamientos sospechosos. Los sistemas IPS van un paso más allá y permiten detener actividades sospechosas.
- Segregación de redes: los servicios se pueden situar en diferentes segmentos de la red para controlar con cuáles puede hablar cada uno. Cada perímetro de la red puede pertenecer a un equipo y puede contar con diferentes medidas de seguridad. AWS ofrece la posibilidad de crear nubes virtuales privadas (VPC) para separar los hosts en subredes y controlar cuáles son accesibles por otras.
- Sistemas operativos: los servicios deben ejecutarse con un usuario con el menor número de permisos necesarios en el SO. Es importante aplicar los parches de seguridad que se publican ya que la mayoría de ataques se hacen a través de vulnerabilidades para las que existía un parche. Portales como Microsoft SCCM pueden mantenernos informados sobre esto. En Linux, se pueden aplicar medidas de seguridad que se controlan a nivel del kernel con aplicaciones como AppArmor.

5. Caso de estudio

En un sistema basado en microservicis, necesidades diferentes de seguridad pueden suponer medidas de seguridad diferentes en cada uno.

Para las aplicaciones clientes, como un navegador web, se puede emplear tráfico HTTPS en todas las páginas accesibles una vez el usuario se ha autenticado.

Para integrar con otros servicios, como un sistema de pagos, se pueden emplear certificados de cliente para garantizar la legitimidad de los mensajes intercambiados.

Para las APIs que nuestro sistema ofrece al público general, como hay interés en que sea fácil de usar y se emplee lo máximo posible, se pueden emplear claves API para identificar quien realiza cada petición e incluso limitar el número de peticiones que puede hacer.

Si contamos con servicios internos que son solo accesibles por nuestros servicios, se pueden emplear subredes para aislarlos del exterior y firewalls

para impedir tráfico no permitido.

En caso de que un servicio almacene datos sensibles, se pueden encriptar sus tablas, pero no las de servicios triviales.

6. Otros factores de la seguridad