CAI 3. CONSULTA SOBRE SEGURIDAD DEL SOFTWARE EN EL DESARROLLO DE SISTEMAS SEGUROS (SAFETY) EN UNA CENTRAL NUCLEAR

DATOS
Fecha: 15/03/2020
Edición: 4

AUTORES
Equipo de trabajo 6

Control de Cambios

EDICIÓN	CAMBIOS	AUTOR	FECHA
1	Exportación de plantilla de tablas	Equipo de trabajo	11/03/2020
2	Creado resumen ejecutivo, ejecutado Informe con SpotBugs	Equipo de trabajo	13/02/2020
3	Ejecutado informe con Codacy	Equipo de trabajo	14/02/2020
4	Ejecutado informe con PMD, versión final del documento	Equipo de trabajo	15/02/2020

Contenido

Resumen Ejecutivo

Para este CAI, se pide al equipo de trabajo que verifique la seguridad del dos desarrollos que se nos proporcionan, buscando vulnerabilidades en el código fuente de ambos y viendo maneras de solucionar dichos errores. Además, se da libertad al equipo de desarrollo para usar las herramientas que consideremos más adecuadas, por lo que hemos elegido tres herramientas diferentes (una por cada miembro del equipo):

SpotBugs: Una herramienta de análisis estático de código que permitirá detectar vulnerabilidades, además de errores de sintaxis, posibles errores en la declaración de variables…

Codacy: Codacy es una herramienta muy utilizada por los equipos de desarrollo, que automatiza el trabajo del análisis de código y cobertura de este.

PMD: es una herramienta de calidad de código encargada de validar los estándares de construcción de un desarrollo, por lo que analiza sintácticamente, además de otros errores que pueden perjudicar a la calidad del código.

Análisis del Desarrollo 1

En este desarrollo, encontramos una única clase java, ServerSensores.java. Hemos analizado el código con las herramientas anteriormente mencionadas, y hemos obtenido los siguientes resultados:

(NOTA: Debido al gran tamaño de las tablas, se incluyen en las siguientes páginas apaisadas)

Análisis con SpotBugs

En la siguiente tabla pueden verse los resultados:

Leyenda de la tabla:

Color de celda naranja, blanco: bug
Color de celda rojo: Falso positivo

Nº	Vulnerabilidad de Seguridad por errores software	Explicación del código de error	Prioridad	Paquete.clase. funcion. línea código	Sanitización Propuesta	Comentarios
1	DLS_DEAD_ LOCAL_STORE	Se asigna un valor a una variable local, pero ese valor ni se lee ni se usa en ninguna instrucción posterior	1	sensoresReactor. ServerSensores. main. 43	Eliminar la propiedad “DatosSensores”, que no se usa nunca.	No procede.
2	DM_DEFAULT_ ENCODING	Confianza en la codificación por defecto. Se ha encontrado una conversión de Byte a String o viceversa, que asume que la plataforma convierte correctamente y en el mismo formato. Esto puede provocar que el funcionamiento cambie entre plataformas.	1	sensoresReactor. ServerSensores. listen. 30	Especificar un charset al InputStreamReader para que use ese independientemente de la plataforma.	No procede.
3	NM_FIELD_ NAMING_ CONVENTION	Los nombres de campos que no son finales deben seguir el estilo camelCase (primera letra minúscula, primera letra de otras palabras en mayúscula)	2	sensoresReactor.Serve rSensores.14	Cambiar el nombre de la propiedad “DatosSensores” a “datosSensores”	No confundir la propiedad “DatosSensores” de la clase ServerSensores con la variable “DatosSensores” que se crea en el método main de la misma.

4	NP_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD	Hay un valor público o protegido que no está siendo inicializado correctamente.	2	sensoresReactor.ServerSensores.main.34	Inicializar la propiedad "DatosSensores" dentro del constructor.	No confundir la propiedad "DatosSensores" de la clase ServerSensores con la variable "DatosSensores" que se crea en el método main de la misma. En este caso nos referimos a la variable.
5	UWF_UNWRITTEN_PUBLIC_OR_PROTECTED_FIELD	Un campo público o protegido en el cual no se escribe nunca.	2	sensoresReactor.ServerSensores.main.34	Revisar la propiedad "DatosSensores" para comprobar su correcto funcionamiento	E Método listen() añade datos a DatosSensores. Pero este bug surge porque la propiedad no está inicializada, como podemos ver en la documentación de SpotBugs: *"...Check for errors (should it have been initialized?) ..."*

Análisis con Codacy

En la siguiente tabla pueden verse los resultados:

Nº	Explicación del código de error	Prioridad	clase. línea código	Sanitización Propuesta	Comentarios
1	Variable sin usar	1	ServerSensores.43	La variable 'DatosSensores' no se usa, eliminela	Evite variables locales no utilizadas como 'DatosSensores'.

En este caso, Codacy ha encontrado muchos menos errores en el desarrollo 1 que la anterior herramienta, lo que demuestra la importancia de analizar el código siempre con tantas herramientas distintas como se pueda.

Análisis con PMD

Nº	Vulnerabilidad de Seguridad por errores software	Explicación del código de error	Prioridad	Paquete.clase. funcion. línea código	Sanitización Propuesta	Comentarios
1	FIELD_NAMING_ CONVENTIONS	El nombre de este atributo no sigue el estilo camelCase	2	sensoresReactor. ServerSensores. main. 14	Nombrar al atributo siguiendo el estilo correcto.	No procede.
2	VARIABLE_NAMING_ CONVENTIONS	El nombre de la variable comienza en mayúscula	2	sensoresReactor. ServerSensores. main. 43	Comenzar el nombre de la variable en minúscula.	No procede.
3	PACKAGE_CASE	El nombre del paquete contiene mayúsculas	2	sensoresReactor. ServerSensores. main. 1	Nombrar al paquete todo en minúscula	No procede.
4	IF_ELSE_STMTS_ MUST_USE_BRACES	Declaración if...else hecha sin paréntesis	2	sensoresReactor. ServerSensores. main. 18	Uso de llaves en el condicional if...else	No procede.
5	CLOSE_RESOURCE	Recurso no cerrado después de usarse	2	sensoresReactor. ServerSensores. main. 29	Cerrar el recurso BufferedReader.	No procede.
6	UNUSED_LOCAL_ VARIABLE	La variable local DatosSensores no es usada.	2	sensoresReactor. ServerSensores. main. 43	Eliminar la variable DatosSensores en el main.	Evitar la declaración de variables locales para posteriormente no usarlas.

Conclusiones: Análisis del desarrollo 1

Para una clase relativamente pequeña, entre todas las herramientas hemos encontrado un total de 12 bugs (algunos de ellos duplicados). Esto nos parece algo bastante preocupante, pues demuestra que el código es bastante mejorable, y también detecta algunos errores potenciales como es el caso del error 2 encontrado con SpotBugs, que tenía que ver con el tipo de codificación que se usaba. Esto puede provocar un fallo del sistema en algún punto en el futuro.

Análisis del desarrollo 2

En este desarrollo, encontramos muchas más clases que en el anterior desarrollo, por lo que esperamos encontrar más errores. Hemos analizado el código con las herramientas anteriormente mencionadas, y hemos obtenido los siguientes resultados:

(NOTA: Debido al gran tamaño de las tablas, se incluyen en las siguientes páginas apaisadas)

Análisis con SpotBugs

En la siguiente tabla pueden verse los resultados:

Leyenda de la tabla:

Color de celda naranja, blanco: bug
Color de celda rojo: Falso positivo

Nº	Vulnerabilidad de Seguridad por errores software	Explicación del código de error	Prio.	Paquete.clase. funcion. línea código	Sanitización Propuesta	Comentarios
1	SS_SHOULD_BE_STATIC	Esta clase contiene un campo final que no está declarado como static.	2	SSII. IntegrityHIDS. ConfigurationFile. 45	Declarar la propiedad “DAEMON_SECTION” como static	No procede.
2	SS_SHOULD_BE_STATIC	Esta clase contiene un campo final que no está declarado como static	2	SSII. IntegrityHIDS. ConfigurationFile. 46	Declarar la propiedad “ALGORITHM_KEY” como static	No procede.
3	SS_SHOULD_BE_STATIC	Esta clase contiene un campo final que no está declarado como static	2	SSII. IntegrityHIDS. ConfigurationFile. 50	Declarar la propiedad “FILES_SECTION” como static	No procede.

4	SS_SHOULD_BE_STATIC	Esta clase contiene un campo final que no está declarado como static	2	SSII. IntegrityHIDS. ConfigurationFile. 48	Declarar la propiedad “FILE_KEY” como static	No procede.
5	SS_SHOULD_BE_STATIC	Esta clase contiene un campo final que no está declarado como static	2	SSII. IntegrityHIDS. ConfigurationFile. 51	Declarar la propiedad “HASHFILE_KEY” como static	No procede.
6	SS_SHOULD_BE_STATIC	Esta clase contiene un campo final que no está declarado como static	2	SSII. IntegrityHIDS. ConfigurationFile. 52	Declarar la propiedad “INCIDENTSFILE_KEY” como static	No procede.
7	SS_SHOULD_BE_STATIC	Esta clase contiene un campo final que no está declarado como static	2	SSII. IntegrityHIDS. ConfigurationFile. 53	Declarar la propiedad “INDICATORSFILE_KEY” como static	No procede.
8	SS_SHOULD_BE_STATIC	Esta clase contiene un campo final que no está declarado como static	2	SSII. IntegrityHIDS. ConfigurationFile. 47	Declarar la propiedad “PERIOD_KEY” como static	No procede.
9	UUF_UNUSED_FIELD	Este campo no se usa nunca.	2	SSII. IntegrityHIDS. ConfigurationFile. 55	Eliminar la propiedad “FilePath”	Esta propiedad sólo se usa dentro de la clase ConfigurationFile en su método setter, además de un uso en el método ConfigurationFile, donde realmente la propiedad no haría falta, pues al método se le pasa un filePath como parámetro, por lo que determinamos que la propiedad no se usa nunca y puede ser eliminada.

10	UUF_UNUSED_FIELD	Este campo no se usa nunca	2	SSII. IntegrityHIDS. ConfigurationFile. 56	Eliminar la propiedad “iniFileReader”	Esta propiedad se usa varias veces en muchos métodos de la clase y es necesario que el valor de ésta permanezca guardado, por lo que creemos que se trata de un falso positivo.
11	ICAST_INTEGER_MULTIPLY_CAST_TO_LONG	El código realiza una multiplicación en Integer y luego convierte el resultado a Long.	2	SSII. IntegrityHIDS. DaemonExecution Mode. executeDaemonOptions.137	Convertir los parámetros integer a long antes de realizar la multiplicación	Thread.sleep(1000 * 60 * options.getMinutesBetweenIntegrityChecks()); El bug se encuentra en el método Thread.sleep, porque, además de otros números, se le está pasando un método “options.getMinutesBetweenIntegrityChecks()”, que devuelve un int como resultado. Para eliminar el bug se puede guardar el resultado en una variable Long y luego realizar el thread.sleep con dicha variable, pero la funcionalidad será la misma, sólo se elimina el reconocimiento del bug.
12	SBSC_USE_STRINGBUFFER_CONCATENATION	El método va concatenando a un String en bucle. Cuando se hace esto, el String se convierte a StringBuffer, se concatena el valor, y se vuelve a convertir a String. Por ello es mejor usar directamente StringBuffer.	2	SSII. IntegrityHIDS. FileHashingUtils. byteArrayToHexString. 83	Cambiar el tipo de la propiedad “rval” a StringBuffer para concatenar valores de manera iterativa sobre él.	No procede.
13	SBSC_USE_STRINGBUFFER_CONCATENATION	El método va concatenando a un String en bucle. Cuando se hace esto, el String se convierte a StringBuffer, se concatena el valor, y se vuelve a convertir a String.	2	SSII. IntegrityHIDS. reporting. IntegrityCheck.toString. 234	Cambiar el tipo de la propiedad “rval” a StringBuffer para concatenar valores de manera iterativa sobre él.	No procede.

Análisis con Codacy

En la siguiente tabla pueden verse los resultados:

Nº	Explicación del código de error	Prioridad	clase. línea código	Sanitización Propuesta	Comentarios
1	Esta clase lanza un error RuntimeException	1	ConfigurationFile.243 ConfigurationFile.269 ConfigurationFile.256	Evite lanzar ciertos tipos de excepciones. En lugar de lanzar una RuntimeException, Throwable, Exception o Error sin procesar, utilice una excepción o error subclasificado.	Cuando vaya a lanzar un error utilice exactamente el error específico que se lanzará.
2	Se declaran dos variables en la misma línea	2	DaemonExecutionMode.327	Java permite el uso de varias variables de declaración del mismo tipo en una línea. Sin embargo, puede conducir a un código bastante desordenado. Esta regla busca varias declaraciones en la misma línea.	No procede
3	Esta clase lanza un error RuntimeException	1	FileHashingUtils.47 FileHashingUtils.57	Evite lanzar ciertos tipos de excepciones. En lugar de lanzar una RuntimeException, Throwable, Exception o Error sin procesar, utilice una excepción o error subclasificado.	Cuando vaya a lanzar un error utilice exactamente el error específico que se lanzará.
4	Instruccion switch sin valor default	2	Options.167 Options.304	Todos los switch deben incluir una opción default para capturar cualquier valor no especificado.	No procede

5	Campos declarados en mitad de una clase	2	Options.323 Options.324 Options.326 Options.327 Options.329	Los campos deben declararse en la parte superior de la clase, antes de cualquier declaración de método, constructores, inicializadores o clases internas.	No procede
6	El método 'checkContentKeys ()' tiene una complejidad NPath de 480	1	ConfigurationFile.99	Dividir el método checkContentKeys() en varios submetodos con el fin de reducir la complejidad de este	La complejidad NPath de un método es el número de rutas de ejecución acíclicas a través de ese método. Mientras que la complejidad ciclomática cuenta el número de puntos de decisión en un método, NPath cuenta el número de rutas completas desde el principio hasta el final del bloque del método. Esa métrica crece exponencialmente, ya que multiplica la complejidad de las declaraciones en el mismo bloque. Para obtener más detalles sobre el cálculo, consulte la documentación de la métrica NPath. Un umbral de 200 generalmente se considera el punto donde se deben tomar medidas para reducir la complejidad y aumentar la legibilidad.
7	Esta clase lanza un error RuntimeException	1	Options.313	Evite lanzar ciertos tipos de excepciones. En lugar de lanzar una RuntimeException, Throwable, Exception o Error sin procesar, utilice una excepción o error subclasificado.	Cuando vaya a lanzar un error utilice exactamente el error específico que se lanzará.
8	Campos declarados en mitad de una clase	2	Options.325 Options.328	Los campos deben declararse en la parte superior de la clase, antes de cualquier declaración de método, constructores, inicializadores o clases internas.	No procede

Análisis con PMD

En la siguiente tabla pueden verse los resultados:

Nº	Vulnerabilidad de Seguridad por errores software	Explicación del código de error	Prioridad	Paquete.clase. funcion. línea código	Sanitización Propuesta	Comentarios
1	FINAL_FIELD_COULD_ BE_STATIC	El atributo final no ha sido declarado estático	2	SSII. IntegrityHIDS. ConfigurationFile. 52	Declarar la propiedad "INCIDENTSFILE_KEY" como static	No procede.
2	AVOID_THROWING_ RAW_EXCEPTION_ TYPES	Lanzamiento no deseado de Raw Exception	2	SSII. IntegrityHIDS. ConfigurationFile. 230	Evitar lanzar Raw Exception	No procede.
3	PREMATURE_ DECLARATION	Variable declarada antes de ser referenciada ante un posible punto de salida.	2	SSII. IntegrityHIDS. ConfigurationFile. 258	Declarar la variable filesSection en su debido momento.	No procede.
4	AT_LEAST_ CONSTRUCTOR	Ningún constructor declarado en toda la clase.	2	SSII. IntegrityHIDS. DaemonExecutionMode. 38	Declarar algún constructor dentro de la clase.	No procede.
5	USE_UTILITY_ CLASS	No se usa una clase utility siendo todos los métodos static.	2	SSII. IntegrityHIDS. FileHashingUtil. 15	Añadir una clase utility.	Se podría añadir también un constuctor privado para corregir este problema.

6	SWITCH_STMTS_ SHOULD_HAVE_ DEFAULT	La declaración switch debería tener un caso default.	2	SSII. IntegrityHIDS. Options. 316	Añadir un default después de los case.	El default es recomendable por si fallan todos los case anteriores, que el algoritmo no se detenga o falle.

Conclusiones: Análisis del desarrollo 2

En este análisis, hemos encontrado bastantes mas bugs y errores potenciales, además de un falso positivo en una de las herramientas. Es cierto que este desarrollo era mucho más grande que el anterior, tanto en número de archivos como en la longitud de los mismos, pero hemos encontrado algunos errores importantes que podrían hacer que el sistema dejase de funcionar correctamente ante un mal uso inesperado (como el switch que no tiene valor por defecto que encontraron Codacy y PMD), además de otros errores de estilo como la declaración de variables en medio de una clase en vez de al inicio, entre otras.

Conclusiones generales

Aunque algunos de los errores sean, por ejemplo, cuestiones meramente estéticas, es importante corregir todos estos errores, pues al final un buen código se define por su buen funcionamiento tanto en el caso esperado como en el inesperado, la eficiencia y la legibilidad del mismo.

Por ello, creemos que es muy importante realizar un análisis periódico del código que se escribe, pues, si se deja para el futuro, cuando la aplicación se encuentre terminada, será muchísimo más costoso de identificar y de arreglar.

Además, es interesante destacar la importancia que tienen las herramientas utilizadas, y es que no existe una “herramienta perfecta” que detecte todos los errores, como se puede ver en nuestro análisis: Muchas herramientas encontraban errores duplicados, pero otras encontraron bugs que otras no. Por ello, creemos que es importante que la empresa tenga estos aspectos en cuenta en el futuro, y se plantee realizar revisiones periódicas del mismo. Por último, los desarrolladores deben estudiar los errores encontrados, comprenderlos, y asegurarse de no repetirlos en el futuro.