



XANITIZER

Ejemplo Guiado

Autores:

Miguel Carbayo Fernández
Adrián Celis Fernández
Carolay B. Corales Acosta
Luis Cruz Varona
Jaime López-Agudo Higuera
Elena Romón López

ÍNDICE

1.	<i>Objetivo del documento</i>	3
2.	<i>Creación de un proyecto.....</i>	3
3.	<i>Exploración de la vista inicial</i>	5
4.	<i>Creación de una Call Graph</i>	6
5.	<i>Ejecutar un análisis de seguridad</i>	6
6.	<i>Historial de mantenimiento (snapshots) y usos.....</i>	8
7.	<i>Generar informes.....</i>	10

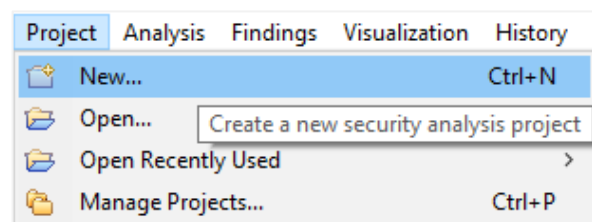
1. Objetivo del documento

El objetivo de este documento es el de mostrar un ejemplo claro de cómo trabajar con Xanitizer y a la vez familiarizarse con las funcionalidades que ofrece dicho software. El tiempo estimado para completar este ejemplo es de xx minutos.

2. Creación de un proyecto

En este apartado vamos a realizar la creación de un nuevo proyecto en Xanitizer, analizando un proyecto ya creado de ejemplo llamado WebGoat que se encuentra ya descargado dentro de la carpeta de instalación de Xanitizer. Para iniciar el software entramos en la carpeta de instalación y ejecutamos Xanitizer.exe.

Para comenzar la creación de un nuevo proyecto, ejecutamos “**Project > New...**” o pulsamos el comando **CTRL+N**. Se nos abrirá un asistente para la creación de un nuevo proyecto.



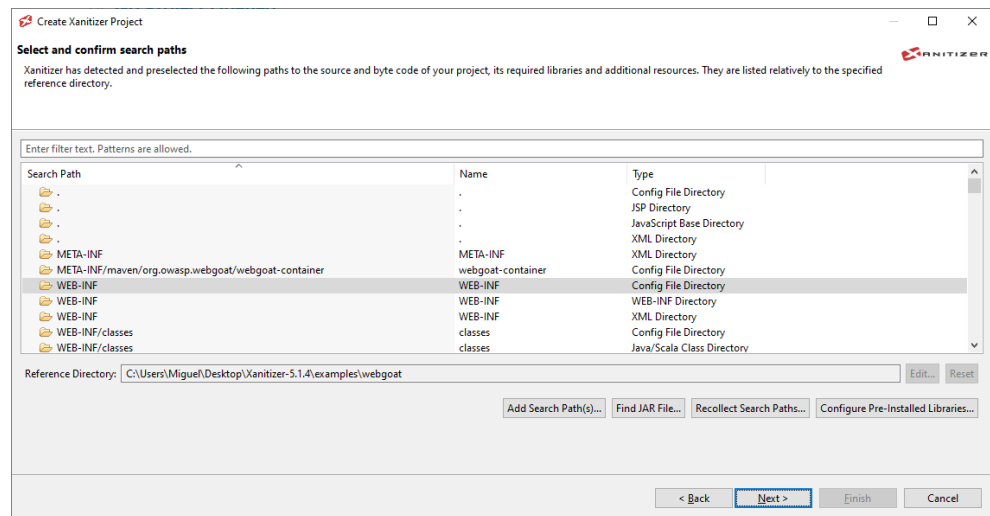
A continuación, debemos introducir los datos necesarios:

- **Project Name:** nombre del proyecto. → “Ejemplo”
- **Project Root Directory:** directorio del proyecto a analizar. → “<xanitizer-dir>/examples/webgoat”
- **Configuration File Directory:** directorio del archivo de configuración de Xanitizer. La dirección puede ser distinta a la Project Root Directory.

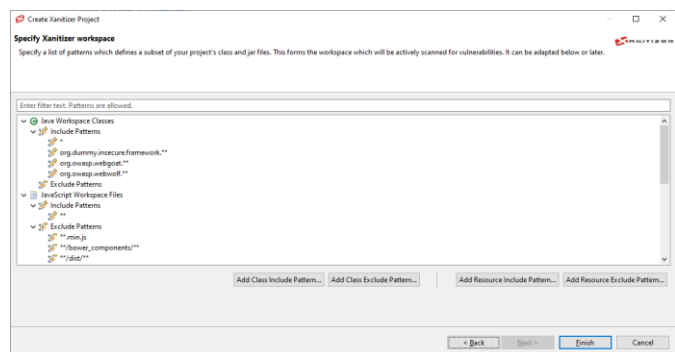
A screenshot of the 'Create Xanitizer Project' dialog box. The title bar says 'Create Xanitizer Project'. Inside, it says 'New Project' and 'Create a new Xanitizer project. All the settings on the following pages can be changed later in the Configuration Editor.' There are three input fields: 'Project Name:' with 'Ejemplo' entered, 'Project Root Directory:' with 'C:\Users\Miguel\Desktop\Xanitizer-5.1.4\examples\webgoat' and a 'Browse...' button, and 'Configuration File Directory:' with the same path and a 'Browse...' button. At the bottom, there are buttons: '< Back', 'Next >' (highlighted), 'Finish', and 'Cancel'.

Hacemos click en “**Next >**”. Es posible que se muestre alguna ventana de aviso. La ignoramos pulsando “**OK**” a cualquier mensaje.

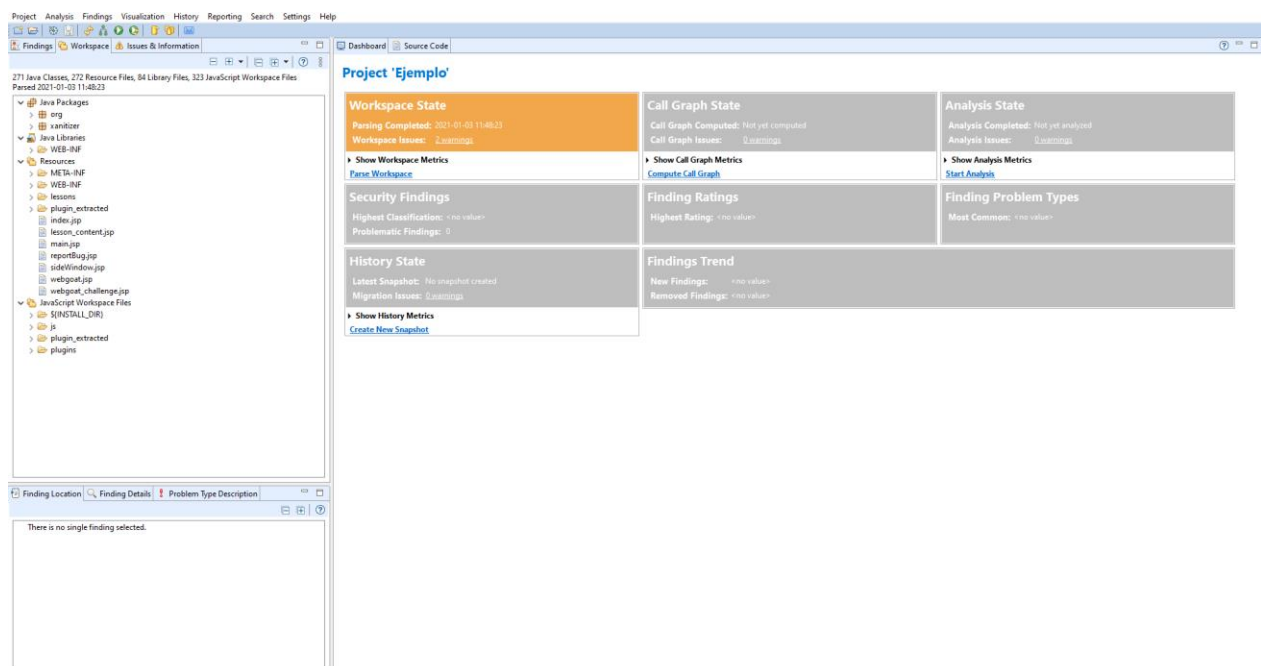
En la siguiente ventana podemos seleccionar qué carpetas del proyecto queremos analizar. De momento, dejamos todo como está para que se busquen en todas las carpetas del proyecto y pulsamos **“Next >”**.



En la siguiente y última ventana podemos especificar los patrones de búsqueda de archivos. Aquí podemos incluir y excluir archivos que no queremos que sean analizados. De momento, dejamos todo como está para que se realicen las búsquedas por defecto, pulsamos **“Finish”**, y esperamos a que se termine de procesar. Este proceso llevará menos de dos minutos.



Tras ello veremos una ventana como esta:

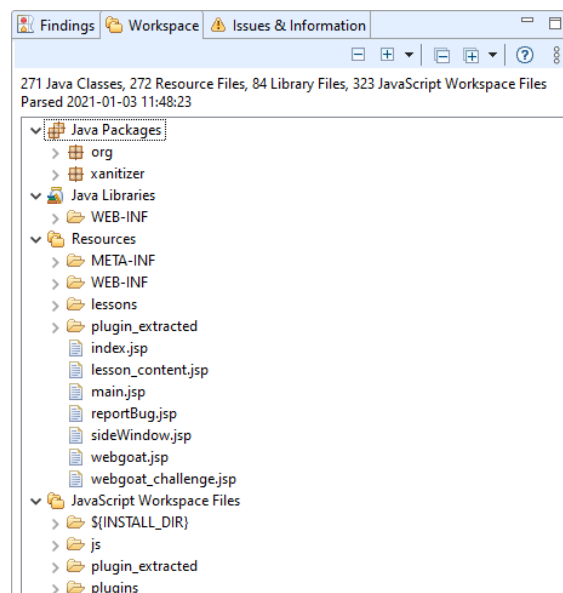


3. Exploración de la vista inicial

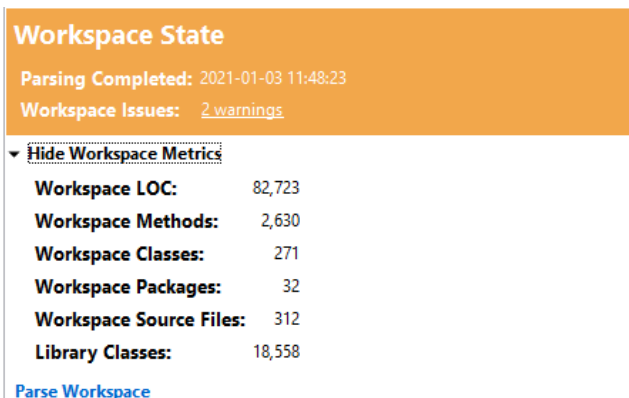
En este apartado vamos a inspeccionar los diferentes paneles que se muestran según se crea un nuevo proyecto en Xanitizer.

En el panel izquierdo encontramos 3 pestañas. Nada más crear un proyecto la pestaña **“Workspace”** debería estar seleccionada, si no lo estuviera haz click en ella para abrirla. En esta pestaña encontraremos todos los archivos del proyecto que hemos importado y vamos a analizar, divididos en 4 paquetes: *Java Packages*, *Java Libraries*, *Resources*, *JavaScript Workspace Files*.

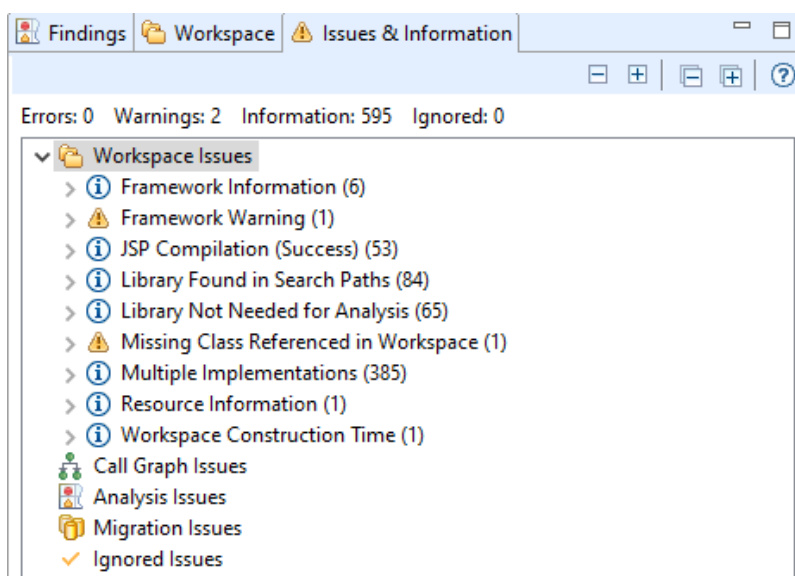
En este momento Xanitizer aún no ha ejecutado ningún análisis de seguridad sobre el código, y, por lo tanto, la pestaña **“Findings”** aparece vacía.



En el panel derecho encontramos 2 pestañas. Seleccionamos la pestaña **“Dashboard”** si no lo está ya. Aquí encontraremos los resultados de los análisis que ejecutemos. Actualmente, ya se puede observar dos problemas encontrados en el proyecto. Si hacemos click en **“Show Workspace Metrics”** podemos observar algunos datos sobre el proyecto, como las líneas de código (LOC) totales, número de archivos, etc.




Por otro lado, si clickamos en **“2 warnings”**, automáticamente en el panel izquierdo se abrirá la pestaña **“Issues & Information”**, en dónde se encuentran los errores actuales encontrados.



4. Creación de una Call Graph

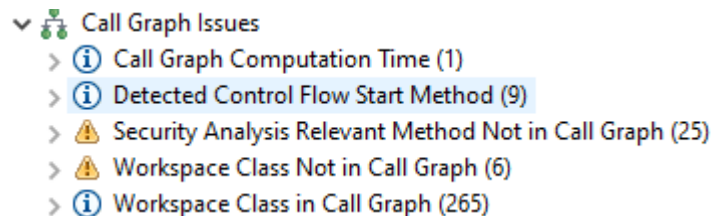
En este apartado vamos a crear una call graph. Lo que hace es analizar el código y encontrar que clases concretas se pueden crear de un objeto. Por ejemplo, si tenemos una línea de código `x.toString()` donde `x` es un `java.lang.Object`, este análisis intenta averiguar que clases concretas puede ser la variable `x`.

Para computar una call graph tenemos cuatro opciones:

- Presionar **F6** en el teclado.
- Click en **"Analysis > Compute Call Graph"**.
- En la barra superior, hacer click en .
- En el Dashboard, en el panel de *Call Graph State* hacer click en **"Compute Call Graph"**.

Automáticamente se abrirá una ventana de progreso. Este proceso debería llevar alrededor de 20 segundos.

En la pestaña "Issues & Information" encontraremos ahora una sección llamada **"Call Graph Issues"**. En ella se dan los resultados encontrados por el análisis realizado.




En *Detected Control Flow Start Method* se dan los métodos que inician una secuencia de eventos, como, por ejemplo, métodos *main*. En *Workspace Class Not in Call Graph* se dan métodos que Xanitizer no ha encontrado que sean usados (si se considera un error se pueden añadir como métodos de inicio haciendo click derecho **"Add Start Method to Configuration..."**). En *Security Analysis Relevant Method Not in Call Graph* se encuentran algunas vulnerabilidades encontradas por Xanitizer clasificadas como "Taint". Un "Taint" es un dato del programa que puede ser controlado potencialmente por un atacante.

5. Ejecutar un análisis de seguridad

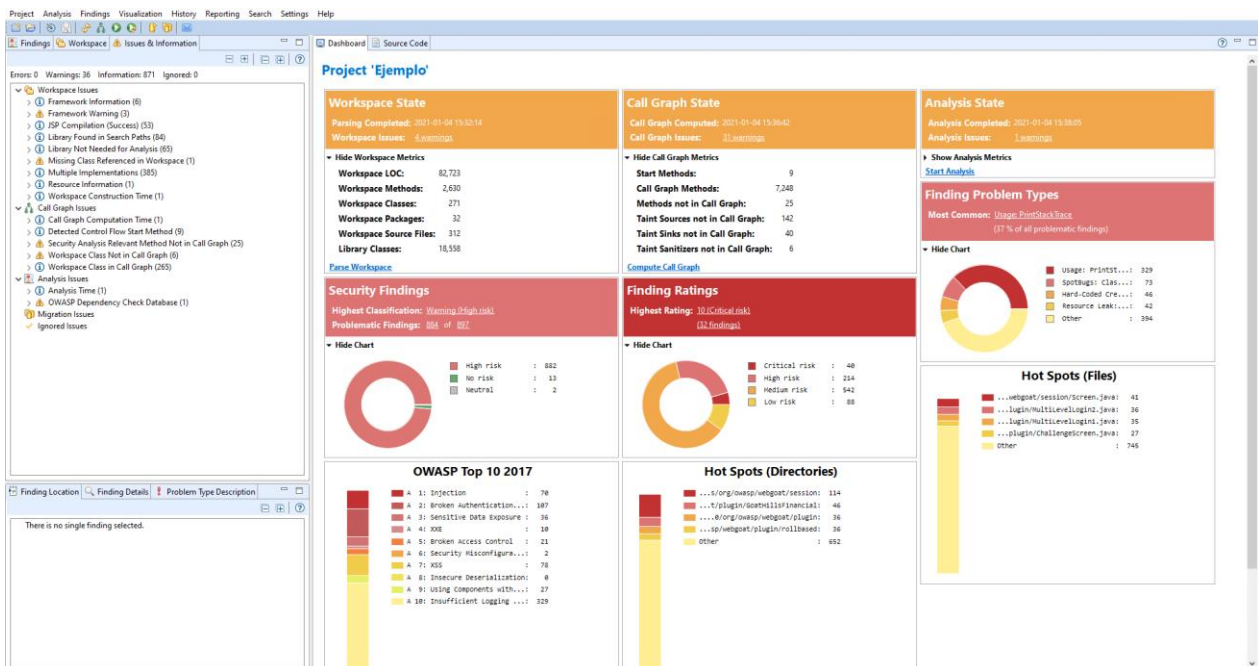
En este apartado vamos a realizar un análisis completo de seguridad sobre el proyecto WebGoat.

Para comenzar tenemos cuatro opciones:

- Presionar **F11** en el teclado.
- Click en **"Analysis > Run Security Analysis"**.
- En la barra superior, hacer click en .
- En el *Dashboard*, en el panel de *Analysis State* hacer click en **"Start Analysis"**.

Es posible que nos salga un mensaje de aviso indicando que se van a descartar los datos del workspace actual (ya que el análisis vuelve a ejecutar la Call Graph). Hacemos click en **"Continue"** si nos sale dicho aviso. Una nueva ventana de progreso se abrirá; esperamos a que se termine y se cierre automáticamente. Este proceso debería llevar entre 3-8 minutos.

Tras terminar el análisis el *Dashboard* se actualizará para mostrar los resultados encontrados. Debería mostrar una pantalla como esta:



En el *Dashboard* encontramos tres cajas de información sobre el análisis realizado:

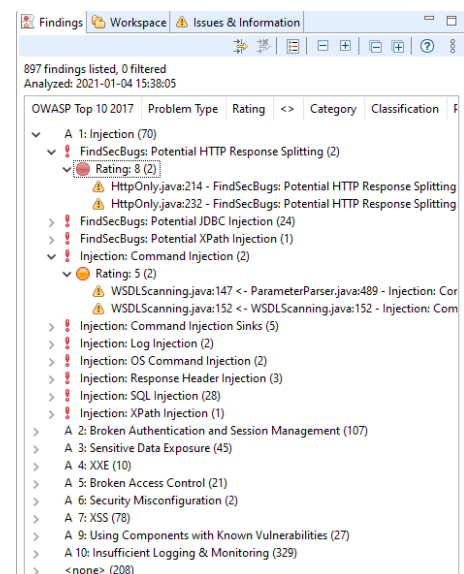
- **Security Findings:** cantidad de fallos encontrados.
- **Finding Ratings:** valoración de los fallos encontrados (crítico-alto-medio-bajo).
- **Finding Problem Types:** tipos de problemas encontrados. En otras palabras, cantidad de problemas encontrados por cada tipo.

También encontramos debajo más casillas para clasificación de problemas: *OWASP Top 10 2017* (Open Web Application Security Project; consenso sobre riesgos de seguridad de aplicaciones web más críticos), *Hot Spots (Directories)* (carpetas con más problemas), *Hot Spots (Files)* (archivos con más problemas).

Para no extender este ejemplo demasiado, vamos a observar solo algunas de las opciones de búsqueda de problemas que ofrece Xanitizer.

En el panel izquierdo, hacemos click en la pestaña **"Findings"**. Aquí se encuentran los problemas clasificados según los riesgos descritos en OWASP. Si ampliamos cualquiera de las categorías encontraremos los tipos de problemas que se dan, y si ampliamos el botón *Rating* podremos observar los problemas que se han encontrado en el proyecto de ese tipo categorizados por *Rating* (0: menor o irrelevante, 10: crítico).

Si hacemos click en un problema, en el panel derecho se abrirá automáticamente el archivo donde se encuentra el problema y se destacará la línea de código que provoca la vulnerabilidad.




Además, en la parte inferior izquierda de Xanitizer podemos ver más datos sobre el problema que se está analizando actualmente:

- **Finding Location:** en dónde se da el problema seleccionado.
- **Finding Details:** información sobre el problema encontrado (fecha de detección, valoración, localización, descripción de cómo se ha encontrado, etc.)
- **Problem Type Description:** descripción del problema, ejemplo de código vulnerable y referencias.

6. Historial de mantenimiento (snapshots) y usos

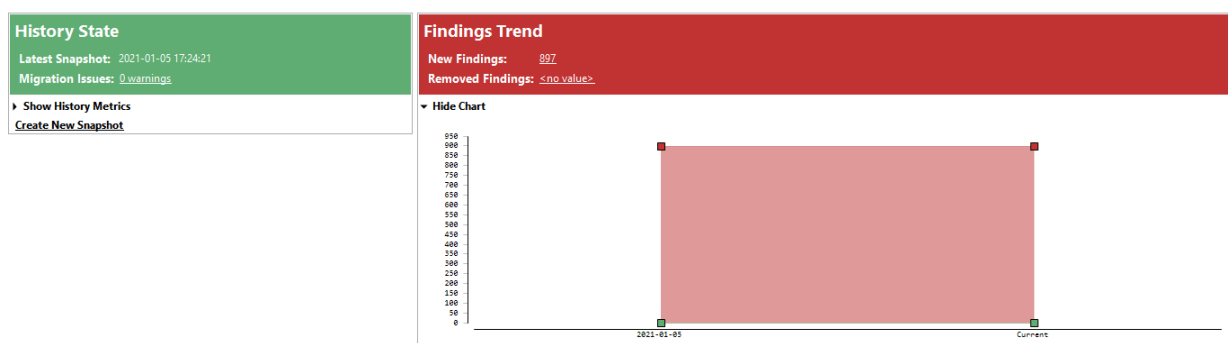
En este apartado vamos a aprender a manejar el historial de mantenimiento mediante el uso de instantáneas. Las instantáneas guardan el estado del proyecto auditado en ese momento.


Para crear una instantánea hay dos opciones:

- En la pestaña Dashboard, en el recuadro **History State** hacemos click en el link **“Create New Snapshot”**.
- Hacer click en el botón de la barra superior: 

En la ventana que se muestra podemos añadir opcionalmente un comentario. Las instantáneas guardan no solo los informes de seguridad o grafos de llamadas (call graphs) generados por Xanitizer sino también las partes del código que pertenecen a los **findings** reportados por la herramienta. Cabe destacar que, si bien las instantáneas permiten el guardado del estado del proyecto en un momento dado, los datos de esa instantánea no se pueden modificar, es decir, no se puede revertir a una versión anterior del proyecto y realizar cambios desde ahí.

Comenzamos creando una instantánea, a la que llamaremos **“Initial State”**. Una vez hecho esto, veremos como el color del recuadro de **“History State”** cambia de color a verde y, además, se crea un gráfico a su derecha, llamado **“Findings Trend”**. Este gráfico mostrará la variación de **findings** a lo largo de la vida del proyecto mediante las instantáneas tomadas.



Además de la información contenida en el **dashboard**, existe también un histórico de instantáneas que permite ver todas las instantáneas tomadas, además de poder revertir el proyecto al estado de cualquiera de ellas. Para acceder a este histórico, el usuario puede pulsar el botón de la barra superior  o seleccionar la opción de **“Manage Snapshots...”** dentro del menú **“History”** de la barra de herramientas.

Dentro del histórico aparecen tanto las instantáneas creadas, como la versión última del proyecto. La versión en la que se encuentra la herramienta actualmente se puede distinguir por usar el color de letra azul.

Para entender mejor cómo funcionan las instantáneas, cerramos Xanitizer, editamos cualquier archivo de código fuente que contenga un **finding** con cualquier editor de texto (para esta demostración usaremos el archivo

"<Xanitizer-dir>/examples/webgoat/plugin_extracted/common-1.0/org/owasp/webgoat/plugin/Exec.java"

y comentaremos la línea 108 (escribiendo *"//"* al principio de la línea), que contiene un **finding** de **Os Injection**). Una vez editado el archivo, volvemos a Xanitizer y hacemos un nuevo análisis de seguridad (pulsando **F11** por ejemplo).


Tras este nuevo análisis, la cabecera del gráfico de **"Findings Trend"** habrá cambiado, indicando que hay un nuevo **finding**, pero que también se ha corregido otro.

Para comprobar que, en efecto, se han producido cambios, abrimos en la vista de código de Xanitizer el archivo modificado. Para ello, vamos en la pestaña Workspace del panel izquierdo, y ampliamos los siguientes elementos:

Java Packages > org > owasp > webgoat > plugin > Exec

Podemos observar que la línea aparece como comentada.

```
95 public static ExecResults execOptions(String[] com
96 {
97     Process child = null;
98     ByteArrayOutputStream output = new ByteArrayOu
99     ByteArrayOutputStream errors = new ByteArrayOu
100     ExecResults results = new ExecResults(Arrays.a
101     BitSet interrupted = new BitSet(1);
102     boolean lazyQuit = false;
103     ThreadWatcher watcher;
104
105     try
106     {
107         // start the command
108         //child = Runtime.getRuntime().exec("");
109         // get the streams in and out of the comma
110         InputStream processIn = child.getInputStre
111         InputStream processError = child.getErrorS
112         OutputStream processOut = child.getOutputS
113
```

Ahora volveremos al snapshot anterior, para ello, vamos al histórico de instantáneas (**Manage Spapshots**, botón ) y hacemos doble click en la instantánea creada previamente. Esto revertirá la versión del proyecto a la versión inicial. Podemos comprobarlo abriendo de nuevo el archivo modificado. Si todo ha ido bien, la línea que comentamos anteriormente debería aparecer ahora como no comentada.

Para terminar, volvemos a ponernos en la última versión del proyecto para continuar con el taller.

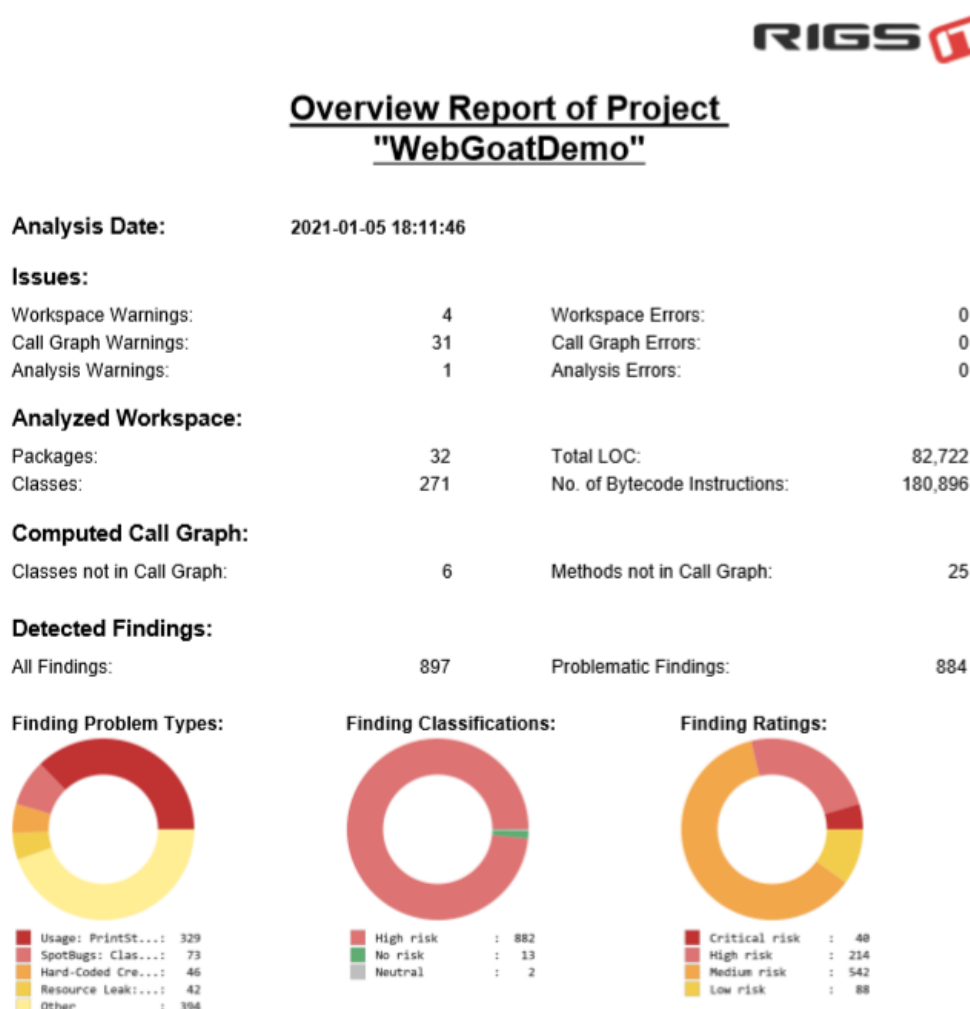
7. Generar informes

Por último, Xanitizer permite la creación de informes de seguridad a partir de los análisis que realiza. Hay tres tipos de informes que permite generar la herramienta:

- **Overview Report:** un informe general de los resultados del análisis.
- **Findings List Report:** un informe en el que se detallan todos los **findings** encontrados al realizar el análisis.
- **Detailed Report:** un informe detallado sobre uno de los **findings**, en el que se muestra el problema, y el camino que sigue la posible brecha de seguridad.

Nos vamos a centrar primero en generar un informe general, para ello seleccionamos “**Reporting > Generate Overview Report...**” en la barra de herramientas. Se abrirá una ventana emergente con varias opciones. Dejamos seleccionada la opción de **PDF** como formato para nuestro informe. En el campo de **Report Output File** seleccionamos cualquier directorio que queramos y le damos el nombre de “**Informe_Taller_General**”. Una vez hecho esto, pulsamos el botón “**Select Trend Charts**” y deseccionamos todo (botón **Deselect all**), para luego seleccionar en la categoría **Problem Type Metrics** solo la opción **Findings (Injection: OS Commandline Injection)**. Tras esto, pulsamos “**Generate Report**” para generar el informe.

Cuando se termine de generar el informe, se abrirá en nuestro lector predeterminado de PDFs, y tendrá un aspecto similar a este:



Además de lo mostrado en pantalla, el informe contendrá también el gráfico que muestra el número de **findings** de **OS cmd injection** a lo largo del tiempo (debería aparecer como una línea plana, ya que no hemos resuelto ninguno), y las librerías de java que usa el proyecto.

También vamos a aprovechar y generar un informe detallado sobre un **finding**. Para ello, seleccionamos en la pestaña **Findings**, y ampliamos

A 1: Injection (70) > Injection: OS Command Injection (2) > Rating: 10 (2) > Exec.java:108

Una vez lo hemos seleccionado y se nos ha abierto la vista de su código, seleccionamos la opción **“Report > Generate Detailed Report”** en la barra de herramientas. Nos aparecerá una ventana similar a la del **Overview Report** pero esta vez sin la opción de seleccionar Trend Charts. De nuevo elegimos **PDF** como formato de salida, y llamamos al informe **“Informe_Taller_Detallado”**. Por último, generamos el informe pulsando **“Generate Report”**.

Una vez generado y abierto el informe, veremos que nos aparecen detalles sobre el **finding** seleccionado.



Detail Information about Security Finding 6m5fmvfew7pjsnvxj3422fd8 in Project "WebGoatDemo"

Analyzed at 2021-01-05 18:11:46

Analyzed Workspace

Classes:	271	Total LOC:	82,722
Packages:	32	No. of Bytecode Instructions:	180,896

Computed Call Graph

Classes not in Call Graph:	6	Methods not in Call Graph:	25
----------------------------	---	----------------------------	----

Finding ID:	6m5fmvfew7pjsnvxj3422fd8	Problem Type	Injection: OS Command Injection (Taint Paths)
Classification:	Warning	First detected at:	2021-01-05 18:11:46 - new
Rating:	10.00	Reviewed State:	Not Reviewed
Location:	Exec.java (<Source Code>/org/owasp/webgoat/plugin):108 <- ParameterParser.java (<Source Code>/org/owasp/webgoat/session):516		
Tags:	no tags assigned		

Comment: no comment

Description: An identified taint path for the problem type 'Injection: OS Command Injection'

Matching pattern in taint source kind 'Servlet Request Input':
java.lang.String[] javax.servlet.ServletRequest.getParameterValues(java.lang.String)

Matching pattern in taint sink kind 'OS Command Injection':
java.lang.Process java.lang.Runtime.exec(java.lang.String[])

Problem Type Description: This security problem represents the dynamic construction of operating system commands using data which originates from the web client.

See CWE number 78 for details.

How to fix: Check that only allowed characters are used, because the value can be controlled from the outside.

Además, a lo largo del informe se mostrará el camino que sigue el parámetro y que puede ocasionar una brecha de seguridad. Para ello, se muestran las líneas de código concretas por las que pasa dicho parámetro, acompañadas por una “pista” (**hint**) y el número del paso que ocupa dicha línea en el camino que sigue dicho parámetro problemático:

Step	Data Flow Hints	Line	Code Excerpt in Method 'getRawParameter(...)'
		514	*/
		515	public String getRawParameter(String name) throws ParameterNotFoundException {
1	Data flow in from Taint Source: getParameterValues(...); Taint value ID 6 (type: java.lang.String[]) 'values';	516	String[] values = request.getParameterValues(name);
2		517	
3	Taint value ID 6 (type: java.lang.String[]) 'values';	518	if (values == null) {

Para el último tipo de informes no vamos a hacer práctica, ya que su ejecución es muy similar y lo único que reporta es un listado de todos los **findings** encontrados en el proyecto con muy ligera información en el siguiente formato:

Finding ID:	6m5fmvfefw7pjsnvxj3422fd8	Problem Type:	Injection: OS Command Injection (Taint Paths)
Classification:	Warning	Date:	2021-01-05 18:11:46 - new
Rating:	10.00	Reviewed State:	Not Reviewed
Location:	Exec.java (<Source Code>/org/owasp/webgoat/plugin):108 <- ParameterParser.java (<Source Code>/org/owasp/webgoat/session):516		
Tags:	no tags assigned		
Comment:	no comment		
Description:	An identified taint path for the problem type 'Injection: OS Command Injection'		
	Matching pattern in taint source kind 'Servlet Request Input': java.lang.String[] javax.servlet.ServletRequest.getParameterValues(java.lang.String)		
	Matching pattern in taint sink kind 'OS Command Injection': java.lang.Process java.lang.Runtime.exec(java.lang.String[])		

8. Referencias

- Más recursos... <https://www.rigs-it.com/resources/>
- Tutorial con más explicaciones y funciones...
<https://www.rigs-it.com/wp-content/uploads/xanitizertutorial.pdf>