SEGURIDAD EN ENTORNOS JEE

TEMARIO

- DEFINICIONES
- AUTENTICACIÓN Y AUTORIZACIÓN
- CIFRADOS Y FIRMA DIGITAL
- HTTPS
- PGP MAILVELOPE
- CORS XSS CSRF SQLInjection

TEMARIO

- MECANISMOS ESTÁNDAR JEE
- ESAPI
- AUTH0
- JWT JsonWebToken

DEFINICIONES

AUTENTICAR ó autentificar : acreditar que un usuario es quien dice ser, generalmente por medio la validación usuario + contraseña en un servidor

DEFINICIONES

AUTORIZAR : acreditar que un usuario tiene la potestad de invocar o emplear determinados servicios dado su rol o privilegios en un sistema

DEFINICIONES

CODIFICAR : es la tarea por la cual un mensaje se traduce a otro formato o representación por cuestiones de economía o conveniencia. La información no se oculta y el algoritmo entre las dos representaciones es público (clave conocida)

DEFINICIONES

CIFRAR ó encriptar : es la tarea por la cual un mensaje se traduce a otro formato o representación por cuestiones de seguridad. La información se oculta y el algoritmo entre las dos representaciones no es público (clave desconocida)

DEFINICIONES

CONFIDENCIALIDAD : es la propiedad de un mensaje que procura que su contenido sea únicamente conocido por el emisor y el legítimo receptor del mensaje

DEFINICIONES

VERIFICACIÓN : comprobación de que un mensaje ha sido enviado por quien dice ser su remitente y su contenido no ha sido alterado

JAAS REALM

JAAS es una especificación Java que define una serie de interfaces y clases estándar para producir la autenticación y autorización de los usuarios

Por otra parte, REALM, es un mecanismo que incorpora TOMCAT que, usando una implementación JAAS, nos permite definir distintos lugares donde almacenar las credenciales

JAAS REALM JDBC

1 Crear tablas en la base de datos con usuarios y permisos (roles)

```
create table users (
  user_name        varchar(15) not null primary key,
  user_pass        varchar(15) not null
);
```

JAAS REALM JDBC

```
create table user_roles (
  user_name         varchar(15) not null,
  role_name         varchar(15) not null,
  primary key (user_name, role_name)
);
```

JAAS REALM JDBC

2 Configurar el contexto

META-INF/context.xml

```
<?xml version='1.0' encoding='utf-8'?>
<Context docBase="WebMicroForum" path="/WebMicroForum" reloadable="true">
<Realm className="org.apache.catalina.realm.JDBCRealm"
connectionName="hr" connectionPassword="password"
connectionURL="jdbc:oracle:thin:@localhost:1521:xe"
driverName="oracle.jdbc.driver.OracleDriver" roleNameCol="ROLE_NAME"
userCredCol="USER_PASS" userNameCol="USER_NAME"
userRoleTable="USER_ROLES"
userTable="USERS"/>
</Context>
```

JAAS REALM JDBC

3 Definir restricción seg en web.xml

```
<security-constraint>
    <web-resource-collection>
      <web-resource-name>Entire Application</web-resource-name>
      <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>admin</role-name>
    </auth-constraint>
  </security-constraint>
```

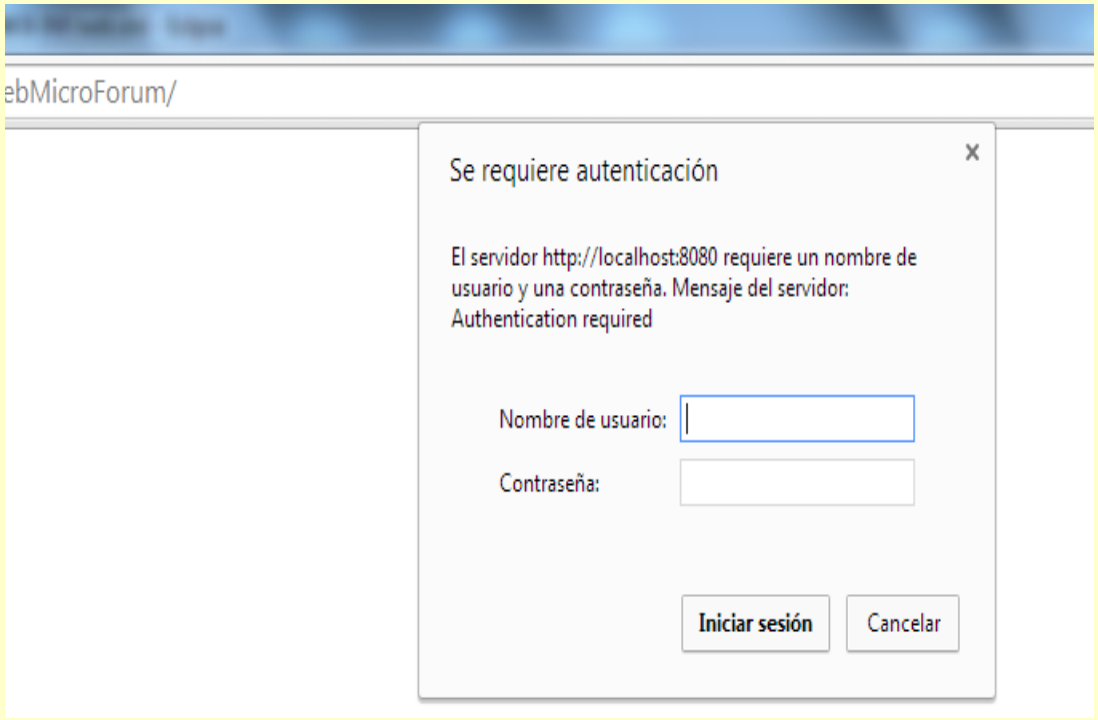
JAAS REALM JDBC

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Administration Area</realm-name>
  <form-login-config>
    <form-login-page>/login.jsp</form-login-page>
    <form-error-page>/login-failed.jsp</form-error-page>
  </form-login-config>
</login-config>
```

JAAS REALM JDBC

IMPORTANTE: YA QUE LA MISIÓN DE VALIDACIÓN RECAE SOBRE TOMCAT Y ÉSTE DEBE ACCEDER A LA BASE DE DATOS, HAY QUE AÑADIR LA BIBLIOTECA DEL DRIVER AL DIRECTORIO DE INSTALACIÓN DE TOMCAT/LIB

JAAS REALM JDBC



JAAS REALM JDBC

Una vez que nuestro usuario ha accedido con éxito, podemos acceder a su nombre a través del objeto Request

request.getUserPrincipal().getName()

NULL si el usuario no está autenticado

El nombre, en caso de que sí lo esté

INTRODUCCIÓN

SPRING Security es un framework que facilita la autenticación y la autorización en entornos Java, dando además respuesta a ataques y vulnerabilidades comunes y permitiendo la integración en entornos distribuidos (MVC)

INTRODUCCIÓN

La versión estable (GA) es la 4.2.3

DOCUMENTACIÓN WEB

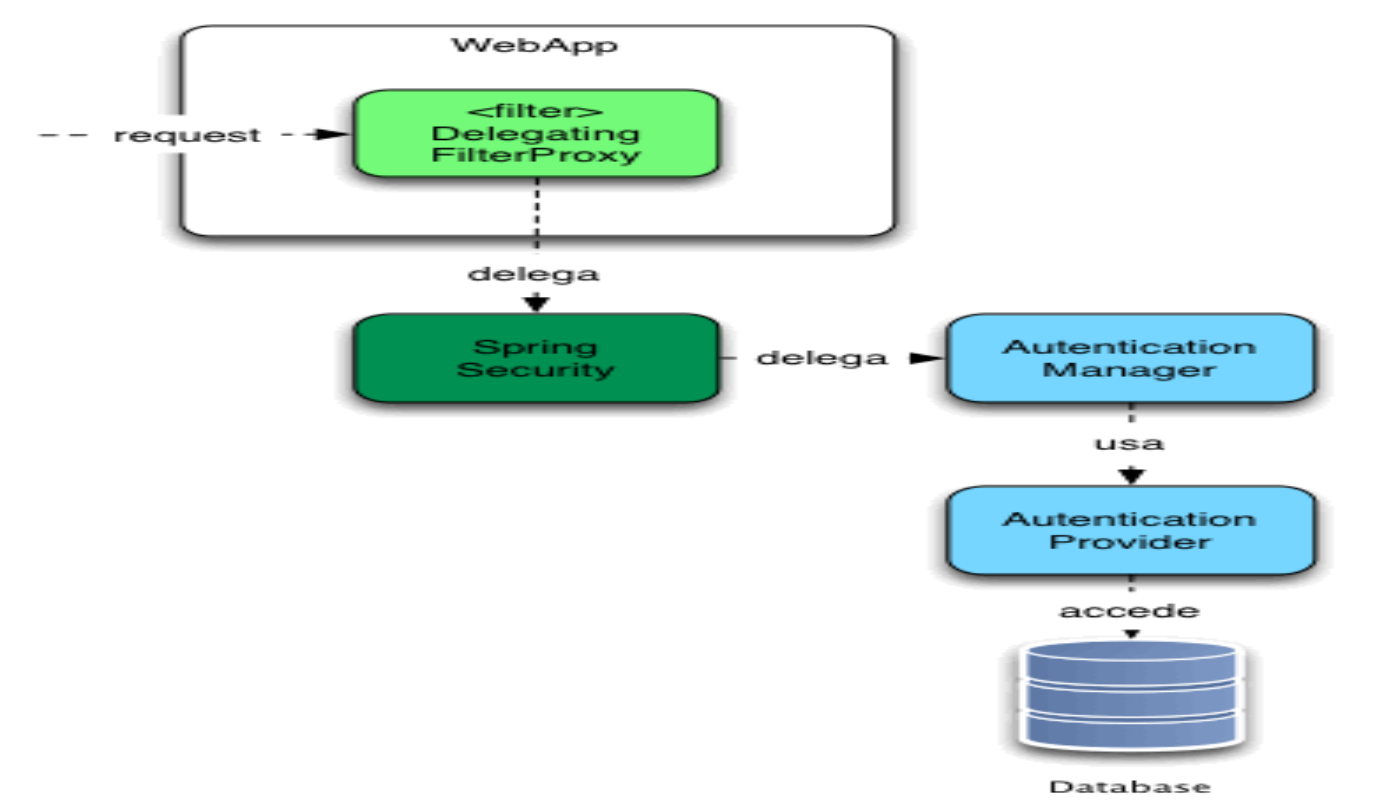
LIBRERÍAS MAVEN

XML vs JConfig

La configuración de una app Spring así como la iniciaización del contexto y dependencias, puede hacerse por medio de anotaciones y clases de configuración o por medio de xml auxiliares, detallados a parte.

Nosotros, usaremos XML

Tablas estándar



Tablas estándar

```
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(50) not null,
    enabled boolean not null);

create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fk_authorities_users foreign key(username) references
 users(username));
    create unique index ix_auth_username on authorities (username,authority);
```

AUTENTICACIÓN SS

```
<authentication-manager>
<authentication-provider>
<user-service>
<user name="user" password="password"
   authorities="ROLE_USER" /> <!--prefijo-->
<user name="admin" password="password"
   authorities="ROLE_USER,ROLE_ADMIN"/>
</user-service>
</authentication-provider>
```

Tablas propias

```
<authentication-provider>
<jdbc-user-service data-source-ref="midatasource"
authorities-by-username-query="select user_name,
  role_name from user_roles where user_name = ?"
users-by-username-query="select user_name,
  user_pass, 1 from users where user_name = ?"/>
</authentication-provider>
```

CIFRADOS

Criptografía de clave simétrica

Método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes en el emisor y el receptor

DES 3DES BLOWFISH IDEA

CIFRADOS

Criptografía de clave asimétrica

Método criptográfico en el cual emisor y receptor tiene un par de claves, pública y privada. La privada para firmar los mensajes emitidos y la pública para que puedan ser verificados por el receptor

CIFRADOS

Criptografía de clave asimétrica

SI A envía un mensaje a B y emplea su clave privada; B, puede verificar que A es el remitente

SI A envía a B un mensaje con la clave pública de B sólo B podrá descifrarlo con su clave privada

CIFRADOS

Criptografía de clave asimétrica

Algoritmos

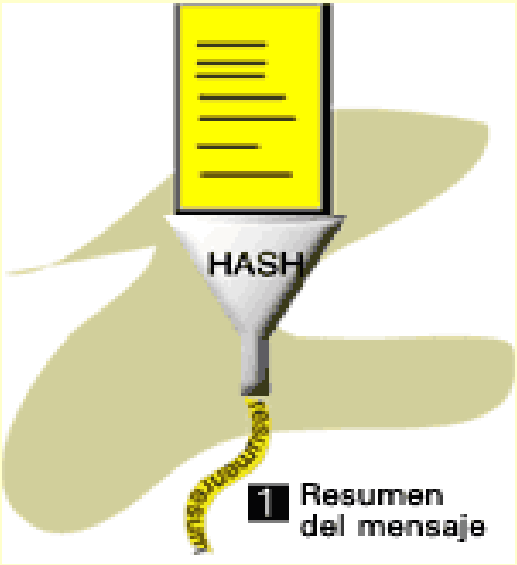
RSA DSA Diffie-Hellman ElGamal

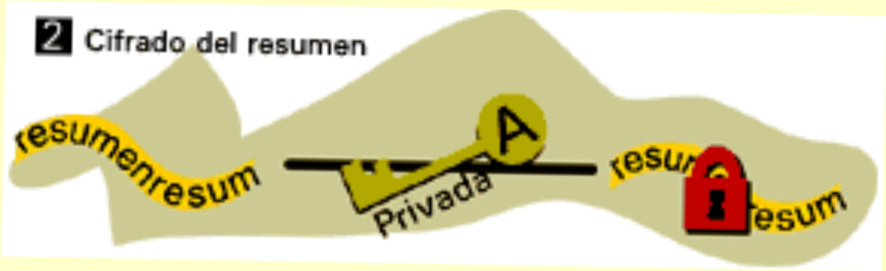
FIRMA DIGITAL

LA FIRMA DIGITAL es ...

Un mensaje que acompaña a la información que quiere transmitirse y sirve para verificar la autenticidad e integridad del mensaje

FIRMA DIGITAL





FIRMA DIGITAL

CERTIFICADOS HTTPS

HTTPS funciona por 433

SSL / TLS

NAVEGADOR certificados de confianza

Servidor firma con la clave privada

Navegador firma con la clave pública

AC hay muchas

Emiten y avalan

SSL en TOMCAT

En tres sencillos pasos puedo hacer que TOMCAT cifre sus comunicaciones con los Clientes

Básicamente debo generar una clave y proporcionársela en su configuración

SSL en TOMCAT

1 Generar clave RSA

cd %JAVA_HOME%/bin

keytool -genkey -alias tomcat -keyalg RSA

El final del proceso debe conllevar a la generación de un archivo oculto .keystore en la carpeta del usuario

SSL en TOMCAT

2 Modificar el server.xml

Localizar y decomentar el elemento

```
<!--
<Connector port="8443" protocol="HTTP/1.1"
   SSLEnabled="true" maxThreads="150"
   scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS" />
-->
```

SSL en TOMCAT

```
<Connector SSLEnabled="true" acceptCount="100"
   clientAuth="false" disableUploadTimeout="true"
   enableLookups="false" keystoreFile="/user/.keystore”
   keystorePass="miclave" maxThreads="25" port="8443"
   protocol="org.apache.coyote.http11.Http11NioProtocol"
   scheme="https" secure="true" sslProtocol="TLS"/>
```

SSL en TOMCAT

3 Testar

Nuestra app ya debería atender llamadas en el puerto 8443

https://localhost:8443/MyApp/index.html

SSL en TOMCAT

4 Forzar HTTPS para todas las peticiones

Añadiendo una mínima configuración en el descriptor de la aplicación (/WEB-INF/web.xml) puedo forzar a que todas las peticiones no cifradas sean redirigidas al puerto seguro

SSL en TOMCAT

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>securedapp</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

PGP

Pretty Good Privacy (privacidad bastante buena), es una herramienta GNU que, basada en el cifrado asimétrico, permite el intercambio de mensajes confidenciales con un alto grado de eficacia

PGP

Periodistas y otros profesionales de la información o nosotros mismos, Podemos emplear fácilmente la herramienta para comunicarnos de forma segura

PGP - COMANDOS

[usuario@equipo ~]$ gpg –gen-key

//el email será clave

[usuario@equipo ~]$ gpg --export --armor
correo@delaclave.com > miclave-publica.asc

//exportar la clave pública, para compartirla

[usuario@equipo ~]$ gpg --export-secret-key --armor
correo@delaclave.com > miclave-privada.asc

PGP - COMANDOS

[usuario@equipo ~]$ gpg --encrypt --recipient correo@delaclave.com archivo.txt

//cifrar

[usuario@equipo ~]$ gpg -d archivo.txt.gpg > archivo.txt

// descifrar contando con la clave privada

MAILVELOPE

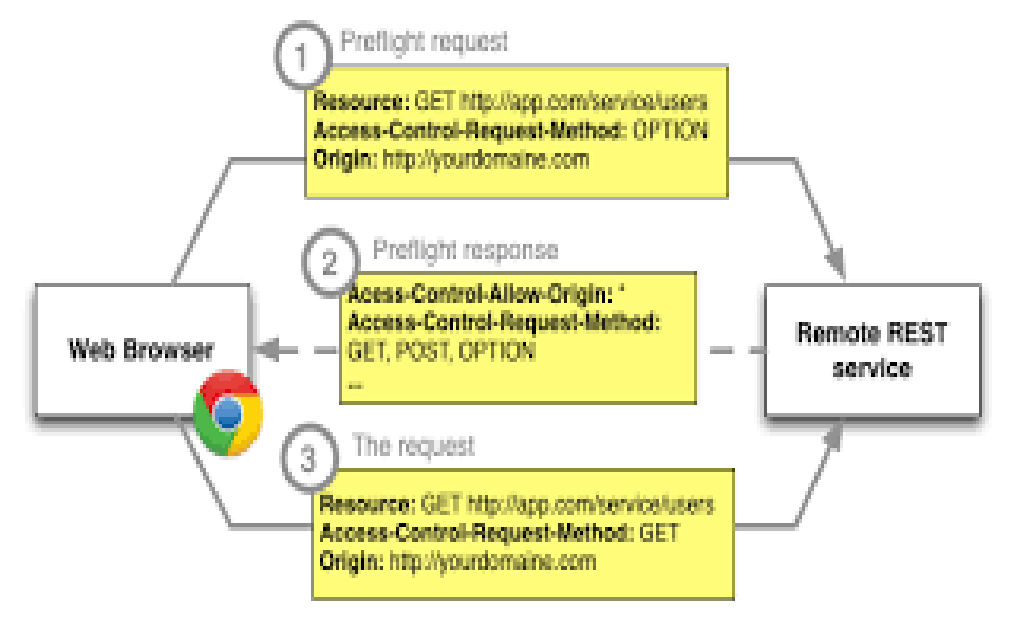
Es un plugin para Chrome y Firefox que Permite gestionar el envío de correos con GMAIL usando las claves privadas y públicas para cifrar el de PGP, y poder así cifrar y firmar correos

ATAQUES CORS

Los ataques Cross-origin Resource Sharing o CORS consisten en realizar peticiones por JS a un dominio distinto del que se realizó la petición original

Las peticiones realizadas por XMLHttpRequest son en principio denegadas pot defecto, según recomendación del W3C

ATAQUES CORS



ATAQUES CORS

Para configurar una respuesta apropiada desde el servidor que ofrece sus servicios a terceros, podemos definir:

Filtros de servidor (SERVIDOR- TOMCAT)
O propios (APP-SPRING)

ATAQUES XSS

Los ataques Cross-site Scripting o XSS consisten en inyectar código JS en formularios o URLs donde no se esperan, haciendo que las instrucciones viajen y se ejecuten en otro cliente pudiendo obtener información de un tercero o provocarle comportamientos no deseados

ATAQUES XSS

EJ. FORMULARIO COMENTARIOS

```
<SCRIPT type="text/javascript">
var adr = '../evil.php?cakemonster=' +
   escape(document.cookie);
</SCRIPT>
```

ATAQUES XSS

EJ. FORMULARIO COMENTARIOS

```
<SCRIPT type="text/javascript">
var adr = '../evil.php?cakemonster=' +
   escape(document.cookie);
</SCRIPT>
```

ATAQUES CSRF

Se trata de ejecutar peticiones a un servidor en el que el usuario está logueado sin el consentimiento de éste

ATAQUES CSRF

DESCARGAS UNA PÁGINA QUE CONTIENE COMANDOS DEL ESTILO.

```
<img src="http://bank.com/transfer.do?
   acct=MARIA&amount=100000" width="0"
   height="0" border="0">
<a href="http://bank.com/transfer.do?
   acct=MARIA&amount=100000">View my
   Pictures!</a>
```

ATAQUES CSRF SS

PARA EVITAR EL ENVÍO DE PETICIONES QUE MODIFICAN EL ESTADO DEL SERVIDOR, ESTAS DEBEN ACOMPAÑARSEDE UN CAMPO OCULTO llamado csrf (distinto al de la sesión)

ATAQUES CSRF SS

<input type="hidden"
name="${_csrf.parameterName}"
value="${_csrf.token}"/>)

ATAQUES CSRF SS

Si el código csrf no coincide con el esperado, se produce un error. Así se controla que la petición enviada sea legítima. Para desactivar esta opción por defecto, hace falta incluir el elemento

<csrf disabled="true"/>

ATAQUES SQL Injection

Con este tipo de ataques, introduzco sentencias SQL nuevas o que modifican las que ya existen

ATAQUES SQL Injection

SELECT * FROM items

WHERE owner =

AND itemname = ;

'name' OR 'a'='a' //introduzco

SELECT * FROM items //conseguido

PreparedStatement

Para evitar el uso de SQL Injections en nuestro sistema, un método rápido y seguro es usar el tipo de dato PreparedStatement en vez de armar uno mismo la sentencia con los parámetros

PreparedStatement

```
String updateString =
    "update " + dbName + ".COFFEES " +
    "set SALES = ? where COF_NAME = ?";
UpdateSales =
con.prepareStatement(updateString);
updateSales.setInt(1, 100);
updateSales.setString(2, "French_Roast");
updateSales.executeUpdate();
```

Otras alternativas SLQi

STORED PROCEDURES
WHITE LIST
ESCAPAR
BUEN DISEÑO
USO FRAMEWORKS

MANEJO DE ERRORES

Lo mejor, es que los posibles errores o excepciones de un servicio se registren internamente, pero al usuario se le muestre un mensaje genérico, ocultando así las posibles pistas sobre los fallos que se han provocado el sistema

MANEJO DE ERRORES

```
<error-page>
    <error-code>404</error-code>
    <location>/ServletError</location>
</error-page>
<error-page>
    <exception-type>javax.servlet.ServletException
    </exception-type >
    <location>/ ServletError </location>
</error-page>
```

Atributos

En el objeto Request que recibe el Servlet que gestiona los errores, recibimos varios atributos, que ayudan a identificar el error

Tipo de expceción - javax.servlet.error.exception

Código HTTP - javax.servlet.error.status_code

Servlet fallido -javax.servlet.error.servlet_name

URL invocada -javax.servlet.error.request_uri

Atributos

Debo hacer request.getAttribute para cada valor, y hacer el casting al tipo de objeto apropiado

Throwable - javax.servlet.error.exception

Integer - javax.servlet.error.status_code

String -javax.servlet.error.servlet_name

String -javax.servlet.error.request_uri

Obteniendo Atributos

```
Throwable excepción = (Throwable)
 request.getAttribute("javax.servlet.error.exception");
 Integer codigoHTTP = (Integer)
 request.getAttribute("javax.servlet.error.status_code");
 String nombreServlet = (String)
 request.getAttribute("javax.servlet.error.servlet_name");
 if (null == nombreServlet ){nombreServlet =
 “Desconocido"; } String uriPedida = (String)
 request.getAttribute("javax.servlet.error.request_uri"); if
 (null== uriPedida ){uriPedida = “Desconocida"; }
```

MANEJO DE ERRORES

Con la anotación @ControllerAdvice puedo indicar en un ecosistema Spring que la clase anotada así, se encarga de recibir excepciones

Además con @ExceptionHandler puedo indicar los métodos que hace “match” con cada tipo de excepción prevista

MANEJO DE ERRORES

```
@ControllerAdvice(basePackages =
{"org.springframework."} )
public class GestionaErrores {
    @ExceptionHandler(Throwable.class)
    public String errores (Exception e)
    {   return "error";
    }
```

Validaciones

Las validaciones de las entradas son claves y punto de riesgo común.

Validar con expresiones regulares (u otras herramientas) tanto en el cliente como en el servidor se convierte en algo obligatorio e imprescindible de cara a garantizar la seguridad

Validaciones

EJEMPLO **VALIDACIÓN JS**

```
const E_R_TELEFONO = /^\+\d{7,15}$/;
Var patron = new RegExp
(E_R_TELEFONO);

Patron.test (cadena) == true || false
```

Validaciones

EJEMPLO **VALIDACION JAVA**

final String PATRON_MAIL = "\\w ([.-]?\\w+)*@\\w+([.-]?\\w+)*(.\\w{2,4})+";

Validaciones

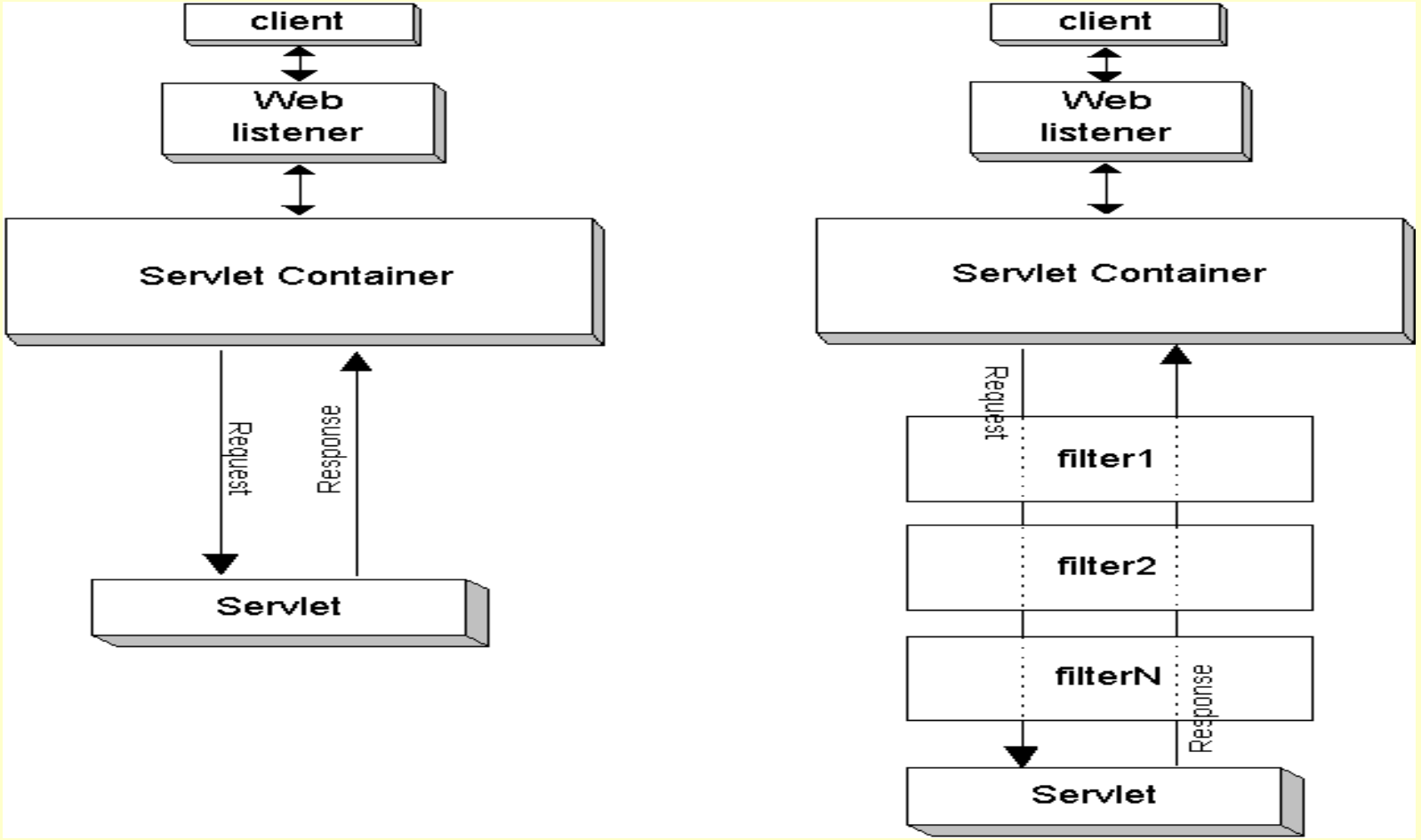
```
public static boolean emailValido (String email)
    {
        boolean bdev = false;

            Pattern p = Pattern.compile(PATRON_MAIL);
            Matcher m = p.matcher(email);
            bdev = m.matches();

        return bdev;
    }
```

Filtros

Los filtros son código Java, que se ejecutarán antes y después de la ejecución de un Servicio Web o Servlet

Es una herramienta importante que proporciona el estándar JEE y un punto clave donde incrustar nuestra operativa de seguridad

Filtros


client
Web listener
Servlet Container
Request
Response
Servlet
client
Web listener
Servlet Container
Request
filter1
filter2
filterN
Response
Servlet

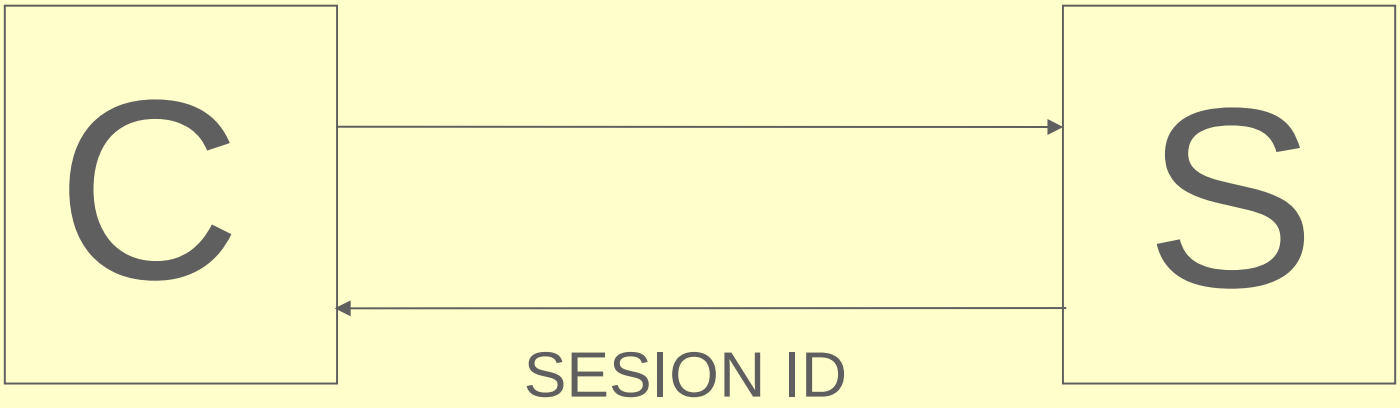
SESIONES HTTP

La sesión puede usarse como elemento de control para saber si:

- Una petición proviene de un usuario autenticado y autorizado para ello
- La petición se produce en un momento de validez (sesión no caducada)

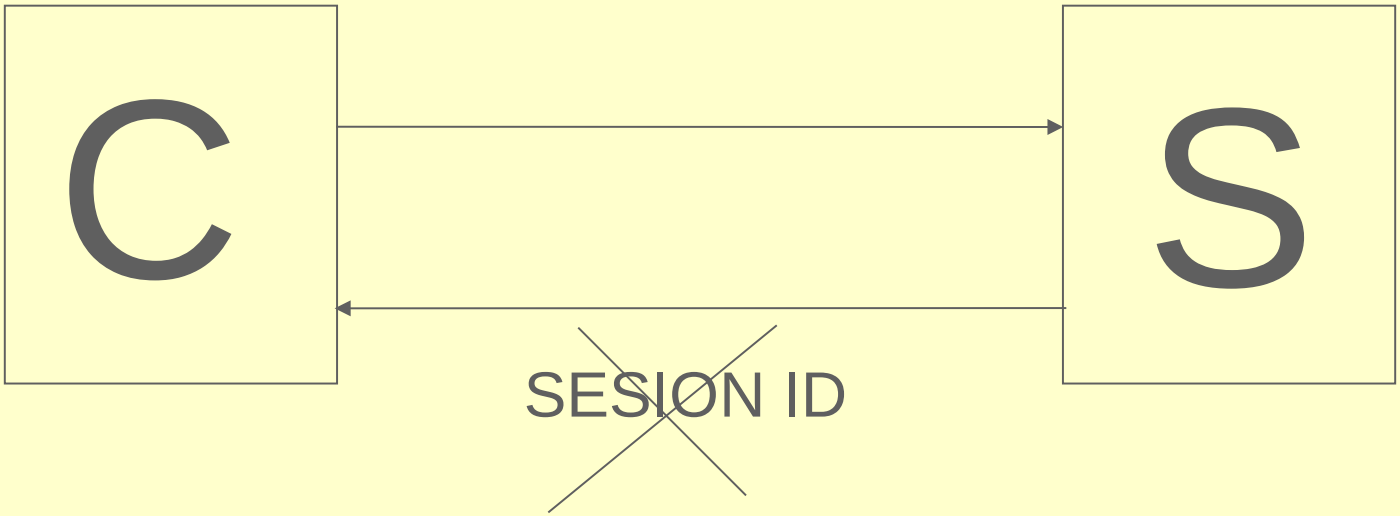
Es por tanto otro elemento básico para la gestión de la seguridad en entornos web

getSession () - (true)



getSession () - (true)

getSession (false)



getSession (false)

Fin de una Sesión

En algún momento, el usuario finalizará la sesión con el servidor.

El método invalidate() será invocado y con ello, el objeto HttpSession y los atributos asociados a él, desaparecen

Fin de una Sesión

- Podemos llamar a invalidate()
- Explícitamente (el usuario pulsa salir-logout)
- Implícitamente, por ausencia de peticiones
- Por defecto, una sesión muere a los 30’
- Fijado por setMaxInactiveInterval(secs)
- Parametrizado web.xml (minutos)

```
<session-config>
    <session-timeout>2</session-timeout>
</session-config>
```

Gestión de Sesión Spring

```
 <http create-session= *>
<session-management
session-fixation-protection = *
invalid-session-url = *>
</session-management>
</http>
```

Atributo invalid-session-url

```
<session-management invalid-session-
   url="/invalidSession.htm" />
</http>
```

Redirijo a la web/servicio deseado

Atributo create-session

always – SIEMPRE se crea una sesión Nueva (si no existe)

ifRequired – Opción por defecto

never – Spring no crea (sí reutiliza)

stateless – No se usa ninguna sesión

Atributo
session-fixation-protection

none – No hace nada. Se mantiene la sesión
newSession – Se crea una nueva sesión, nueva, sin datos de la anterior
migrateSession – Se crea una nueva sesión, manteniendo los valores anteiores
ChangeSessionId – Sólo se cambia el ID (3,1)

ESAPI

Muchos de estos casos, por tratarse de aspectos comunes, están tratados por librerías como ESAPI, que permite parsear, validar y prevenir entradas maliciosas.

ESAPI está disponible en librerías JAVA MAVEN y es mantenido por Open Web Application Security Project (OWASP)

ESAPI - ANTISAMY

En la librería ESAPI destaca el proyecto AntiSamy, dedicado a la validación de HTML Y CSS

Con AntiSamy, puedo prevenir la aparición de código malicioso en su perfil, comentarios, etc..

AUTH0

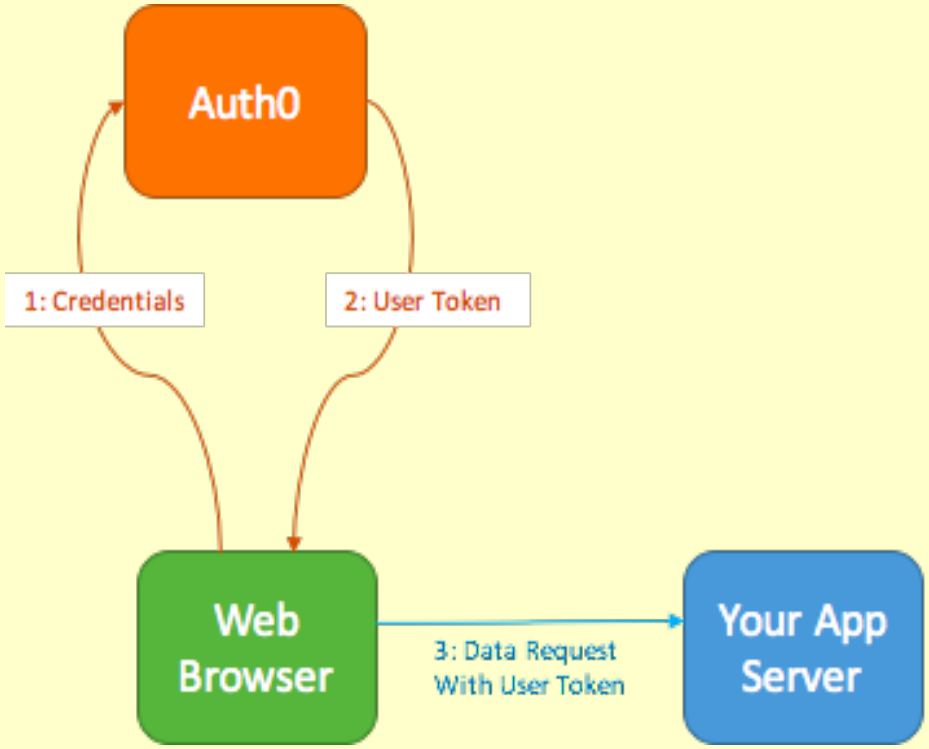
Auth0 es un servidor dedicado a autenticar a usuarios. Está pensado para gestionar diferentes aplicaciones y clientes, de modo que la misión de autenticar, quedé externalizada a nuestro proyecto

AUTH0

Contiene funcionalidades gratuitas y otras de cobro, permitiendo además autenticarse ante servicios externos como Google, Facebook o Github

Está diseñado además para integrarse con diferentes plataformas

AUTH0 - Esquema



JWT

JSON Web TOKEN es una de las últimas tendencias en la autenticación de usuarios WEB

Básicamente es un Código String que recibe el usuario al autentincarse y al que acompañan otras informaciones

JWT

Campos como tiempo de validez, dominio para el que está autenticado, datos del usuario y la firma del servidor (lo que garantiza autenticidad, integridad y no repudio)

Está admitido como un protocolo estándar y se describe formalmente en el RFC7519

JWT Compacto

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
ZdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpv
G4gRG9lIiwiYWRtaW4iOnRydWV9.**TJVA95**
RM7E2cBab30RMHrHDcEfxjoYZgeFONFh7
gQ

JWT Compacto

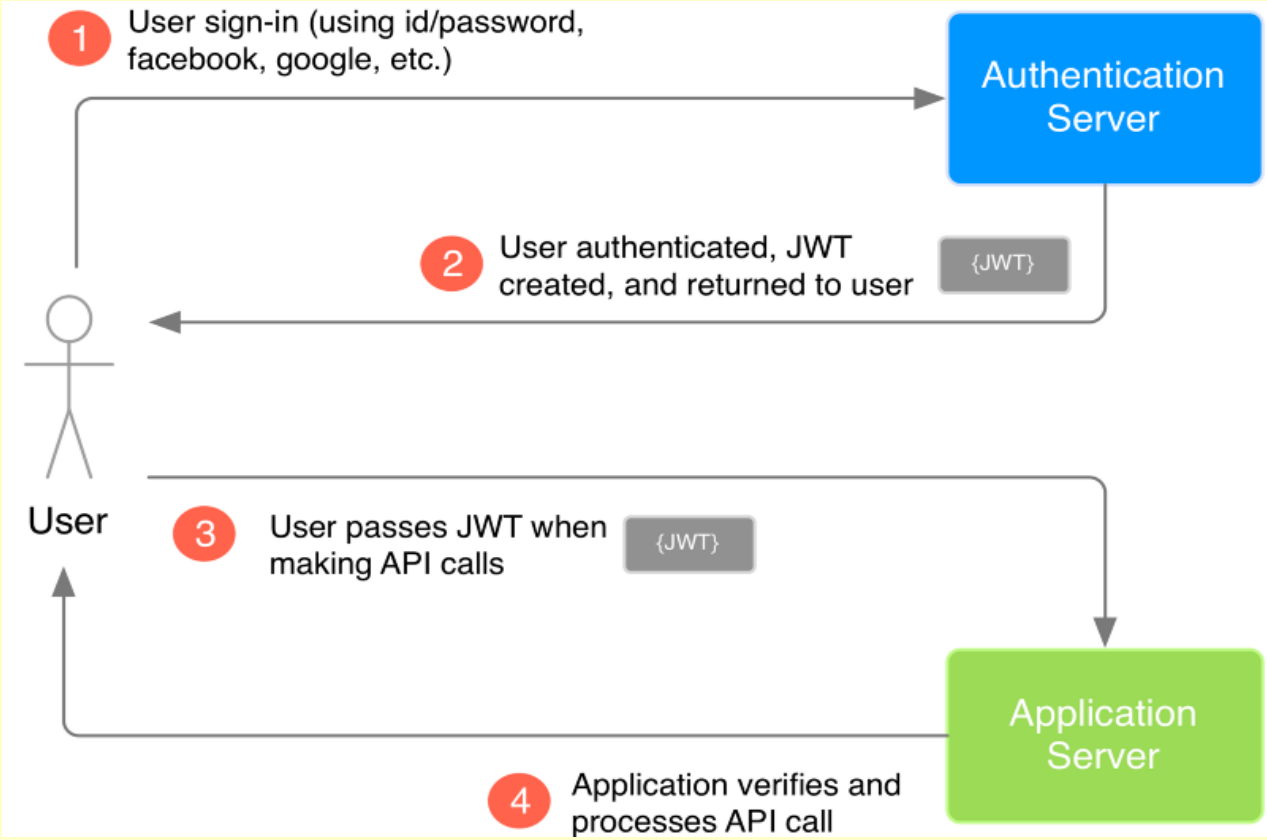
```
{ //Cabecera - HEADER
  "alg": "HS256",
  "typ": "JWT"
}
{ //Contenido -PAYLOAD
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

JWT Librerías

Hay diversas implementaciones de JWT tanto para firmar y generar mensajes, como para verificarlos.

Jsonwebtoken, JOSE, auth0

AUTH0 - JWT


1
User sign-in (using id/password, facebook, google, etc.)
Authentication Server
2
User authenticated, JWT created, and returned to user
{JWT}
User
3
User passes JWT when making API calls
{JWT}
Application Server
4
Application verifies and processes API call

ENLACES DE INTERÉS

Spring Security Ejemplos

HTTPS y Certificados

HTTPS y Certificados (CT)

HTTPS y Certificados (CA Google)

JWT Json Web Token

Vulnerabilidades comunes

Vulnerabilidades abordadas desde JAVA

ENLACES DE INTERÉS

INYECCIÓN SQL ORACLE

Auth0 Servicios de Autenticación

On line JSON Viewer

NOTICIAS DE Seguridad

PGP Guía fácil

Mailvelope Guía fácil