# Prácticas de Seguridad Informática

—

# Práctica 6

—

# Grado Ingeniería Informática

**Curso 2013-2014**

Universidad de Zaragoza
Escuela de Ingeniería y Arquitectura
Departamento de Informática e
Ingeniería de Sistemas
Area de Lenguajes y Sistemas Informáticos

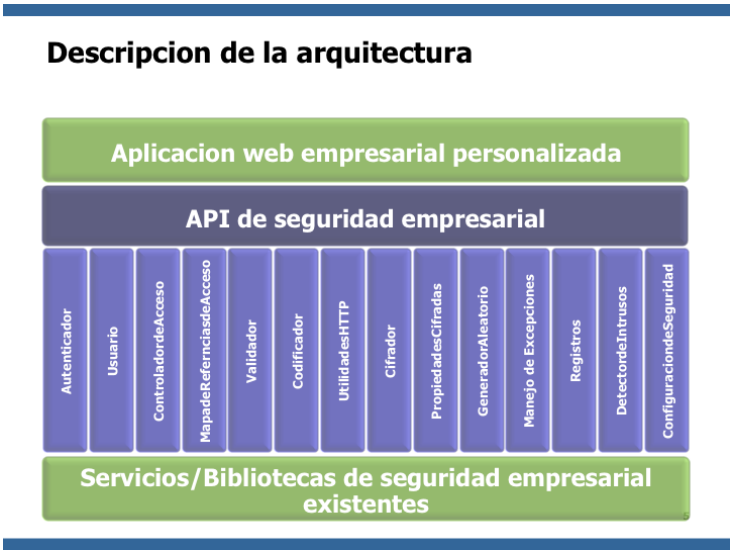
17 de diciembre de 2014

# Introducción

En clase hemos hablado de la importancia de validar los datos de entrada de las aplicaciones y también de la necesidad de sanear y codificar adecuadamente los datos que ofrecen nuestras aplicaciones a otros subsistemas.

Este trabajo es posible hacerlo por nuestros propios medios pero desde hace unos años el proyecto Open Wep Application Security Project (OWASP[1]) ha estado trabajando en el desarrolo de una biblioteca denominada Entreprise Security API[2] (ESAPI). Esta biblioteca puede simplificar bastante nuestro trabajo, siempre que podamos utilizarla.

La biblioteca se organiza en los módulos que podemos ver en la siguiente figura:




Como puede verse, se ofrecen servicios muy diferentes de los que vamos a prestar especial atención a los siguientes:

- Validador
  - Canonicalizador
- Codificador

## Validador

El módulo Validador (*ESAPI Validator*) proporciona un conjunto de métodos que permiten validar las entradas de una aplicación. Incluye algunos tipos de validación ya preparados y además nos permite crear nuestros propios validadores (también sería posible modificar los disponibles).

[1] `https://www.owasp.org/`

[2] `https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API`. A pesar de que recientemente ha habido ciertas dudas sobre su continuidad, parece que sigue en desarrollo.

**Canonicalizador**

Lo primero que haremos con cualquier cadena que recibamos como dato de entrada será convertirla a la forma más simple posible.

Para ello confiaremos en el método:

```
String canonicalize(String input) throws EncodingException;
```

que reduce una cadena de caracteres a su forma canónica (¿qué tipos de problemas resuelve?).

Posteriormente procederemos a su validación, que consiste en ver si se ajusta a las reglas que hayamos definido para el tipo de dato que pretendamos manejar.

Por ejemplo, el método `isValidCreditCard` permite hacer justamente lo que su nombre sugiere: comprobar que el dato que se le proporciona como entrada corresponde a un número de una tarjeta de crédito, según esta expresión regular:

```
Validator.CreditCard=^(\\d{4}[- ]?){3}\\d{4}$
```

Nótese que sólo hace validación del formato[3].

El método está definido como sigue:

```
boolean isValidCreditCard(String context, String input, boolean allowNull)
throws IntrusionException;
```

Sus parámetros son:

- String context - Un nombre para el parámetro que queremos validar. Se utiliza en cualquier proceso de registro (*logging*) o error que pueda ser necesario.
- String input - La entrada que queremos validar.
- boolean allowNull - Si se pone valor **true** la entrada puede ser un valor vacío (*NULL*). Si se pone valor **false** el valor (*NULL*) o la cadena vacía producirá un error.
- throws IntrusionException - Si la entrada no valida se puede generar una excepción que lo indica.

Podemos añadir otros métodos de validación para diferentes tipos de datos añadiendo las expresiones regulares adecuadas el fichero `esapi.properties`[4].

Vamos a suponer que tenemos un formulario con los siguientes datos:

[3] Algunos de los dígitos tienen significado (por ejemplo, el banco emisor, número de cuenta...) y además el último es un dígito de control que garantiza que el resto son correctos; esa verificación no se hace con la expresión regular.

[4] Podemos ver un vídeo donde lo explican en `http://www.youtube.com/watch?v=4zElrFVRS6M` y también algunas pistas -aunque son para ficheros de configuración en web las ideas son las mismas- en `http://community.jaspersoft.com/documentation/jasperreports-server-administration-guide-beta/configuring-input-validation`, en la sección 'Creating Validator Expressions'. El fichero está disponible en:

`./src/test/resources/esapi/ESAPI.properties`

| Dato | Contenido | Longitud máxima |
|---|---|---|
| Nombre | a-zA-Z0-9'- &. | 50 |
| Dirección | a-zA-Z0-9'-&,./º | 50 |
| Tarjeta de crédito | | |
| Tipo | (MC—VISA—AMEX) | 4 |
| Número | 0-9 | 16 |
| Mes Expira | 0-9 | 2 |
| Año Expira | 0-9 | 4 |
| CVN | 0-9 | 3 |
| DNI | 0-9A-Z | 9 |

Vamos a utilizar los métodos disponibles (y añadir lo que sea necesario) para validar una hipotética entrada de este formulario (para simplificar la práctica leeremos directamente de la entrada estándar, un dato por línea[5]).

## Codificador

Vamos a suponer que necesitaremos pasar los datos de nuestro formulario a algunos subsistemas. Para ello utilizaremos los métodos:

1. `String encodeForSQL(Codec codec, String input);`

   Permite codificar una entrada para su uso en una instrucción SQL, de acuerdo al *codec* seleccionado (Por ejemplo: MySQLCodec).

   **Ejemplo**

   ```
   MySQLCodec codec = new MySQLCodec();
   ESAPI.encoder().encodeForSQL(codec, data1);
   ```

   **Probar:** ¿Qué sucede cuando la cadena `input` contiene "`Paul O'Malley`" ?

   Probar con otras.

2. `String encodeForURL(String input) throws EncodingException`

   Permite codificar una entrada para su uso en una URL.

   **Probar:** ¿Qué sucede cuando la cadena `input` contiene "`Paul O'Malley`" ? ¿Y cuándo contiene el carácter '`&`' o el carácter '`;`'?

   Probar con otras.

3. `String encodeForHTML(String input)`

   Permite condificar la entrada para su uso insertándola de manera segura (que no sea interpretada) dentro de código HTML.

[5] Recordad, para no tener que teclear todos los datos cada vez, que podemos redirigir a la entrada estándar de nuestro programa el contenido de un fichero con:
`miPrograma < miFichero`

**Probar:** ¿Qué sucede cuando la cadena `input` contiene "`Paul O'Malley`" ? ¿Y cuándo contiene el carácter '`&`' o el carácter '`;`'? ¿Y si contiene el carácter '<' o el carácter '>'?

Probar con otras.

Podemos comparar los resultados con lo que muestre nuestro buscador favorito (en la URL y en el código HTML).

Un caso de uso habitual sería:

Canonicalización → Validación → Codificación

## ¿Qué hay que hacer?

Lo primero de todo, deberemos descargar e instalar en nuestro directorio de trabajo la versión más actual de ESAPI (en este momento es la versión 2.1.0, se puede descargar de: `https://code.google.com/p/owasp-esapi-java/`.

Vamos a programar una aplicación que nos permita experimentar con todas estas funciones: leerá cadenas de caracteres de la entrada estándar y las muestra codificadas, según las diferentes posibilidades disponibles.

Por ejemplo, si la aplicación se llama `leer` podría tener las siguientes opciones:

- `-v` Valida
- `-c` Canonicaliza
- `-e` Codifica con:
  - `SQL`
  - `HTML`
  - `URL`

**Ejemplos:**

- `leer -v -c`

  Leería el formulario de la entrada estándar y si el contenido es válido de acuerdo a las reglas fijadas lo escribiría en la salida estándar en formato canónico.

  Si la entrada no es válida muestra los errores encontrados.

- `leer -c -e SQL -e URL`

  (Sin -v)

  Leería una cadena de caracteres (o varias, en líneas sucesivas) de la entrada estándar y las escribiría en la salida estándar en formato canónico para ser usadas en una instrucción SQL y en una URL (por cada cadena de entrada, dos cadenas de salida).

- `leer -v -c -e HTML -e URL`

  (Con -v)

  Leería tantas líneas como sea necesario para completar el formulario (podemos suponer que cada línea contiene un dato) de la entrada estándar y si el contenido es válido de acuerdo a las reglas fijadas las escribiría en la salida estándar en formato canónico para ser usadas dentro de código HTML y en una URL.

  Si la entrada no es válida muestra los errores encontrados.

- `leer -c -e SQL`

  (Sin -v)

  Leería una cadena de caracteres (o varias, en líneas sucesivas) de la entrada estándar y las escribiría en la salida estándar en formato canónico para ser utilizadas en una instrucción SQL.

## Bolas extra

- Programar una pequeña aplicación web que admita un formulario parecido (puede tener menos datos) al que hemos propuesto arriba y que no haga ningún tipo de validación de los datos de entrada. Comprobar que es vulnerable, por ejemplo, con inyecciones de HTML. Hacer una segunda versión que incluye la validación con ESAPI.
- Ver qué otros métodos hay disponibles en los módulos que hemos estado probando. Añadir funcionalidades a nuestro programa.
- Relacionado con la práctica anterior. ESAPI incluye un módulo denominado `Cifrador` (`Encryptor`). Comprobar los servicios que ofrece y comparar con los nativos de Java. Podríamos hacer medidas de coste temporal similares a las que realizamos en la práctica anterior
- Elegir otro módulo diferente de ESAPI y añadir algunas características del módulo a nuestro programa.

### Documentación adicional

- OWASP Enterprise Security API

  `https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API#tab=Home`

  - PROJECT INFO What does this OWASP project offer you?

    `https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API#tab=Project_Details`

    - Data sheet, PDF:
      `http://www.owasp.org/images/8/81/Esapi-datasheet.pdf`
    - Project presentation, PowerPoint
      `http://owasp-esapi-java.googlecode.com/files/OWASP ESAPI.ppt`

- ESAPI 2.0.1 API

  `http://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/index.html`

- ‘Input Validation using the OWASP ESAPI’
  `http://www.securityninja.co.uk/application-security/input-validation-using-the-owasp-esapi/`
- ‘Output Validation using the OWASP ESAPI’
  `http://www.securityninja.co.uk/application-security/output-validation-using-the-owasp-esapi/`