uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática
2017-2018

Trabajo Fin de Grado

"Evaluación de la Seguridad de un Sistema desarrollado"

Radu Nicolae Zaicanu

Tutor

Ana Isabel González-Tablas Ferreres

Leganés, 2018

RESUMEN

La continua evolución de la tecnología, sumada al creciente tamaño y complejidad de los sistemas software, ha llevado el desarrollo de productos software a un punto en el que es complicado asegurar la fiabilidad de este. Como consecuencia de esta tendencia, la seguridad del software se ha convertido en un requisito imprescindible en el proceso de desarrollo, llegando incluso a ser un factor importante a la hora de determinar si el código de un sistema software es de calidad.

A lo largo de este documento se analizará una de las posibles técnicas que pueden aplicarse para garantizar, dentro de lo posible, la seguridad y por tanto la calidad de un sistema software. Así mismo, se proporcionará material didáctico suficiente para empezar a utilizar una de las herramientas que aplican esta técnica conocida como análisis estático de código fuente.

Palabras clave

Software; Seguridad; Calidad; Análisis estático de código

DEDICATORIA

Alguien dijo una vez que la vida es una serie de habitaciones y aquellos con quienes coincidimos en ellas configuran nuestras vidas. Con esta analogía bien presente, y habiendo llegado el momento de cerrar este capítulo de mi vida, me doy cuenta de la gran importancia que ha tenido esta etapa tanto en mi vida personal como en mi carrera profesional.

No existen palabras que puedan expresar mi agradecimiento a todos los que me han acompañado y ayudado durante este viaje, por lo que intentaré hacerles justicia con las que sí existen.

Con las únicas palabras que he sido capaz de encontrar, quiero dedicar este trabajo a mi familia, quien ha estado a mi lado siempre y se ha sacrificado por mí tantas veces, a los compañeros de carrera con los que he compartido tantas experiencias, a la tutora que me ha soportado en la recta final de este viaje y a todos los profesores que he tenido la suerte de conocer.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. Motivación del Trabajo

La economía actual depende en gran medida de los sistemas y redes informáticas, y muchos servicios, como el de las finanzas, el comercio electrónico, el transporte, la energía y la atención sanitaria, no pueden funcionar sin estos sistemas. El crecimiento del comercio online, las transacciones conexas, así como el volumen cada vez mayor de información confidencial accesible online, han provocado un crecimiento de los ataques cibernéticos.

En la actualidad, para desarrollar un sistema se aplica el ciclo SDLC o Ciclo de Vida del Desarrollo de Sistemas. Este método posibilita el desarrollo de un SI o Sistema de Información, ofreciendo un soporte para que los procesos de diseño y puesta en marcha se puedan planificar por el gestor del proyecto, concretando el tiempo de inversión del presupuesto y el tiempo de desarrollo. A pesar de ser un modelo que abarca el ciclo de vida de un sistema, desde que se plantea la idea hasta que se decide su caducidad, no incluye un ciclo específico para la seguridad.

Teniendo en cuenta estos hechos, parece más que evidente la necesidad urgente de tener en cuenta la seguridad de cualquier producto o sistema que está en desarrollo. Esta seguridad debe aplicarse durante el propio desarrollo, y no tras la finalización de este, para tener la oportunidad, ante cualquier problema de seguridad, de aplicar una solución acorde a la gravedad de este.

1.2. Objetivos

Como se ha mencionado en el epígrafe anterior, la continua evolución de la tecnología supone un avance en la facilidad de uso de esta tecnología, pero también supone un incremento de los problemas de seguridad en la mayoría de los casos. Un pequeño fallo en la seguridad de un sistema podría provocar una fuga masiva de información sensible.

La propuesta descrita en este documento tiene como finalidad elaborar un material instructivo para proporcionar una solución viable y que cualquier usuario, que está desarrollando un sistema, evite, en la medida de lo posible, los problemas de seguridad presentes en el sistema. Estos problemas de seguridad incluyen desde malos usos del lenguaje de programación, pasando por los bugs, hasta las vulnerabilidades. Estas vulnerabilidades son la puerta de entrada para cualquier ataque cibernético, por lo que la propuesta descrita se centra en este tipo de problemas en particular.

Por lo tanto, uno de los objetivos de este trabajo es crear un documento en el que se proporcionen las instrucciones necesarias para conocer en detalle las características de una herramienta de análisis estático de código, su funcionamiento básico y su aplicación a un proyecto real. Además, se proporcionará la documentación necesaria para hacer un uso avanzado de las funcionalidades de este tipo de herramientas.

1.3. Estructura del documento

El presente documento está organizado según el formato, estructura y estilo recomendados. La estructura está compuesta de capítulos, dentro de los cuales se sitúan uno o varios epígrafes para facilitar el seguimiento del documento. Además, se incluyen varios anexos para especificar un glosario de términos, las respuestas de la autoevaluación y el resumen del documento en lengua inglesa. A continuación, se describe brevemente esta estructura:

- **Capítulo 1 – Introducción**. En este capítulo se proporciona una visión general de este trabajo, como puede ser la motivación, los objetivos que se esperan conseguir con este trabajo, la estructura del documento y los estándares técnicos o legislación vigente que se le puede aplicar.
- **Capítulo 2 – Estado de la cuestión**. En este capítulo se describen las razones por las que se debe llevar a cabo este trabajo haciendo un análisis del estado de la seguridad de los sistemas software, así como de las técnicas y herramientas que existen para controlar la seguridad.
- **Capítulo 3 – Análisis del problema**. En este capítulo se describen las razones por las que se ha elegido una solución en favor de otras, sus características y los objetivos que se espera conseguir tras aplicar esta solución.
- **Capítulo 4 – Diseño de la solución**. En este capítulo se especifica la estructura de la solución, los requisitos que hace falta satisfacer para implementarla y, además, se elabora todo el contenido de la solución.
- **Capítulo 5 – Evaluación**. En este capítulo se describe el método de evaluación que se ha aplicado y los resultados obtenidos en la evaluación del curso.
- **Capítulo 6 – Planificación y presupuesto**. En este capítulo se detalla la metodología seguida para elaborar el trabajo, así como el presupuesto necesario para la elaboración de este.
- **Capítulo 7 – Conclusiones**. En este capítulo se hace un análisis para saber si se han cumplido los objetivos fijados, posibles líneas futuras de trabajo y el impacto socio-económico que tiene este trabajo.
- **Bibliografía**. En este capítulo se detallan las referencias consultadas durante la redacción de este documento.

1.4. Marco Regulador

Existen diferentes normas y estándares técnicos, aplicables a este trabajo, que marcan la forma de utilizar algunas medidas apropiadas para la seguridad de un sistema y la protección de los datos de carácter sensible. A continuación, se detallan algunas de estas normas y estándares:

La familia de normas ISO/IEC 27000. Esta familia de normas, desarrollada por la Organización Internacional de Normalización en colaboración con el Comité Electrotécnico Internacional, ayuda a las organizaciones a mantener la información segura. El uso de estas normas facilita a la organización la gestión de la seguridad de la información, como puede ser la información financiera, la propiedad intelectual, los datos

de los empleados o la información de terceros. Por ejemplo, la norma ISO/IEC 27001, la más conocida a nivel internacional, establece los requisitos que debe cumplir un sistema que gestiona la seguridad de la información.

Aplicando este conjunto de normas al entorno de una herramienta de análisis estático de código evita que cualquier usuario que no sea el desarrollador o el gestor del proyecto tenga acceso a la administración de la herramienta y pueda saber las vulnerabilidades detectadas que aún no se han resuelto.

CISQ. El Consorcio para la Calidad del Software en las Tecnologías de la Información es un grupo de liderazgo que desarrolla estándares internacionales para automatizar el proceso de medición del tamaño y calidad de los sistemas software.

Estos estándares se utilizan para gestionar las propiedades de seguridad, confiabilidad, eficiencia del rendimiento y mantenibilidad del riesgo de un sistema software. El estándar de la seguridad se basa en las vulnerabilidades de seguridad más extendidas y explotadas en la actualidad, como las que aparecen en OWASP Top 10, CWE y SANS Top 25.

Aplicando estos estándares al entorno de una herramienta de análisis estático de código se mejora la detección de bugs, vulnerabilidades y malos hábitos de programación, así como se fortalecen los estándares de calidad del proyecto.

RFC 2196. Memorándum publicado por el Grupo de Trabajo de Ingeniería de Internet, más conocido como IETF, para desarrollar procedimiento y políticas de seguridad para los sistemas de información que están conectados a internet. Este documento ofrece una visión global de lo que es la seguridad de la red y de la información, la respuesta a posibles conflictos y las políticas de seguridad que se deben aplicar.

Aplicando estas políticas y procedimientos de seguridad al entorno de este trabajo se consigue una visión más real de las posibles situaciones que pueden ocurrir en un entorno de trabajo, como puede ser el procedimiento a seguir si el sistema tiene una brecha de seguridad o está bajo ataque cibernético.

2. ESTADO DE LA CUESTIÓN

2.1. Situación actual

La tecnología nunca ha sido más útil, ni más peligrosa, para nuestra información, identidad y activos, de lo que es en la actualidad. Esta tecnología es la base de todos los servicios del sector terciario y cuaternario de la economía y, por tanto, es el objetivo principal de todos los ataques cibernéticos que se producen en la actualidad. Servicios tan importantes como el de la información, los negocios, la atención médica, el gobierno, el ejército y la educación, se han visto comprometidos en los últimos años debido a brechas de seguridad en sus sistemas. Algunos de estos ejemplos son:

Información. En Septiembre de 2016, Yahoo! Inc. reveló que había sufrido una brecha de seguridad, en el año 2014, en la que se robaron más de 500 millones de cuentas de usuario. Un mes más tarde, en Diciembre de 2014, esta misma compañía reveló que habían sufrido otra brecha de seguridad, en Agosto de 2013, en la que quedaron expuestas alrededor de mil millones de cuentas de usuario [1]. Aprovechando esta brecha en la seguridad, los atacantes consiguieron datos confidenciales como nombres de usuario, direcciones de correo electrónico, números de teléfono, contraseñas sin cifrar, preguntas y respuestas de seguridad en claro.

Negocios. La empresa Equifax, dedicada a recopilar y gestionar informes de crédito al consumidor, informó, en septiembre de 2017, a la Comisión de Valores y Bolsa de Estados Unidos [2] del alcance del ataque cibernético que habían sufrido en mayo de ese mismo año. Según este informe, los atacantes robaron datos sensibles de cerca de 150 millones de consumidores estadounidenses y 16 millones de consumidores británicos [3]. Entre estos datos se incluyen el nombre, la fecha de nacimiento, el NSS, la dirección postal y los datos bancarios.

Atención médica. La segunda compañía de seguros de salud más grande de Estados Unidos, Anthem Insurance Companies, vio comprometidos los datos de aproximadamente 80 millones de sus clientes en febrero de 2015 [4]. Siete comisionados estatales de seguros elaboraron un informe [5], en diciembre de 2016, en el que se detalla la causa de la brecha de seguridad y la gravedad del ataque, en términos de datos confidenciales robados. Estos comisionados llegaron a un acuerdo con la aseguradora en el que se instaba a la compañía a invertir significativamente en la seguridad. A raíz de este problema, la compañía ha invertido más de 260 millones de dólares en medidas de seguridad [6].

Gobierno. En junio de 2015, la Oficina de Administración de Personal de Estados Unidos anunció que había sido víctima de una fuga masiva de información, la cual involucraba a 4 millones de empleados gubernamentales [7]. Una fuente interna de la policía estadounidense relató a un medio de comunicación [8] que el gobierno tenía razón para creer que el ataque provenía de una entidad o gobierno extranjero.

Educación. Según la información publicada en el periódico Huffington Post [9], en marzo de 2014 la Universidad de Maryland sufrió el robo de más de 300 mil registros de estudiantes, profesores y empleados [9]. De acuerdo con el Centro de Intercambio de información sobre Derechos de Privacidad [10], desde el año 2005 se han producido 854 brechas de seguridad y más de 25 millones de registros con datos robados solo en el servicio educativo de Estados Unidos.

Estos son solo algunos de los muchos ejemplos que se han dado en los últimos años, puesto que, según el Centro de Recursos para el Robo de Identidad [11], en los últimos cuatro años se han producido más de 9395 ataques a organizaciones de estos sectores, llegando a perder alrededor de 1 billón de registros con información confidencial.

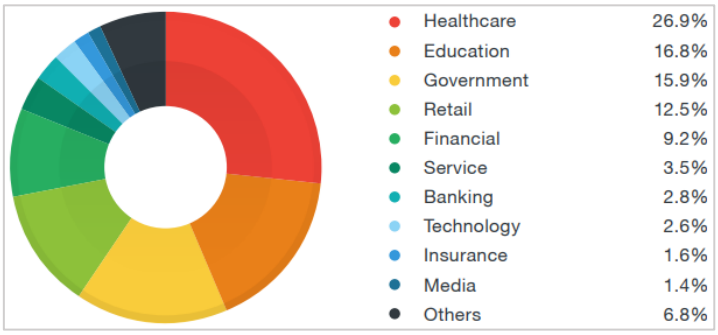

Fig. 1. Servicios afectados por la fuga de datos [12]

En la Figura 1 se observa el estudio realizado por el equipo de Trend Micro en relación con la fuga de datos, en base al tipo de servicio, que se ha producido entre los años 2005 y 2015. Como puede apreciarse, los servicios más afectados por los ataques cibernéticos son la atención médica con un 26.9% de los ataques, la educación con un 16.8% y el gobierno con un 15.9%.

El Ponemon Institute [13], en colaboración con Accenture, realiza un informe anual en el que evalúan las respuestas de 2182 entrevistados de 254 compañías en siete países, a saber: Australia, Francia, Alemania, Italia, Japón, Reino Unido y Estados Unidos. Con este informe se pretende dar a conocer el impacto que tiene el crimen cibernético en un país o sector y como mantenerse alejado de estas amenazas cibernéticas.

Según el último informe, del año 2017, el coste medio anual del delito cibernético para estas 254 compañías era de 11.7 millones de dólares, con un promedio de 50 días para contener un ataque. Además, se calcula que el número de ataques aumenta de media cada año un 27.4% y, como consecuencia, el coste de la seguridad ha aumentado un 22.7% en el último año. Este informe también revela que los servicios financieros tienen el coste de seguridad más elevado, a pesar de que el servicio de atención médica es el que más ataques recibe.

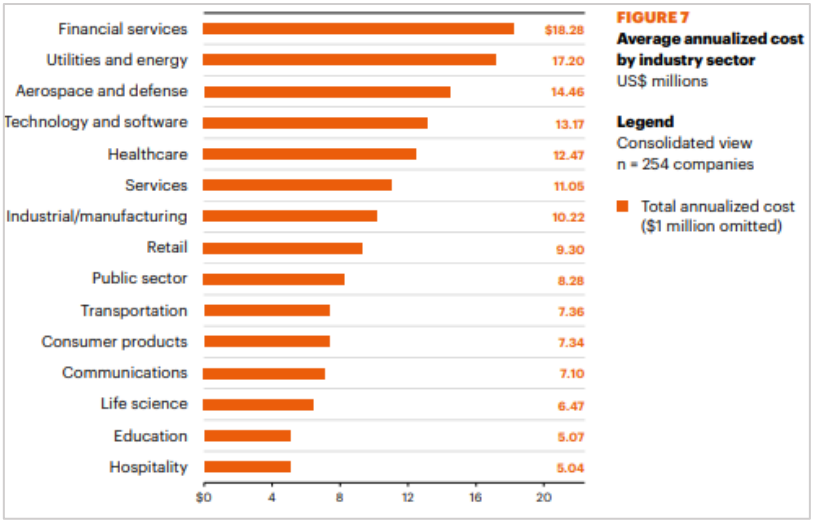

Fig. 2. Coste anual medio por tipo de servicio [13]

En la Figura 2 se observa el coste anual medio del crimen cibernético organizado por el tipo de servicio, es decir, servicios financieros, atención médica, gobierno, etc. Así mismo, se puede apreciar que los servicios financieros son los que más se gastan debido al crimen cibernético, unos 18 millones de dólares de media, seguidos de cerca por el servicio energético, con unos 17 millones de media, y en el quinto lugar se encuentra la atención médica con unos 12 millones, a pesar de ser el servicio más atacado.

Las deficiencias en la calidad del software son la primera razón de las vulnerabilidades de seguridad de un sistema. Una vulnerabilidad se define como una propiedad de los requisitos de seguridad del sistema, el diseño, la implementación o la ejecución y que una vez activada accidental o intencionadamente da lugar a un fallo de seguridad [14]. Básicamente, si se ha producido un fallo de seguridad es porque debe haber habido una vulnerabilidad.

La empresa Tricentis, dedicada a hacer pruebas de software, elabora un análisis anual de los errores de software que han sido mencionados en los artículos de noticias [15]. En el año 2017, se detectaron 606 errores de software en 1177 artículos y 314 compañías. Entre los resultados de este análisis destaca la clasificación que se hace de los errores encontrados y la medida en que estos errores están presentes en los diferentes servicios.

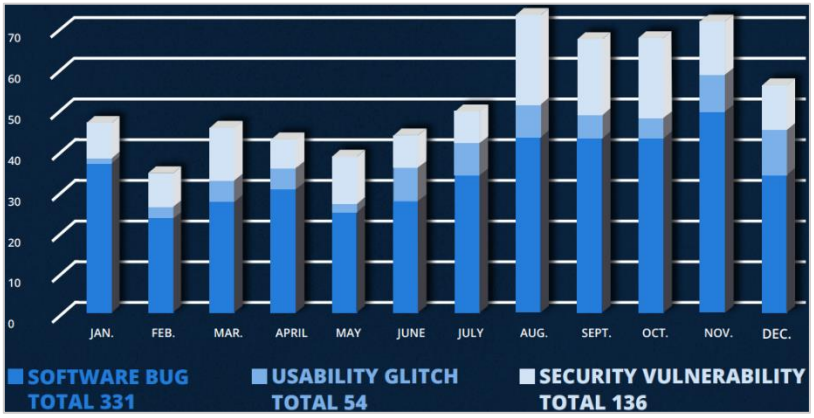

Fig. 3. Clasificación de los errores de software [15]

En la Figura 3 se puede apreciar la clasificación que hace la empresa Tricentis de los errores encontrados durante los meses del año 2017 y el porcentaje de cada uno de estos errores en cada uno de los meses.

Los errores encontrados durante ese año se pueden clasificar en una de estas tres categorías:

- El primer tipo de error, y el más común, es un **bug**, es decir, una instancia en la que un sistema software no funciona según lo previsto. Durante el año 2017, se encontraron un total de 331 bugs.
- El segundo tipo es una **vulnerabilidad**, es decir, un defecto que puede ser explotado por lo atacantes para alterar el comportamiento normal de un sistema o robar datos. Durante el año 2017, se encontraron un total de 136 vulnerabilidades.
- El tercer tipo es **fallo de usabilidad**, es decir, un defecto en el diseño del software que dificulta el uso del producto o sistema y que, en ocasiones, puede llegar a inutilizarlo. Durante el año 2017, se encontraron un total de 54 fallos de usabilidad.

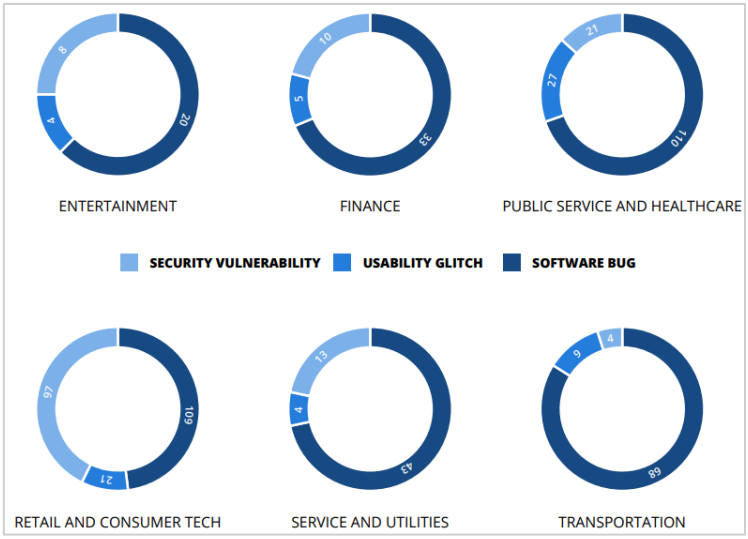

Fig. 4. Tipos de errores software en cada servicio [15]

En la Figura 4 se puede observar el número total de errores de cada tipo encontrados en cada servicio. Como se puede advertir, el error de software más frecuente en todos los servicios son los bugs, seguidos por las vulnerabilidades y, en menor medida, los fallos de usabilidad. Los servicios con más bugs son los de la atención médica y los servicios públicos, datos que coinciden con los mencionados anteriormente sobre qué servicios eran los más atacados por los delincuentes cibernéticos.

La mayoría de las metodologías tradicionales de desarrollo de software no incluyen explícitamente un método estandarizado para incorporar la seguridad de la información en sus ciclos de vida (SDLC), o bien lo incluyen en la etapa de pruebas [16]. Este hecho incluye en el coste de solucionar estos errores, pues cuanto más tarde se detectan más caro resultan de solucionar.

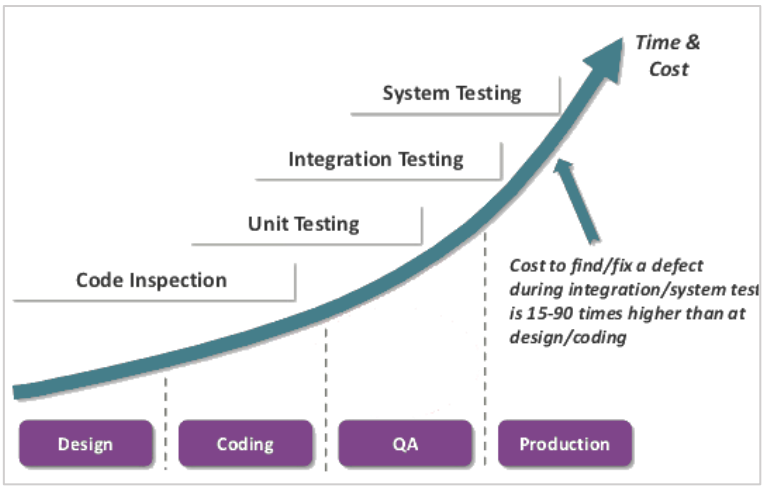

Fig. 5. Coste de solucionar un error en cada fase del SDLC [17]

En la Figura 5 se observa un gráfico en el que se pone de manifiesto el coste que supone arreglar un error en cada una de las fases de desarrollo de un sistema software. El coste de arreglar el error en la última fase es de 15 a 90 veces más grande que si se arreglase en las primeras dos fases.

Debido al incremento de los ataques cibernéticos y al elevado coste que supone poner parches a las brechas de seguridad, tanto el sector público como el privado están incorporando la seguridad a lo largo de todas las etapas del SDLC [16]. Por lo tanto, se está convirtiendo en un imperativo tener en cuenta la seguridad durante el diseño y desarrollo de los sistemas software, así como ampliar la capacidad de verificación y validación para cubrir todas las preocupaciones en cuanto a la seguridad de la información [18].

Para incorporar la seguridad durante el SLC se requiere utilizar una variedad de técnicas de prevención y descubrimiento de vulnerabilidades [19]. Una de estas técnicas de descubrimiento de vulnerabilidades es el análisis estático de código fuente. Esta técnica proporciona una forma escalable para la revisión del código, se puede aplicar al principio del ciclo de vida, no requiere que el sistema esté terminado y puede aplicarse a partes del código individualmente [20].

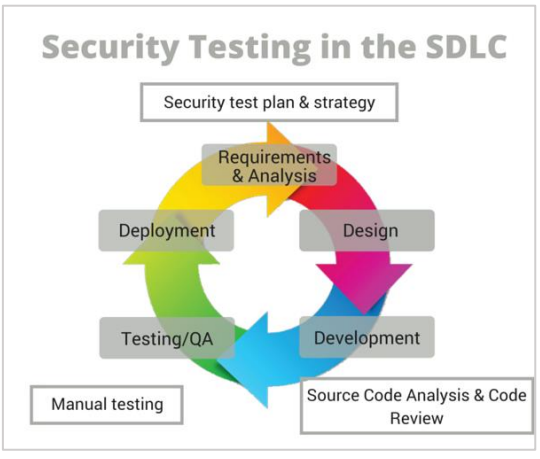

Fig. 6. Integración de la seguridad en el SDLC [17]

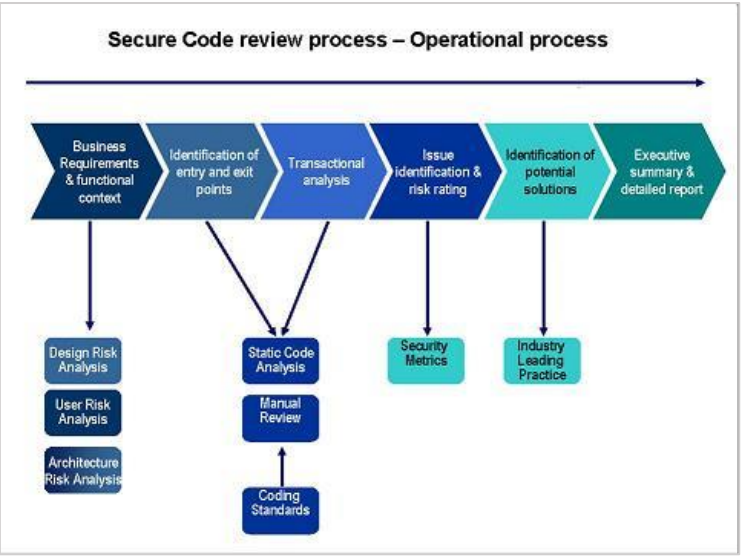

Fig. 7. Integración de la seguridad en el SDLC [21]

En la Figura 6 se puede observar la aplicación de diferentes técnicas de prevención y descubrimiento de vulnerabilidades al SDLC, entre las que está el análisis estático de código fuente.

2.2. Marco actual

Las herramientas de análisis de código fuente, también conocidas como herramientas de pruebas estáticas de seguridad de aplicaciones (SAST), están diseñadas para analizar el código fuente, o una versión compilada de ese código, y ayudar a encontrar fallos de seguridad.

Las SAST han evolucionado rápidamente en la última década, desde el simple análisis léxico hasta emplear técnicas mucho más complejas. Sin embargo, en general, los problemas de análisis estático dependen de la complejidad de los errores de seguridad y del conocimiento que tiene el autor del código.

Por lo tanto, las SAST no detectan todas las vulnerabilidades en el código fuente (también conocidos como falsos negativos) y son propensas a reportar fallos que al examinarlos más de cerca resultan no ser vulnerabilidades de seguridad (también conocidos como falsos positivos). Para ser de utilidad práctica, una herramienta de análisis de código estático debe encontrar tantas vulnerabilidades como sea posible y con una cantidad mínima de falsos positivos.

Al igual que ocurre con cualquier otra tecnología, esta herramienta tiene ventajas y también tiene inconvenientes y más cuando se aplica en desarrollo de un proyecto.

Las ventajas de las SAST son:

- **Buena escalabilidad**. Están diseñadas para ser usadas en entornos de integración continua donde el código y el proyecto en sí se modifica con frecuencia.
- **Útil para problemas conocidos**. Son muy útiles, especialmente cuando se trata de encontrar fallos frecuentes que no dejan lugar a duda, como puede ser el desbordamiento del búfer, inyección SQL, etc.

- **Buen feedback para los desarrolladores**. La información proporcionada al desarrollador, en relación con el fallo, es muy precisa. Esta información comprende desde la localización exacta del fallo, pasando por la gravedad de este e incluso las posibles soluciones.

Las desventajas de las SAST son:

- **Tasa elevada de falsos positivos**. La complejidad del código y la facilidad para detectar el fallo son factores determinantes para obtener una tasa reducida de falsos positivos.
- **Dificultad para comprobar si un fallo es una vulnerabilidad**. Es complicado demostrar que el fallo encontrado es una vulnerabilidad si esta no es frecuente.
- **Dificultad para encontrar fallos de configuración**. Si la configuración no está presente en el código, la herramienta no puede detectar si existen problemas en la misma.

En la actualidad, existen una gran variedad de SAST en el mercado. Algunas de ellas son de código abierto, aunque las más completas se distribuyen bajo licencia comercial.

A continuación, se detallan algunas de las SAST que hay disponibles en el mercado:

CA Veracode Static Analysis

CA Veracode Static Analysis es una herramienta de análisis estático que se basa en el modelo SaaS (Software como un Servicio), es decir, sus servicios están ubicados en la nube. Esta herramienta permite ejecutar el escáner desde la misma plataforma o desde el IDE utilizado (Entorno de Desarrollo Integrado). Así mismo, se encarga de encontrar los fallos de seguridad y recomendar una posible solución. Además, permite elaborar informes sobre la calidad del código fuente en función de diferentes métricas, aunque la principal es la seguridad. Soporta cerca de 20 lenguajes de programación y se distribuye bajo una licencia comercial.

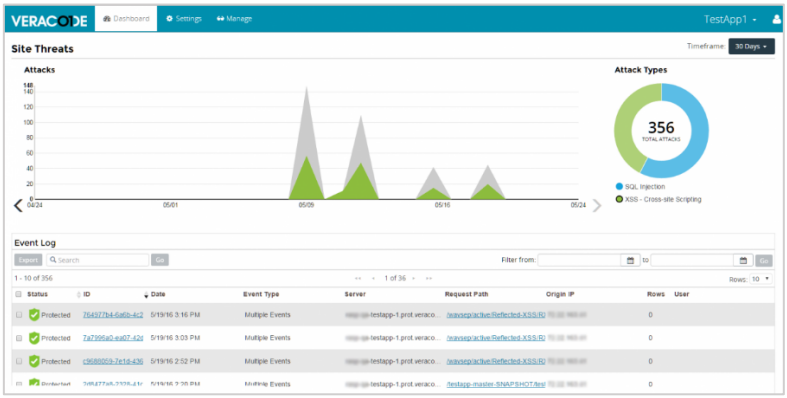

Fig. 8. Veracode Static Analysis [21]

Kiuwan

Kiuwan es una herramienta de análisis estático que se centra en identificar y corregir los fallos de seguridad de manera rápida y eficiente, integrándose completamente en el SDLC del proyecto. Esta herramienta cumple con los estándares de seguridad más conocidos,

como puede ser OWASP, CWE, MISRA o CERT. Además, proporciona informes y cuadros de mando con información sobre la seguridad. Así mismo, permite clasificar los proyectos por su nivel de importancia y criticidad. Soporta cerca de 35 lenguajes de programación y se distribuye bajo una licencia comercial.

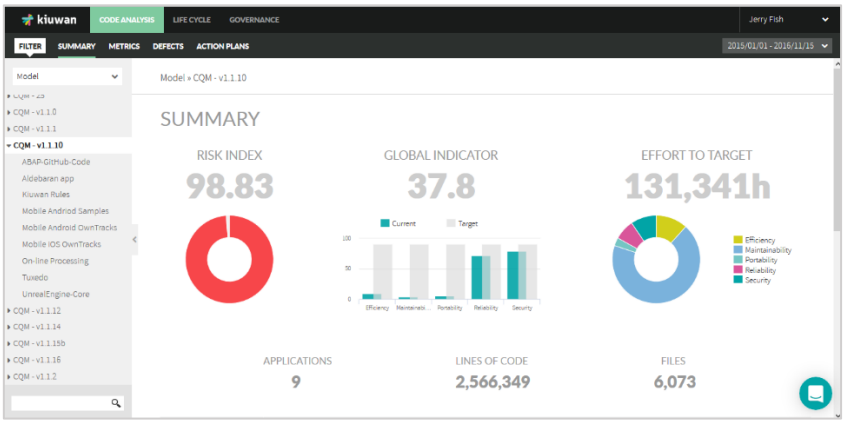

Fig. 9. Kiuwan [22]

Fortify Static Code Analyzer

Fortify Static Code Analyzer es una herramienta de análisis estático que se centra en identificar las vulnerabilidades que representan una mayor amenaza, identificar su causa y priorizar los resultados para que los desarrolladores puedan resolver los problemas más importantes en primer lugar. Esta herramienta permite a los desarrolladores y administradores trabajar juntos en la revisión del código y tomar medidas inmediatas para resolver las vulnerabilidades, reduciendo de esta manera el riesgo y la exposición. Soporta cerca de 25 lenguajes de programación y, al igual que las anteriores, se distribuye bajo una licencia comercial.

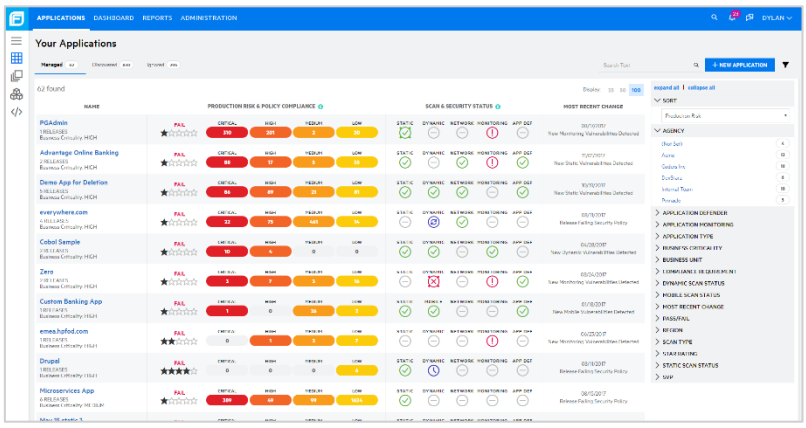

Fig. 10. Fortify Static Code Analyzer [23]

3. ANÁLISIS DEL PROBLEMA

3.1. Descripción

En el capítulo anterior se ha descrito el principal problema (ataques cibernéticos) que tiene la tecnología actualmente y con el que tiene que lidiar cualquier desarrollador de sistemas software. Así mismo, se ha descrito una de las técnicas con las que cuentan estos desarrolladores (SAST), sus ventajas y sus desventajas.

Tras un análisis exhaustivo de las posibles soluciones que se pueden llevar a cabo, se ha optado por realizar un curso, imitando en lo posible la estructura de un MOOC, en el que se enseña cómo utilizar una de estas herramientas de análisis estático de código fuente. Cabe mencionar, sin embargo, que este curso no incluye todos los componentes (vídeos, lecturas, cuestionarios) disponibles en un MOOC y tampoco está destinado a tener un público masivo.

Para que el contenido de este curso sirva realmente de guía práctica, se ha decidido enfocar el uso de la herramienta sobre un lenguaje concreto (Python) por las siguientes razones:

- La revista IEEE Spectrum [24] elabora un ranking anual en el que clasifican los lenguajes de programación en base a cuál es el más utilizado y que tipo de aplicaciones se pueden desarrollar con ese lenguaje (aplicaciones web, móviles, escritorio o empotrado). Este año el lenguaje más utilizado por los desarrolladores es Python.

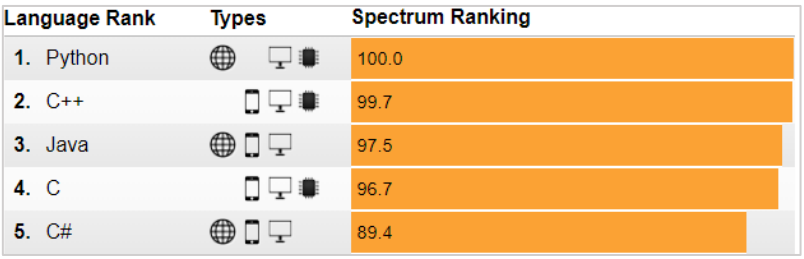

Fig. 11. IEEE Spectrum Top 2018 [25]

- La comunidad de programación TIOBE [25] mantiene un índice en el que clasifican los lenguajes de programación en base a su popularidad. Este mismo año, Python ha entrado en el Top 3 por primera vez desde que se elabora este índice.

Sep 2018	Sep 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.436%	+4.75%
2	2		C	15.447%	+8.06%
3	5	^	Python	7.653%	+4.67%
4	3	v	C++	7.394%	+1.83%
5	8	^	Visual Basic .NET	5.308%	+3.33%

Fig. 12. TIOBE Index para Septiembre 2018 [25]

- Al igual que otros lenguajes de programación, Python es susceptible de tener bugs y vulnerabilidades, por lo que un sistema software basado en este lenguaje podría ser víctima de un ataque cibernético.
- Actualmente existe poco material en la red que sirva de ayuda para realizar un buen análisis estático de código Python, a diferencia de otros lenguajes.

Teniendo en cuenta estos factores, se ha decidido que enfocar el curso sobre el lenguaje Python sería una contribución mayor que hacerlo para cualquier otro lenguaje.

La herramienta que se va a utilizar en este curso se denomina SonarQube y está desarrollada por la compañía SonarSource. Esta herramienta, a diferencia de las mencionadas en el capítulo anterior, cuenta con una versión de código abierto (Community Edition) y además dispone de tres versiones bajo licencia comercial. En el siguiente epígrafe se describirá más en detalle las características de esta herramienta.

3.2. Requisitos

SonarQube es una herramienta de código abierto desarrollada por SonarSource con el objetivo de tener una inspección continua de la calidad del código. Esta inspección continuada se realiza mediante revisiones automáticas en las que se analiza de manera estática el código para detectar bugs, vulnerabilidades y malas prácticas de programación, denominadas code smells.

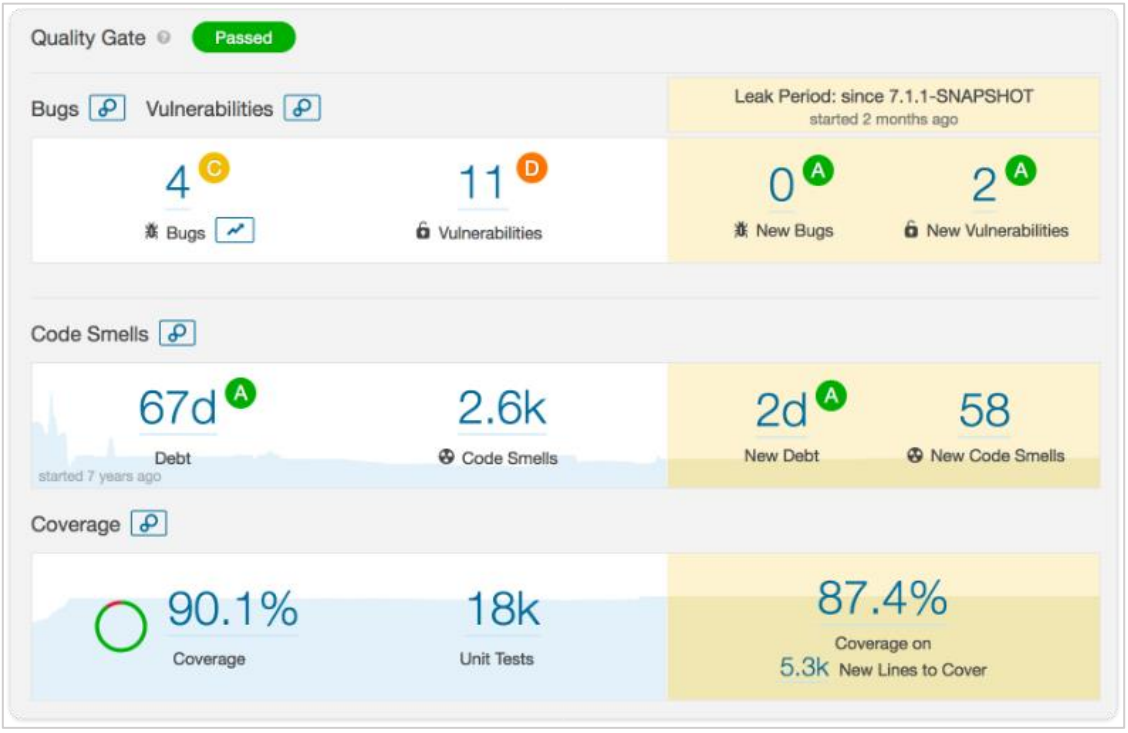

Fig. 13. SonarQube [26]

Esta herramienta ofrece la posibilidad de generar diversos informes sobre la calidad del código, como puede ser el porcentaje de código que está duplicado, las pruebas unitarias, que porcentaje del código está cubierto por el análisis, la complejidad del código, etc.

A continuación, se describen las características más destacadas de esta herramienta:

- **Inspección continua**. Además de proporcionar datos sobre la calidad del código, y por tanto del proyecto, SonarQube también resalta los problemas (bugs, vulnerabilidades, code smells) introducidos recientemente.
- **Detección de problemas complicados**. Haciendo uso de varios motores de flujo de datos sensibles a las rutas, SonarQube es capaz de explorar todas las rutas de ejecución posibles para detectar cualquier posible error, algo que sería prácticamente imposible de hacer manualmente.
- **Personalizar los parámetros de calidad**. SonarQube permite a los desarrolladores fijar los parámetros de calidad de cualquier proyecto según lo que estimen oportuno. Además, permite crear y habilitar nuevas reglas de análisis, deshabilitar las existentes, y crear perfiles de calidad escogiendo un conjunto de reglas en particular.
- **Soporte a multitud de lenguajes de programación**. Ofrece soporte para más de veinte lenguajes de programación, entre los que se encuentra Python, Java, C, C#, Javascript, etc.
- **Integración de otras herramientas**. SonarQube integra en su propia interfaz las reglas de calidad/seguridad que proporcionan otras herramientas de análisis estático de código, como puede ser FindBugs para Java y Pylint para Python. Esta integración perfecciona la búsqueda de bugs y vulnerabilidades mediante un análisis más exhaustivo.
- **Integración de DevOps**. Para algunos lenguajes dinámicos como Python o PHP basta con pasar a SonarQube los ficheros del proyecto, pero en caso de lenguajes como Java o C es necesario integrar esta herramienta en Maven, Ant o MSBuild. Además, SonarQube permite programar la ejecución de un análisis desde cualquier motor de integración continua, como Jenkins, Azure DevOps, etc**.**
- **Comunidad abierta y entregada**. La comunidad de desarrolladores que utiliza esta herramienta también intenta mejorarla proponiendo nuevas reglas de análisis, nuevas funcionalidades, ayudando a otros desarrolladores principiantes, informando de problemas con la herramienta y compartiendo la información.
- **Aplicación de estándares conocidos**. Una gran parte de las reglas de análisis que se aplican en SonarQube tienen como origen algunas listas de vulnerabilidades sofware reconocidas internacionalmente, como puede ser la lista CWE, CVE, OWASP Top 10, etc.

El objetivo principal de esta herramienta es mejorar la calidad del código y, por lo tanto, las reglas de las que dispone por defecto se centran en comprobar la calidad del código, consiguiendo de paso evitar malas prácticas de programación que podrían suponer un riesgo de seguridad. Por lo tanto, estas reglas no se centran en comprobar vulnerabilidades específicas como la inyección, la pérdida de autenticación, la exposición de datos sensibles o los XSS.

A pesar del inconveniente mencionado, existe una razón por la que se ha elegido esta herramienta, y es la posibilidad de implementar reglas propias y configurar todos los parámetros del análisis, dando al usuario la libertad de elegir el enfoque de este.

4. DISEÑO DE LA SOLUCIÓN

4.1. Resultados de aprendizaje

El objetivo de este curso es que el alumno adquiera una serie de conocimientos y capacidades, que se describen a continuación:

- La capacidad de leer, analizar y comprender las instrucciones de un curso.
- La capacidad de localizar, identificar y seleccionar los conceptos clave de una lección.
- La capacidad de examinar la documentación de un sistema software.
- La capacidad de completar un test de evaluación en base al conocimiento adquirido.
- Un conocimiento integral de las funcionalidades que ofrece una herramienta de análisis estático de código fuente.
- La capacidad de inspeccionar y probar una herramienta de análisis estático de código fuente.
- La capacidad de ejecutar un análisis sobre un proyecto software.
- La capacidad de valorar y clasificar el resultado de un análisis de código fuente.
- La capacidad de aplicar una solución a los problemas encontrados durante el análisis.
- La capacidad de preparar, diseñar y elaborar una regla de análisis personalizada.

4.2. Estructura de la solución

En este primer epígrafe se describe la estructura de la solución diseñada para el problema mencionado en los capítulos anteriores.

Este curso se estructura en cinco módulos, a lo largo de los cuales se conocerá cada una de las funcionalidades de SonarQube. Los cuatro primeros módulos tienen como objetivo preparar el entorno de trabajo y dar a conocer la interfaz de cada herramienta a utilizar, mientras que el quinto tiene como objetivo poner en práctica, lo aprendido en los módulos anteriores, en un caso real.

A continuación, se describe el contenido de cada módulo:

Módulo 1. Requisitos previos: Se proporciona información e instrucciones para realizar configuraciones e instalaciones previas.

Módulo 2. SonarQube: Se proporciona información e instrucciones para realizar la instalación y configuración de la herramienta SonarQube.

Módulo 3. SonarQube Scanner: Se proporciona información e instrucciones para realizar la instalación y configuración de la herramienta SonarQube Scanner.

Módulo 4. Interfaz de usuario: Se proporciona información sobre las diferentes vistas que componen la interfaz web de la herramienta SonarQube y las posibles funciones de cada una.

Módulo 5. Caso Práctico: Se proporcionan instrucciones detalladas para probar las principales funcionalidades de SonarQube sobre un proyecto real. Estas funcionalidades incluyen el análisis de un proyecto, la evaluación de los resultados, la resolución de los problemas encontrados y la creación de reglas de análisis personalizadas para detectar una de las principales vulnerabilidades que existen en la actualidad.

Cada uno de estos módulos está dividido en una o más secciones, en las que se proporciona la información y las instrucciones necesarias para entender los conceptos que se intentan transmitir en el módulo.

Al final de los primeros cuatro módulos hay una o más cuestiones de Autoevaluación, las cuales es recomendable realizar para afianzar los conocimientos adquiridos y cuya solución se puede encontrar en el Anexo B.

En la Tabla 4.1. se puede observar la relación entre cada módulo y los conocimientos que se adquieren en cada uno de estos:

TABLA 4.1. RELACIÓN ENTRE MÓDULOS Y TAREAS

Tarea \ Módulo	**Módulo 1.Requisitos previos**	**Módulo 2. SonarQube**	**Módulo 3. SonarQube Scanner**	**Módulo 4. Interfaz de usuario**	**Módulo 5. Caso Práctico**
Instalación de Java	X				
Configuración de memoria	X				
Instalación de Pylint	X				
Instalación de SonarQube		X			
Arranque de SonarQube		X			
Acceso a la interfaz web		X			
Consulta de Logs		X			
Instalación de SonarQube Scanner			X		
Arranque de SonarQube Scanner			X		
Configuración de parámetros del análisis			X		
Descripción de la interfaz web				X	
Cambio de contraseña				X	
Actualización de plugins				X	
Preparación del proyecto					X
Análisis del proyecto					X
Evaluación de los resultados					X
Resolución de los problemas					X
Creación de nuevas reglas					X

Una vez detallados los módulos del curso y las tareas que se llevarán a cabo, es necesario proporcionar una estimación de la duración de este curso. En la Tabla 4.2. se observa el tiempo estimado que se debe dedicar a cada módulo para conseguir los resultados de aprendizaje.

TABLA 4.2. DURACIÓN ESTIMADA DEL CURSO

Módulo	Horas estimadas
1. Requisitos previos	2

Módulo	Horas estimadas
2. SonarQube	2
3.SonarQube Scanner	2
4. Interfaz de Usuario	4
5. Caso Práctico	10
TOTAL	**20**

4.3. Requisitos software y hardware

En este epígrafe se detallan los requisitos técnicos que debe cumplir el sistema donde se instalará y ejecutará tanto SonarQube como SonarQube Scanner.

Para ejecutar y usar el servidor de SonarQube a pequeña escala (grupo pequeño) se necesita al menos 2GB de memoria RAM para la instancia del servidor y 1GB de memoria RAM para el SO.

La cantidad de espacio necesario en el disco duro depende exclusivamente de cuanto código se desea analizar y la frecuencia de los análisis, aunque para obtener el máximo rendimiento del servidor de SonarQube, este debe ser instalado en un disco duro con una alta velocidad de lectura y escritura.

Así mismo, tanto el servidor de SonarQube como SonarQube Scanner requieren una versión específica de JVM para funcionar correctamente, así como tener la última versión del navegador. En la Tabla 3.2. se detallan los requisitos en cuando a la versión de Java y del navegador.

TABLA 4.3. REQUISITOS TÉCNICOS

Java		**Navegador**	
Oracle JRE	8	Microsoft Internet Explorer	IE 11
OpenJDK	8	Microsoft Edge	último
IBM JRE	x	Google Chrome	último
GCJ	x	Mozilla Firefox	último
Oracle JRockit	x	Safari	último

4.4. Diseño y elaboración de la solución

En este epígrafe se describe la solución técnica diseñada, siguiendo la estructura y requisitos mencionados anteriormente, para el problema descrito en los primeros capítulos de este documento.

Módulo 0: Aclaraciones previas

En este módulo introductorio se describen aspectos importantes relacionados con la utilidad de SonarQube y las funcionalidades de las que dispone.

SonarQube es una herramienta que permite realizar análisis estático de código fuente para detectar vulnerabilidades, bugs y code smells (malos hábitos de programación).

El objetivo de esta herramienta es mejorar la calidad del código y, por lo tanto, las reglas de las que dispone se centran en esta propiedad. Estas reglas no centran su enfoque en vulnerabilidades específicas como la inyección, la pérdida de autenticación, la exposición de datos sensibles o los XSS.

A pesar de este inconveniente, SonarQube ofrece la posibilidad de implementar reglas personalizadas y configurar todos los parámetros del análisis, dando al usuario la libertad de elegir el enfoque de la seguridad o el de la calidad.

Así pues, al finalizar el estudio de los siguientes cinco módulos, el lector será capaz de manejar con fluidez tanto SonarQube como SonarQube Scanner, analizar cualquier proyecto, evaluar los resultados y, si el enfoque de las reglas por defecto no es el deseado, crear reglas personalizadas enfocadas a detectar vulnerabilidades específicas.

Módulo 1: Requisitos previos

En este primer módulo se detallarán los pasos necesarios para satisfacer los requisitos previos a la instalación y uso de SonarQube y SonarQube Scanner.

Módulo 1 Sección 1: Java

En esta sección se describen las instrucciones que deben seguirse para instalar la versión de Java requerida para permitir la correcta ejecución de ambas herramientas.

Instrucciones

1. Comprobar si el sistema tiene Java (Oracle JRE 8 u OpenJDK 8) instalado.

   # Windows/Linux # - Abrir un terminal y ejecutar el siguiente comando:

   ```
   java -version
   ```

   Si, tras ejecutar el comando, el terminal imprime las siguientes líneas, se debe saltar al paso 3:

   java/openjdk version "1.X.X_XXX"
   java (TM) SE/OpenJDK Runtime Environment ...

2. Instalar Java (Oracle JRE 8 u OpenJDK 8) en la máquina donde se va a ejecutar SonarQube.

   # Windows # - Seguir los siguientes pasos:

   - Abrir el navegador y realizar una búsqueda con el criterio *"Java SE Runtime Environment 8"*.
   - Seleccionar el resultado *"Java SE Runtime Environment 8 – Downloads - Oracle"*.
   - Aceptar los Acuerdos de Licencia y descargar el fichero con extensión *".exe"* dependiendo de la arquitectura (32 o 64 bits) de la máquina.
   - Instalar el fichero descargado.
   - Reiniciar el sistema.

   # Linux # - Abrir un terminal y ejecutar el siguiente comando:

   ```
   sudo apt-get install openjdk-8-jre && sudo apt-get update
   ```

Módulo 1 Sección 2: Parámetros de memoria

En esta sección se describen las instrucciones que deben seguirse para comprobar determinados parámetros de la memoria que usa la herramienta y, en caso de ser

necesario, modificar dichos parámetros para permitir la correcta ejecución de ambas herramientas.

Instrucciones

1. Comprobar los parámetros de la memoria del sistema para permitir la correcta ejecución de la herramienta. Este paso solo se debe tener en cuenta si la herramienta se ejecuta en un S.O. Linux.

   # Linux # - Abrir un terminal y ejecutar el siguiente comando:

   - *sysctl vm.max_map_count && sysctl fs.file-max && ulimit -n && ulimit -u*

   Si tras ejecutar los comandos anteriores el terminal imprime las siguientes líneas con unos valores que cumplan la condición, se debe omitir el paso 4:

   vm.max_map_count ≥ 262144
   fs.file-max ≥ 65536
   ≥ 65536
   ≥ 2048

2. Configurar los parámetros, mencionados en el paso anterior, en cada sesión de trabajo. Este paso solo se debe tener en cuenta si la herramienta se ejecuta en un S.O. Linux.

   # Linux # - Abrir un terminal y ejecutar los siguientes comandos:

   - *sudo sysctl vm.max_map_count=262144 && sudo sysctl fs.file-max=65536 && ulimit -n 65536 && ulimit -u 2048*

Módulo 1 Sección 3: Pylint

En esta sección se describen las instrucciones que deben seguirse para instalar un analizador estático de código que es externo a la herramienta pero que es usado por el plugin SonarPython. Este plugin usa sus propias reglas y las proporcionadas por Pylint para realizar un análisis más exhaustivo y, por tanto, poder hallar mas problemas en el código.

Instrucciones

1. Instalar la herramienta Pylint.

   # Linux # - Abrir un terminal y ejecutar el siguiente comando:

   - *sudo apt-get install pylint*

 # Windows # - Abrir un terminal y ejecutar el siguiente comando:

- *pip install pylint*

Módulo 1: Autoevaluación

1. Investigar la Documentación de SonarQube, así como fuentes de información externas, para dar respuesta a las siguientes cuestiones:
    a. ¿Qué representa cada uno de los parámetros de memoria mencionados en la Sección 2?
    Pista: Sección 'Requirements'
    b. ¿Existe alguna forma de fijar el valor de dichos parámetros de manera permanente, de forma que no sea necesario configurarlos dinámicamente en cada sesión de trabajo? En caso afirmativo, ¿cuáles son los pasos que se deben seguir para conseguir tal objetivo?
    Pista: Sección 'Requirements'
2. Investigar la Documentación de SonarQube, así como fuentes de información externas, para dar respuesta a la siguiente cuestión:
    a. ¿Existe alguna otra característica que debe tenerse en cuenta para conseguir el correcto funcionamiento de SonarQube o SonarQube Scanner?

Módulo 2: SonarQube

En este segundo módulo se detallarán los pasos necesarios para llevar a cabo la instalación, así como el arranque, de la herramienta SonarQube en uno de los entornos descritos en el epígrafe anterior, es decir, en un S.O. Linux y/o en un S.O. Windows.

Módulo 2 Sección 1: Instalación

En esta sección se describen las instrucciones que deben seguirse para instalar la herramienta SonarQube.

Instrucciones

1. Descargar y configurar los parámetros necesarios para permitir la ejecución de SonarQube.

   # Windows # - Seguir los siguientes pasos:

   - Abrir el navegador y acceder a la página oficial de SonarQube.
   - Descargar la versión Community Edition con LTS. En este caso, la versión actual es 6.7.5.
   - Extraer el contenido del fichero, con extensión *".zip"*, en cualquier directorio, aunque por facilidad se recomienda el directorio *"C:\sonarqube"*.

   # Linux # - Seguir los siguientes pasos:

   - Abrir el navegador y acceder a la página oficial de SonarQube.
   - Descargar la versión Community Edition con LTS. En este caso, la versión actual es 6.7.5.
   - Extraer el contenido del fichero, con extensión *".zip"*, en cualquier directorio, aunque por facilidad se recomienda el directorio *"/opt/sonarqube"*.
   - Conceder permisos de escritura a otros usuarios y/o grupos, para permitir la correcta ejecución de los componentes requeridos por Sonarqube para arrancar, abriendo un terminal y ejecutando el siguiente comando:

     ```
     sudo chmod -R 756 /opt/sonarqube/
     ```

Módulo 2 Sección 2: Arranque

En esta sección se describen las instrucciones que deben seguirse para poner en funcionamiento la herramienta SonarQube.

Instrucciones

1. Arrancar el servidor de SonarQube instalado en la sección anterior.

   # Windows # - Abrir un terminal y ejecutar el siguiente comando:

   ```
   C:\sonarqube\bin\windows-x86-xx\StartSonar.bat
   ```

   NOTA: ***xx*** corresponde a arquitectura del sistema (32 o 64)

   # Linux # - Abrir un terminal y ejecutar el siguiente comando:

   ```
   /opt/sonarqube/bin/[OS]/sonar.sh console
   ```

   NOTA: ***[OS]*** corresponde al tipo de arquitectura Unix (Linux o MacOS)

2. Acceder a la interfaz de la herramienta mediante el navegador.

   # Windows/Linux # - Seguir los siguientes pasos:

   - Abrir el navegador e introducir la siguiente URL:

     http://localhost:9000

   - Iniciar sesión utilizando las siguientes credenciales:

     Login: admin

     Password: admin

Módulo 2: Autoevaluación

1. Investigar la Documentación de SonarQube para dar respuesta a la siguiente cuestión:
   a. ¿Qué logs se pueden consultar para obtener información útil sobre la ejecución de SonarQube?
      Pista: Sección 'Setup and Upgrade', apartado 'Troubleshooting'

Módulo 3: SonarQube Scanner

En este módulo se detallarán los pasos necesarios para llevar a cabo la instalación, así como el arranque, de la herramienta SonarQube Scanner en uno de los entornos descritos en el epígrafe anterior, es decir, en un S.O. Linux y/o en un S.O. Windows.

Módulo 3 Sección 1: Instalación

En esta sección se describen las instrucciones que deben seguirse para instalar la herramienta SonarQube Scanner.

Instrucciones

1. Descargar y configurar los parámetros necesarios para permitir la ejecución de SonarQube Scanner.

   # Windows # - Seguir los siguientes pasos:

   - Abrir el navegador y acceder a la página oficial de SonarQube.
   - Acceder a la sección de SonarQube Scanners, situada en el apartado de Descargas, y descargar la versión SonarQube Scanner X.X de acuerdo con el S.O. utilizado. En este caso, la versión actual es 3.2.
   - Extraer el contenido del fichero, con extensión *".zip"*, en cualquier directorio, aunque por facilidad se recomienda el directorio *"C:\sonar-scanner"*.
   - Editar la variable *Path*, situada en las variables de entorno del sistema, agregando la siguiente ruta al valor de la variable:

     C:\sonar-scanner\bin;

   - Comprobar si el paso anterior se ha realizado correctamente mediante la ejecución del siguiente comando en un terminal:

     ```
     sonar-scanner.bat -h
     ```

   # Linux # - Seguir los siguientes pasos:

   - Abrir el navegador y acceder a la página oficial de SonarQube.
   - Acceder a la sección de SonarQube Scanners, situada en el apartado de Descargas, y descargar la versión SonarQube Scanner X.X de acuerdo con el S.O. utilizado. En este caso, la versión actual es 3.2.
   - Extraer el contenido del fichero, con extensión *".zip"*, en cualquier directorio, aunque por facilidad se recomienda el directorio *"/opt/sonar-scanner"*.
   - Editar el fichero *~/.bash_profile* (o *~/.profile*) o el fichero */etc/profile*, para incorporar el directorio de ficheros binarios de Sonar Scanner a las variables de entorno del S.O., agregando la siguiente expresión al final del documento:

     export PATH=/opt/sonar-scanner/bin:$PATH

- Comprobar si el paso anterior se ha realizado correctamente mediante la ejecución del siguiente comando en un terminal:

```
sonar-scanner -h
```

Módulo 3 Sección 2: Arranque

En esta sección se describen las instrucciones que deben seguirse para poner en funcionamiento la herramienta SonarQube Scanner.

Instrucciones

1. Arrancar SonarQube Scanner, instalado en la sección anterior, para realizar el análisis del proyecto seleccionado.

 # Windows/Linux # - Seguir los siguientes pasos:

- Arrancar la herramienta SonarQube siguiendo las instrucciones indicadas en el módulo anterior.
- Crear un fichero de configuración, en el directorio raíz del proyecto, con el nombre "*sonar-project.properties*" e introducir los siguientes datos:

  sonar.projectKey=mi:proyecto

  sonar.projectName=Mi proyecto

  sonar.projectVersion=1.0

  sonar.sources=.

  #sonar.sourceEncoding=UTF-8

  Aclaraciones:

  La clave del proyecto (*projectKey*) debe ser un identificador único del proyecto para la instancia actual del servidor SonarQube. No deben existir dos proyectos con la misma clave.

  El nombre del proyecto (*projectName*) solo se utiliza para identificar dicho proyecto en la interfaz de SonarQube, por lo que no es necesario que sea único en la instancia.

  La versión del proyecto (*projectVersion*) debe incrementarse a medida que se analiza el mismo proyecto en repetidas ocasiones.

  La localización del código fuente (*sources*) es relativa al fichero de configuración "*sonar-project.properties*".

  La codificación del código fuente (*sourceEncoding*) se debe tener en cuenta únicamente cuando se desea utilizar una codificación diferente a la del sistema.

- Abrir un terminal en el directorio raíz del proyecto y ejecutar el siguiente comando:

```
sonar-scanner
```

Módulo 3: Autoevaluación

1. Investigar la Documentación de SonarQube para dar respuesta a la siguiente cuestión:
   a. ¿Existe una alternativa a tener que crear el fichero *sonar-project.properties* para analizar un proyecto? En caso afirmativo, ¿Cuáles son los pasos que se deben seguir para conseguirlo?

      Pista: Sección 'Scanners', apartado 'SonarQube Scanner', subapartado 'Advanced SonarQube Scanner Usages'

2. Investigar la Documentación de SonarQube para dar respuesta a la siguiente cuestión:
   a. ¿Cuál es el problema más recurrente que podría producirse al ejecutar SonarQube Scanner? ¿Tiene solución?
      Pista: Sección 'Scanners', apartado 'SonarQube Scanner'

Módulo 4: Interfaz de usuario

En este módulo se presentará la interfaz de usuario de la herramienta SonarQube, así como las nociones más relevantes para manejar dicha interfaz con facilidad.

Módulo 4 Sección 1: Vista previa

En esta sección se describen en detalle los componentes más importantes de la interfaz previa al inicio de sesión.

Fig. 14. Vista previa al inicio de sesión

La Figura 1 representa la vista previa de la interfaz de SonarQube una vez que se ha puesto en marcha el servidor y se ha accedido a la interfaz mediante el navegador. Esta vista previa permite visualizar los componentes más importantes de la herramienta, aunque no permite modificar ninguno de ellos hasta que no se inicie sesión.

Como se puede observar en la Figura 1, esta vista previa está compuesta por nueve elementos bien diferenciados:

1. *Projects*. Vista que contiene información detallada sobre todos los proyectos analizados hasta el momento.

2. *Issues*. Vista que contiene información detallada todos los bugs, vulnerabilidades y malos hábitos de programación que se han encontrado hasta el momento en todos los proyectos analizados.

3. *Rules*. Vista que contiene información detallada todas las reglas que existen, para todos los lenguajes soportados por la herramienta, y que se aplican al ejecutar un análisis sobre un proyecto.

4. *Quality Profiles*. Vista que contiene información detallada sobre todos los perfiles de calidad que tiene la herramienta para cualquiera de los lenguajes de programación soportados. Estos perfiles de calidad no son más que colecciones de reglas que se aplican durante el análisis.

5. *Quality Gates*. Vista que contiene información detallada sobre los límites definidos que permiten establecer si el código de un proyecto es de calidad o no. Cabe mencionar que estos límites se pueden personalizar según la situación.

6. *Búsqueda*. Cuadro de búsqueda que permite encontrar fácilmente un proyecto concreto.

7. *Log in*. Vista que permite iniciar sesión, mediante el uso de unas credenciales, en la interfaz de la herramienta.

8. *Read documentation*. Vista que redirige a la documentación oficial de la herramienta.

9. *Panel de información*. Cuadro de información que permite conocer el número total de proyectos analizados hasta el momento y los problemas encontrados en conjunto.

Módulo 4 Sección 2: Vista principal

En esta sección se describen en detalle los componentes más importantes de la interfaz una vez que se ha iniciado sesión.

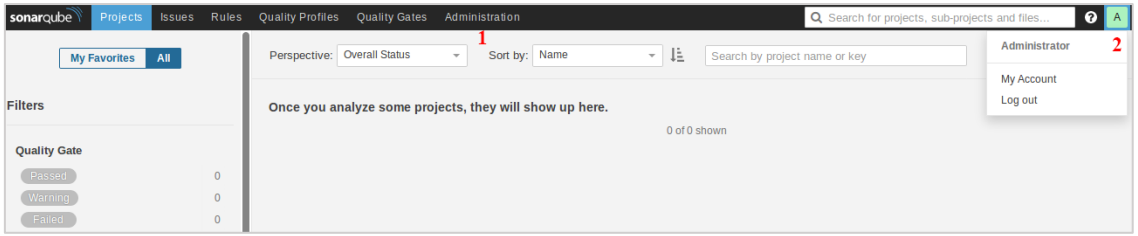

Fig. 15. Vista principal

La Figura 2 representa la vista principal de la interfaz de SonarQube una vez que se ha iniciado sesión en la vista previa. Esta vista principal cuenta con dos componentes adicionales y, a diferencia de la anterior, permite modificar cada uno de los componentes dependiendo de los permisos que tenga la cuenta logueada. En este caso, se han utilizado las credenciales de la sección 2 del segundo módulo de este capítulo.

1. *Administration*. Vista que contiene información detallada sobre la configuración global de la herramienta, la seguridad y los permisos de los usuarios que tienen acceso a esta interfaz, la gestión de los proyectos, la máquina en la que se ejecuta la herramienta y el marketplace.

2. *Panel de usuario*. Vista que contiene información sobre la cuenta con la que se ha iniciar sesión y que permite editar dicha información.

Módulo 4 Sección 3: Vista de Projects

En esta sección se describen los componentes más importantes que se pueden encontrar al acceder a la vista de Projects.

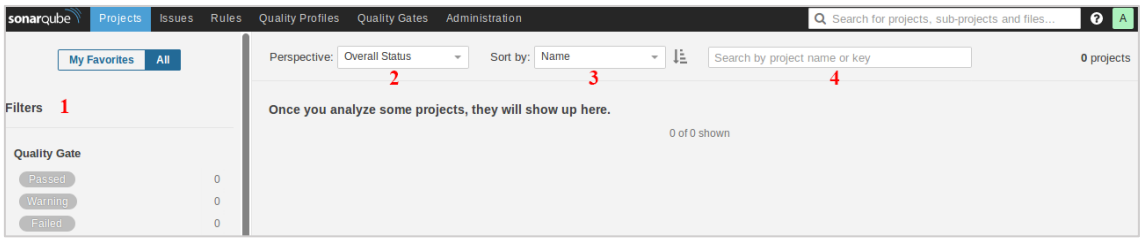

Fig. 16. Vista de Projects

La Figura 3 representa la vista de los proyectos analizados hasta el momento. Esta vista cuenta con cuatro componentes bien diferenciados.

1. *Filters*. Panel compuesto por diferentes reglas de filtrado relacionadas con la calidad del código y que permite visualizar únicamente aquellos proyectos que cumplan la/s regla/s de filtrado seleccionada/s.

2. *Perspective*. Panel que permite visualizar diferentes perspectivas de los proyectos analizados, como pueden ser diferentes gráficos que muestran el grado de mantenibilidad en base a las líneas de código o qué proporción del código analizado presenta riesgos de contener una vulnerabilidad.

3. *Sort by*. Panel que permite clasificar los proyectos mediante diferentes criterios, como puede ser el nombre, la seguridad, el tamaño, etc.

4. *Búsqueda*. Cuadro de búsqueda que permite encontrar fácilmente un proyecto por su nombre o su key.

Módulo 4 Sección 4: Vista de Issues

En esta sección se describen los componentes más importantes que se pueden encontrar al acceder a la vista de Issues.

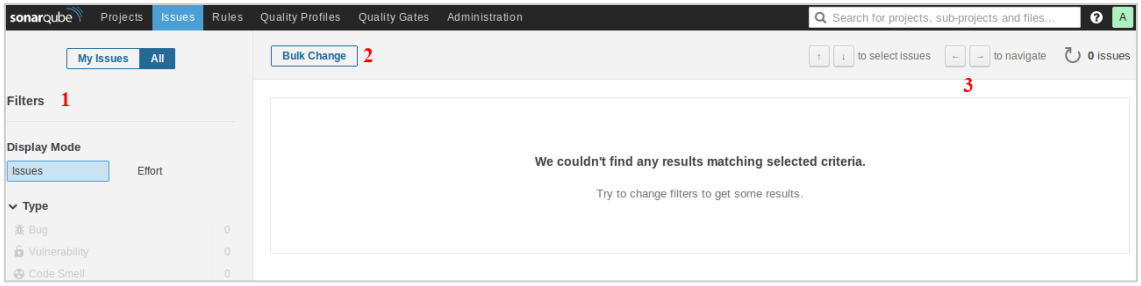

Fig. 17. Vista de Issues

La Figura 4 representa la vista de los problemas (Bugs, Vulnerabilidades y/o Code Smell's) encontrados hasta el momento en cada uno de los proyectos analizados. Esta vista cuenta con tres componentes bien diferenciados.

1. *Filters*. Panel compuesto por diferentes criterios de filtrado (por tipo de error, gravedad, etc) y que permite visualizar únicamente aquellos proyectos que cumplan los criterios de filtrado seleccionados.

2. *Bulk Change*. Panel que permite aplicar un cambio determinado a más de un problema a la vez. En este caso, este panel permite asignar un cierto número de problemas a un usuario determinado, cambiar la gravedad de los problemas, etc.

3. *Navigate*. Panel compuesto por diferentes métodos de recorrer la lista de problemas.

Módulo 4 Sección 5: Vista de Rules

En esta sección se describen los componentes más importantes que se pueden encontrar al acceder a la vista de Rules.

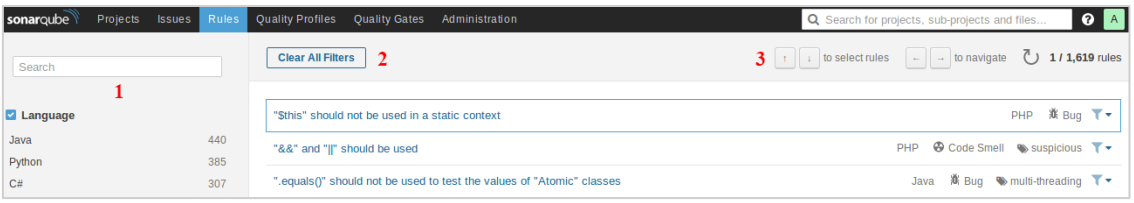

Fig. 18. Vista de Rules

La Figura 5 representa la vista de las diferentes reglas que se aplican durante el análisis del código fuente. Esta vista cuenta con tres componentes bien diferenciados.

1. *Filtrado*. Panel compuesto por diferentes criterios de filtrado (por lenguaje de programación, tipo de problema, perfil de calidad, etc) y que permite visualizar únicamente aquellas reglas que cumplan los criterios de filtrado seleccionados.

2. *Clear All Filters*. Panel que permite eliminar, a la vez, todos los criterios de filtrado seleccionados.

3. *Navigate*. Panel compuesto por diferentes métodos de recorrer la lista de reglas.

Módulo 4 Sección 6: Vista de Quality Profiles

En esta sección se describen los componentes más importantes que se pueden encontrar al acceder a la vista de Quality Profiles.

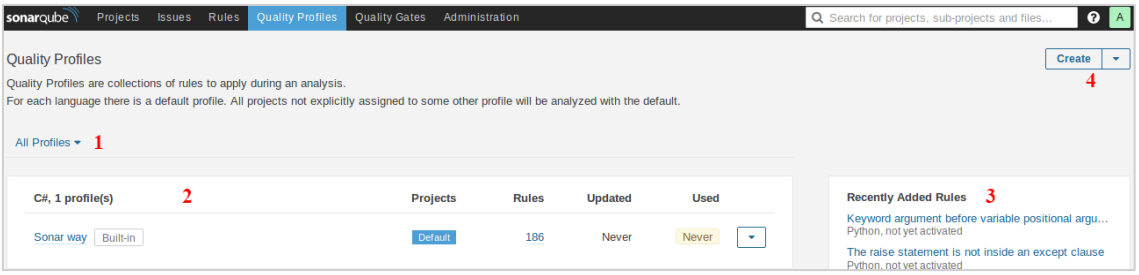

Fig. 19. Vista de Quality Profiles

La Figura 6 representa la vista de los diferentes perfiles de calidad que están disponibles actualmente en la herramienta y que se aplican durante el análisis del código fuente. Esta vista cuenta con cuatro componentes bien diferenciados.

1. *Filtrado*. Panel que permite filtrar los perfiles de calidad por el lenguaje de programación al que se pueden aplicar.

2. *Lista de perfiles de calidad*. Panel que muestra información sobre los diferentes perfiles de calidad disponibles. Esta información incluye el lenguaje al que se puede aplicar, a qué proyectos se aplica, el número de reglas incluidas en el perfil, la fecha en la que se actualizó por última vez y algunas acciones útiles como clonar el perfil o compararlo con otro.

3. *Recently Added Rules*. Panel que muestra información sobre las reglas añadidas recientemente a los diferentes perfiles de calidad.

4. *Create*. Panel que permite crear un perfil de calidad completamente nuevo.

Módulo 4 Sección 7: Vista de Quality Gates

En esta sección se describen los componentes más importantes que se pueden encontrar al acceder a la vista de Quality Gates.

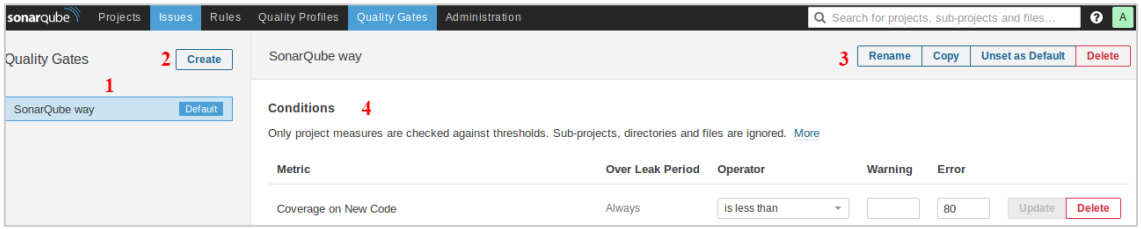

Fig. 20. Vista de Quality Gates

La Figura 7 representa la vista de los diferentes umbrales de calidad que se pueden aplicar para establecer si el código fuente de un proyecto cumple ciertos estándares de calidad. Esta vista cuenta con cuatro componentes bien diferenciados.

1. *Lista de umbrales de calidad*. Panel que muestra los diferentes umbrales de calidad disponibles.

2. *Create*. Panel que permite crear un nuevo umbral de calidad.

3. *Panel de edición*. Panel que muestra diferentes acciones (renombrar, copiar, borrar, etc) aplicables al umbral de calidad seleccionado.

4. *Condiciones del umbral*. Panel que muestra información sobre las diferentes métricas que se aplican en el umbral seleccionado, así como el límite a partir del cual dichas métricas se consideran incumplidas.

Módulo 4 Sección 8: Vista de Administration

En esta sección se describen los componentes más importantes que se pueden encontrar al acceder a la vista de Administration.

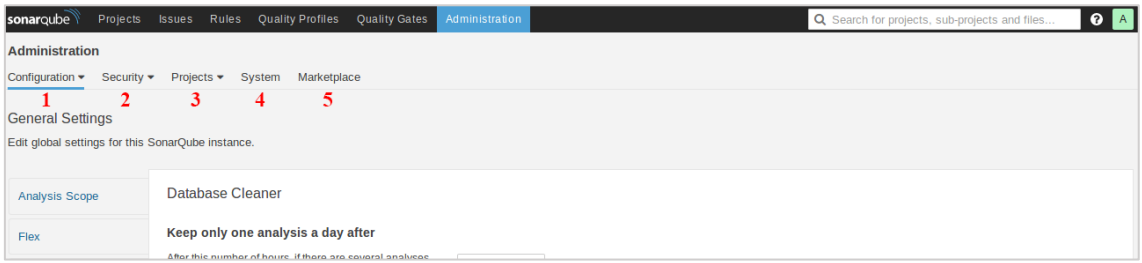

Fig. 21. Vista de Administration

La Figura 8 representa la vista de los ajustes que se pueden aplicar a los diferentes componentes de SonarQube. Esta vista cuenta con cinco componentes bien diferenciados.

1. *Configuration*. Vista que permite visualizar y/o realizar ajustes globales a la instancia actual de la herramienta.

2. *Security*. Vista que permite visualizar y/o realizar ajustes de seguridad, como puede ser la gestión de usuarios y los permisos asignados a cada uno de estos.

3. *Projects*. Vista que permite visualizar y/o realizar ajustes generales a los proyectos, como puede ser la gestión de estos o las tareas que se ejecutan en segundo plano cuando se analiza un proyecto.

4. *System*. Vista que permite visualizar toda la información relacionada con la instancia de la herramienta o el sistema en el que se está ejecutando, así como la posibilidad de obtener diferentes logs. Así mismo, en esta vista se permite reiniciar el servidor de SonarQube.

5. *Marketplace*. Vista que permite descubrir e instalar nuevas funcionalidades (plugins) y actualizar las existentes.

Módulo 4: Autoevaluación

1. Teniendo en cuenta lo descrito en este mismo módulo, responda a la siguiente cuestión:
    a. ¿Cuáles son los pasos que deben seguirse para cambiar la contraseña de la cuenta de Administrador?

       Pista: Vista 'Administration' o 'Panel de usuario'
2. Teniendo en cuenta lo descrito en este mismo módulo, responda a la siguiente cuestión:
    a. ¿Cuáles son los pasos que deben seguirse para actualizar la funcionalidad SonarPython?

       Pista: Vista 'Administration'

Módulo 5: Caso práctico

En este módulo se describirá, de manera práctica, como realizar el análisis de un proyecto real, la interpretación y evaluación de los resultados del análisis, la creación de un perfil de calidad personalizado y, por último, la creación de reglas personalizadas escritas en el lenguaje XPath que servirán para detectar las vulnerabilidades más comunes.

Paso 1: Preparar el proyecto

Para llevar a cabo la demostración, de un caso práctico, se necesita un proyecto real. En este caso, para demostrar que esta herramienta de análisis se puede aplicar a cualquier proyecto, en este caso escrito en Python, se ha decidido escoger de manera aleatoria un proyecto de la plataforma Github, cuya url se proporciona a continuación.

URL = https://github.com/python-telegram-bot/python-telegram-bot

El proyecto se puede clonar, mediante la herramienta Git, o se puede descargar directamente. Por facilidad se ha optado por descargar el ZIP del proyecto directamente.

Una vez descargado el ZIP del proyecto, debe extraerse el contenido en una localización de fácil acceso, pues se accederá en más de una ocasión al mismo.

Paso 2: Analizar el proyecto

El primer paso para analizar un proyecto es arrancar el servidor SonarQube:

 # Windows #

- *C:\sonarqube\bin\windows-x86-xx\StartSonar.bat*

NOTA: xx corresponde a arquitectura del sistema (32 o 64)

 # Linux #

- */opt/sonarqube/bin/[OS]/sonar.sh console*

NOTA: ***[OS]*** corresponde al tipo de arquitectura Unix (Linux o MacOS)

Una vez arrancado se debe abrir la interfaz web de SonarQube e iniciar sesión (por defecto es *login:admin password:admin*):

http://localhost:9000

Después de iniciar sesión, se deben habilitar las reglas de Pylint puesto que están deshabilitadas por defecto. Para hacerlo, se deben seguir las siguientes instrucciones:

- Abrir la vista *Quality Profiles*.
- Buscar el perfil de calidad asignado a Python (*'Sonar way'*) en la lista de perfiles.
- Desplegar el menú, situado en la parte derecha, del perfil y seleccionar la opción '*Copy*'.
- Introducir el nombre para el nuevo perfil (en este caso se ha elegido '*Modulo5'*) y confirmar la operación de copia.
- En la nueva vista, fijar el perfil creado como opción por defecto desplegando el menú '*Actions*' y, a continuación, seleccionando la opción '*Set as default*'.
- Seleccionar la opción '*Activate More*' para activar la totalidad de las reglas.
- Seleccionar la opción '*Bulk Change*' y, a continuación, '*Activate in Modulo5*'.

Tras habilitar estas reglas se debe configurar los parámetros de SonarQube Scanner. Estos parámetros se pueden fijar directamente en el terminal o creando el fichero *sonar-project.properties* dentro de la raíz del proyecto. A continuación, se indican ambas opciones:

- Abrir un terminal en la raíz del proyecto y ejecutar el siguiente comando con los parámetros correspondientes:

 # Windows #

- *sonar-scanner.bat -Dsonar.projectKey= m:o:d:u:l:o:5 -Dsonar.projectName=Modulo5 -Dsonar.projectVersion=1.0 -Dsonar.sources=.*

 # Linux #

- *sonar-scanner -Dsonar.projectKey= m:o:d:u:l:o:5 -Dsonar.projectName=Modulo5 -Dsonar.projectVersion=1.0 -Dsonar.sources=.*

- Crear un fichero con nombre *sonar-project.properties* dentro de la raíz del proyecto con el siguiente contenido:

  sonar.projectKey=m:o:d:u:l:o:5

  sonar.projectName=Modulo5

  sonar.projectVersion=1.0

  sonar.sources=.

  Una vez creado el fichero, se debe abrir un terminal en la raíz del proyecto y ejecutar el siguiente comando:

 # Windows #

- *sonar-scanner.bat*

 # Linux #

- *sonar-scanner*

Paso 3: Evaluar el resultado del análisis

Tras confirmar la finalización del análisis, se debe abrir la interfaz web de SonarQube para visualizar los resultados.

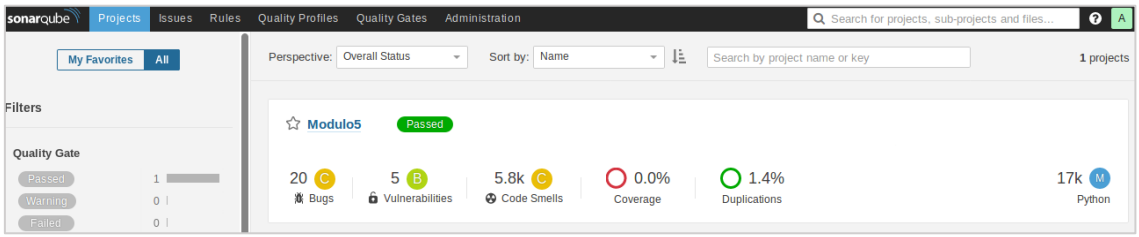

Fig. 22. Resultados del análisis

En la Figura 9 se puede observar los resultados obtenidos tras analizar el código fuente del proyecto. En la misma se puede apreciar que la interfaz proporciona la siguiente información:

Nombre del proyecto: Modulo5

Resultado del umbral de calidad: Passed

Número de bugs encontrados: 20

Número de vulnerabilidades encontradas: 5

Número de code smells encontrados: 5.8k

Porcentaje de líneas cubiertas con los tests automáticos: 0%

Porcentaje de líneas que están duplicadas: 1.4%

Número total de líneas de código: 17k

Lenguaje identificado en el proyecto: Python

Para saber más en detalle cada uno de los problemas encontrados, se debe pulsar sobre el nombre del proyecto.

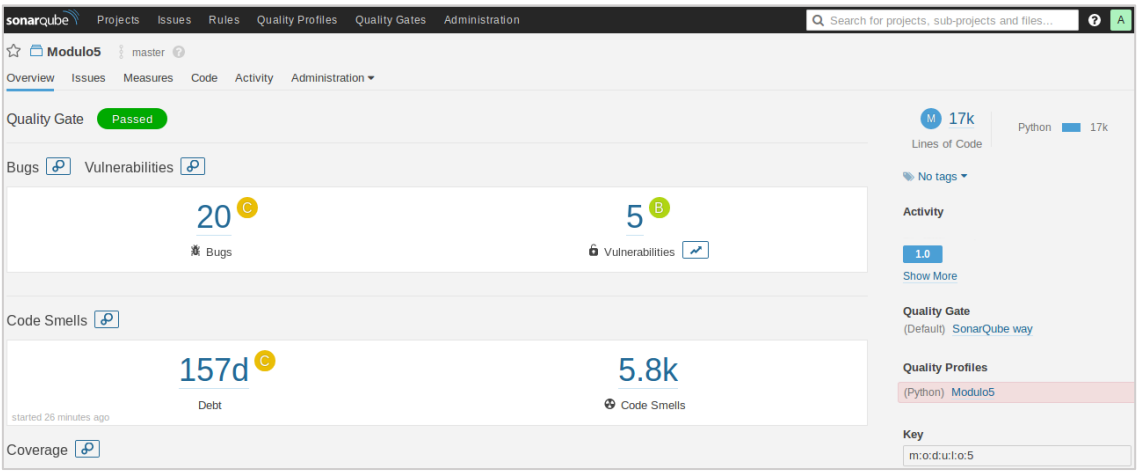

Fig. 23. Resultados detallados del análisis

En la Figura 10 se pueden observar los mismos resultados de la vista anterior y algunos detalles adicionales. Además de los detalles generales, se facilita el acceso a los detalles específicos del análisis, como puede ser la lista completa y detallada de problemas encontrados (*Issues*), una serie de métricas expresadas en gráficos (*Measures*), la lista detallada de ficheros analizados (*Code*) y la actividad del proyecto (*Activity*). Asimismo, esta vista permite acceder a todos los ajustes concernientes al proyecto (*Administration*).

Para conocer más detalles sobre los problemas encontrados se debe acceder a la vista *Issues* donde se puede observar la lista completa de problemas encontrados.

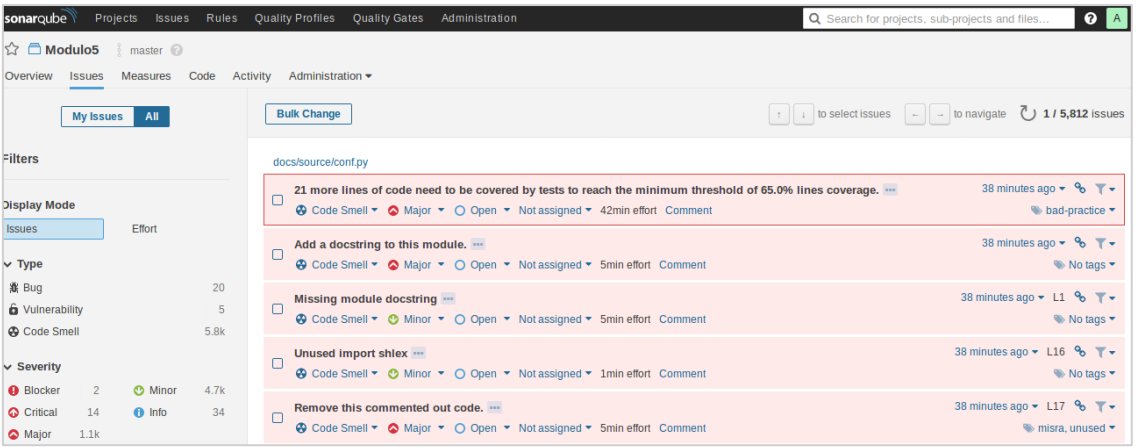

Fig. 24. Lista de problemas encontrados

En la Figura 11 se puede observar la lista detallada de todos los problemas encontrados al analizar el proyecto. Como se puede apreciar, cada uno de los problemas está enmarcado en un recuadro de color rojo y dispone de una serie de parámetros que ayudan a obtener más información sobre el problema y pueden ser modificados para mejorar el estándar de calidad.

Tomando como ejemplo el primer problema listado en la Figura 11, se pueden visualizar los siguientes datos:

> **Nombre del problema**: 21 more lines of code need to be covered by tests to reach the minimum threshold of 65.0% lines coverage
>
> **Tipo de problema**: Code smell
>
> **Gravedad del problema**: Major
>
> **Estado del problema**: Open
>
> **Usuario asignado**: Not assigned
>
> **Tiempo estimado para arreglar este problema**: 42 min effort
>
> **Etiqueta asignada**: bad-practice
>
> **Tiempo transcurrido desde su descubrimiento**: 38 minutes ago

Además de los datos anteriores, dentro del mismo recuadro del problema se pueden realizar las siguientes acciones:

> **Icono en forma de cadena**: Abrir una vista individual en la que aparece la/s línea/s donde se ha encontrado este problema en particular.
>
> **Icono en forma de embudo**: Filtrar el resto de los problemas usando alguno de los parámetros del problema actual (tipo de problema, gravedad, estado del problema, etc) como filtro.
>
> **Comment**: Permite agregar un comentario al problema actual.

Para saber la/s línea/s donde se ha detectado el problema, basta con pulsar sobre recuadro de este y se abrirá una nueva vista donde se muestra el problema actual, sobre el código fuente, y el resto de los problemas anteriores/posteriores de la lista anterior.

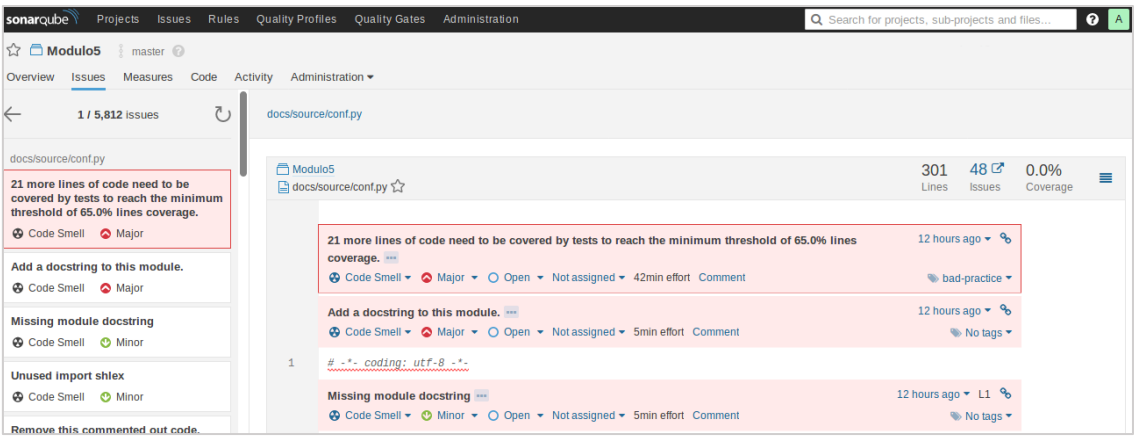

Fig. 25. Localización del problema

En la Figura 12 se puede apreciar que, al pulsar sobre el recuadro del primer problema, se abre una nueva vista en la que se indica la línea exacta de cada uno de los problemas de la lista. Además, para facilitar la navegación por la lista de problemas, se ha añadido la lista en la parte derecha de la vista, evitando de este modo tener que regresar a la vista anterior para visualizar otro problema.

Para conocer la descripción del problema y todos los detalles relacionados con este, basta con pulsar sobre el icono, representado con puntos suspensivos, situado al lado del nombre del problema.

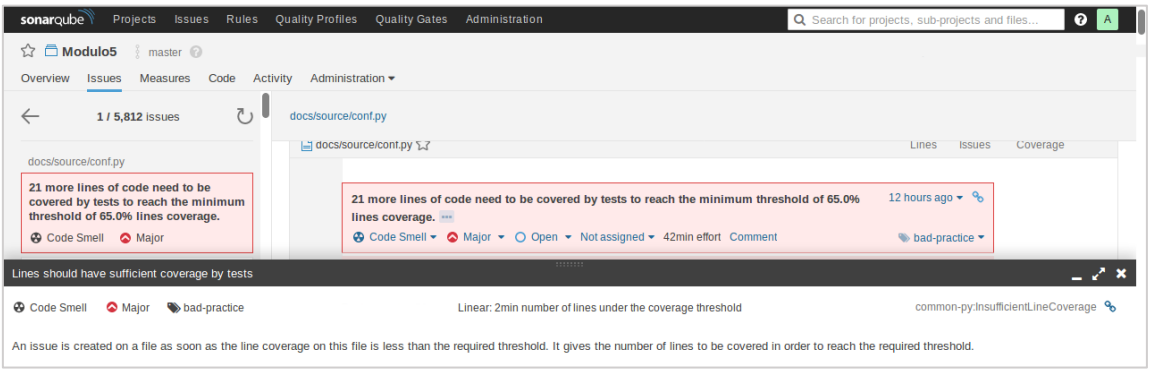

Fig. 26. Descripción del problema

En la Figura 13 se puede observar que al pulsar el icono anterior se abre una ventana, en la parte inferior de la interfaz, que contiene toda la información que se tiene sobre ese problema. En este caso, la ventana contiene únicamente una pequeña descripción que indica cuándo se activa la regla. En otros casos más complejos, la ventana contendrá información detallada sobre la razón por la que se detecta como un problema y una posible solución al mismo.

Paso 4: Resolver los problemas encontrados

Teniendo en cuenta los problemas encontrados en este proyecto, y que han sido ilustrados en la Figura 11, se procede a mostrar de manera práctica como resolver algunos de ellos.

En este caso, se van a revisar los problemas que conciernen a este curso, es decir, las vulnerabilidades. Para ello, hay que dirigirse de nuevo a la vista de Issues en la que

aparece la lista entera de problemas. En la parte izquierda de esta vista se encuentran las reglas de filtrado y entre estas reglas se encuentra la que filtra por tipo de problema. Se debe elegir el tipo vulnerabilidad, que en este caso se han detectado únicamente cinco.

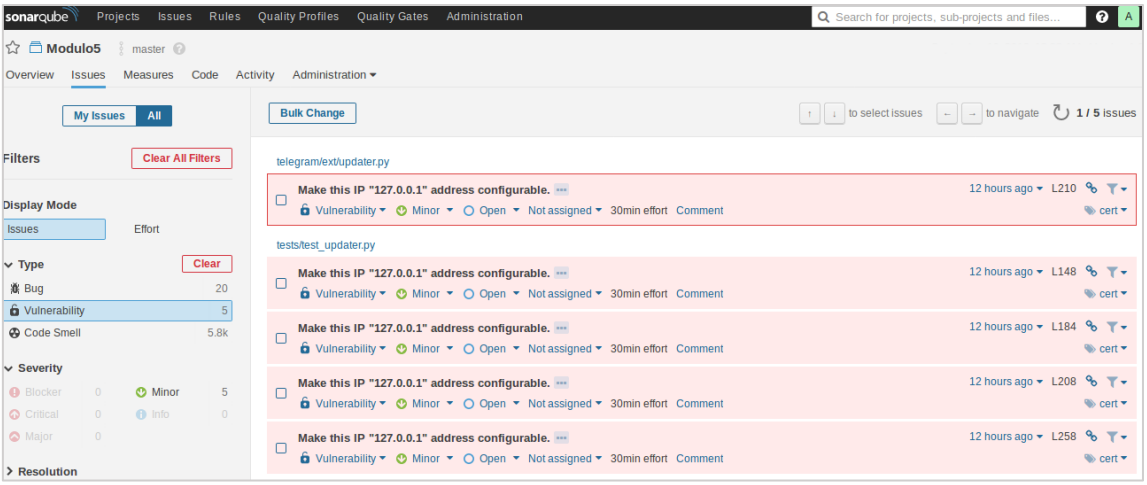

Fig. 27. Vulnerabilidades encontradas

En la Figura 14 se puede apreciar la lista de vulnerabilidades encontradas en el proyecto y en este caso se ha encontrado la misma vulnerabilidad en los cinco casos, aunque en diferentes localizaciones.

A continuación, se procede a visualizar tanto la descripción del primer problema como la línea en la que se ha encontrado.

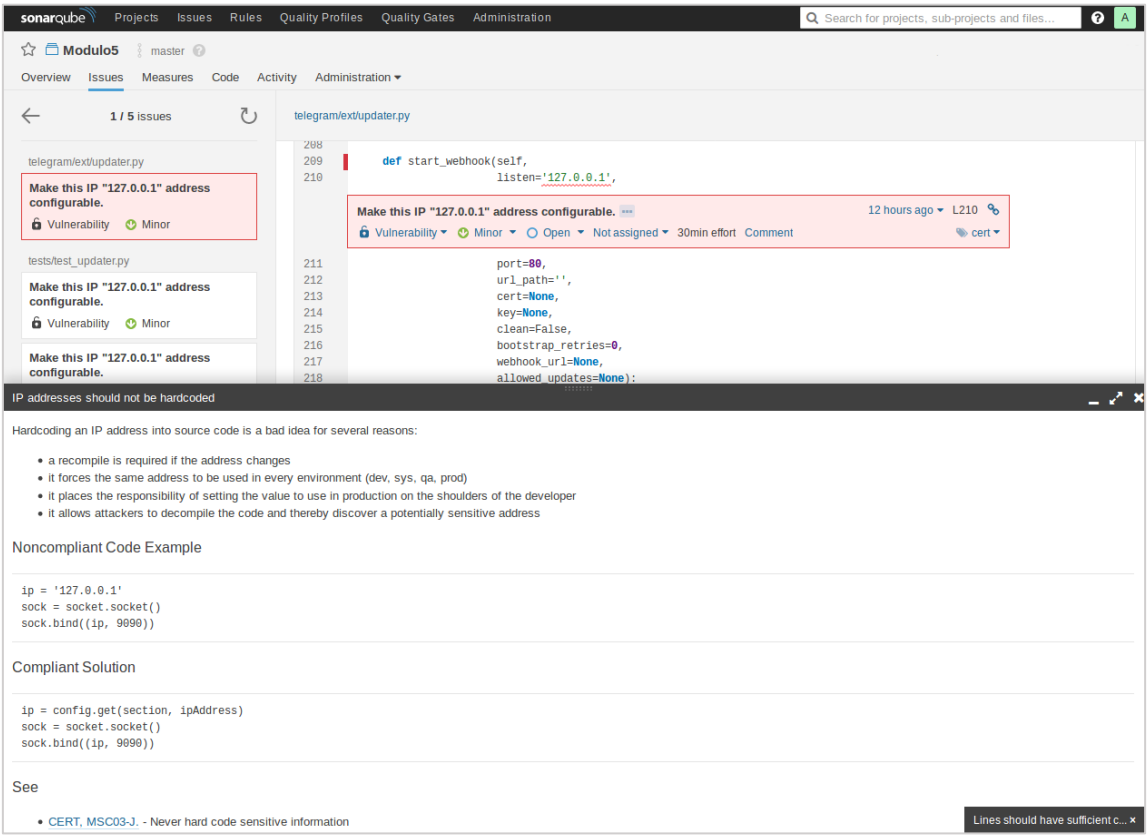

Fig. 28. Descripción y localización de la vulnerabilidad

En la Figura 15 se puede observar la línea, subrayado con rojo, donde se ha detectado la vulnerabilidad, así como el contexto (líneas anteriores y posteriores) de esta vulnerabilidad.

Además, en la parte inferior se ha abierto la ventana en la que se encuentra la información relacionada con este problema. El primer párrafo describe las razones por las que se debe solucionar este problema, la mas peligrosa siendo que un atacante podría descompilar este código y descubrir una dirección IP, en claro, potencialmente sensible a ser atacada. En este caso, se puede observar que se trata de la dirección IP local ('*127.0.0.1*'), la cual no plantea un problema de seguridad en este contexto, pues se está utilizando para crear un pequeño servidor http que está escuchando continuamente para obtener actualizaciones.

Una vez que se ha inspeccionado en detalle esta vulnerabilidad, se puede concluir que no se trata de una vulnerabilidad importante y por tanto existen dos caminos a tomar:

- Cambiar el código, para evitar que la dirección IP aparezca en claro, de la siguiente manera:

  import socket
  print socket.gethostbyname(socket.gethostname())
- Cambiar el estado de este problema de *open* a:

  Resolve as a false positive: El problema será ignorado y su tiempo estimado de resolución no será tenido en cuenta en la estimación global.

  Resolve as won't fix: El problema será ignorado porque la regla es irrelevante en el contexto actual y su tiempo estimado de resolución no será tenido en cuenta en la estimación global.

Por facilidad se ha decidido escoger la segunda opción, es decir, cambiar el estado de la vulnerabilidad a *Resolve as won't fix*.

Revisando el resto de las vulnerabilidades se puede apreciar que están en la misma situación que la primera. Según el contexto en el que se han detectado, estas vulnerabilidades no suponen un problema de seguridad y por lo tanto pueden aplicarse una de las dos opciones descritas para el problema anterior.

Tras inspeccionar y resolver todas las vulnerabilidades encontradas, se puede seguir revisando los bugs y code smells encontrados, realizando los mismos pasos seguidos para las vulnerabilidades.

Finalizado el trabajo de resolución de los problemas, se debe realizar de nuevo el análisis para saber si estos realmente han sido resueltos y si han surgido nuevos problemas desde el último análisis.

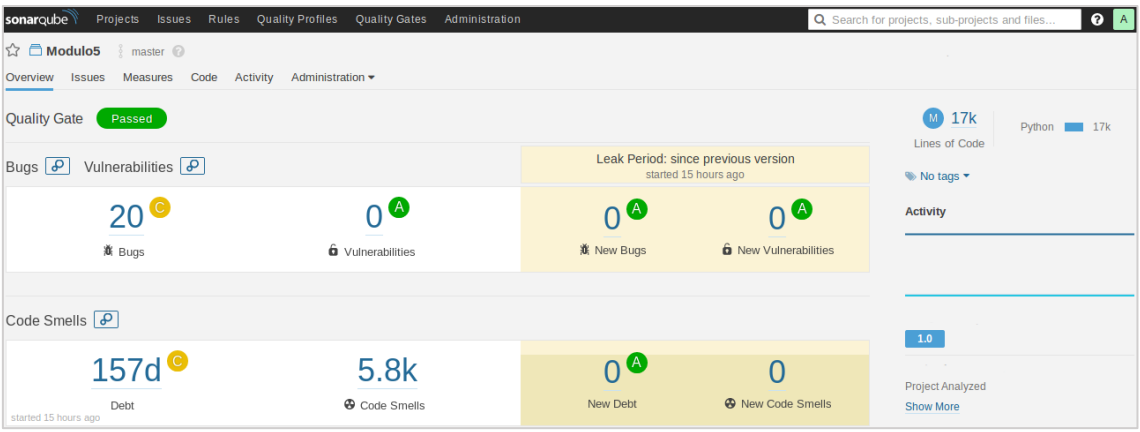

Fig. 29. Resultados del segundo análisis

En la Figura 16 se pueden apreciar los resultados obtenidos tras ejecutar el segundo análisis. En la parte derecha se muestra una comparación entre el análisis anterior y el actual en cuanto a nuevas vulnerabilidades, bugs y code smells. En esta vista se puede comprobar que las vulnerabilidades anteriores ya no existen en el nuevo análisis y tampoco se han generado nuevos problemas.

Paso 5: Creación de nuevas reglas

Con el objetivo de detectar verdaderas vulnerabilidades, como las descritas en el documento OWASP Top 10 del año 2017 [33], se debe crear reglas personalizadas que estén escritas usando el lenguaje XPath.

OWASP es un proyecto abierto creado con la intención de acordar y contener las causas que provocan problemas de seguridad para el software. Este proyecto publica, cada cierto tiempo, una lista de las diez vulnerabilidades más importantes que existen para las aplicaciones web. A pesar de estar enfocado para las aplicaciones web, esta lista se puede aplicar a cualquier aplicación y a cualquier lenguaje de programación.

Para este caso práctico, se va a explicar el proceso que se debe seguir para crear una regla que detecte la vulnerabilidad más común que existe según el OWASP Top 10, y es la Inyección (SQL, NoSQL, LDAP, etc).

Las reglas deben escribirse, en la versión 1.0 de XPath, con el objetivo de navegar por el Árbol de Sintaxis Abstracta (AST) del lenguaje de programación. Para recorrer el AST, se dispone de una herramienta denominada SSLR Toolkit (URL al final del módulo). Esta herramienta muestra el AST de un fragmento de código introducido, permitiendo visualizar como se estructuran los nodos y sus atributos. Mediante el lenguaje XPath, se puede construir una consulta que seleccione un nodo o varios nodos del AST, los cuales representan una expresión del código fuente.

Para aprender los fundamentos básicos necesarios para crear reglas en XPath se debe revisar el tutorial de ZVON y W3SCHOOLS (URL's al final del módulo). Para el alcance de este curso no se ha considerado necesario crear un tutorial adicional para aprender el lenguaje XPath puesto que los ya mencionados son suficientes para los objetivos de este curso.

Para empezar a crear reglas en XPath se debe obtener la herramienta SSLR Toolkit para Python en formato JAR. Puesto que la última versión proporcionada por SonarSource no funciona correctamente, debido a un fallo de compilación, se recomienda una versión anterior de SSLR Python Toolkit.

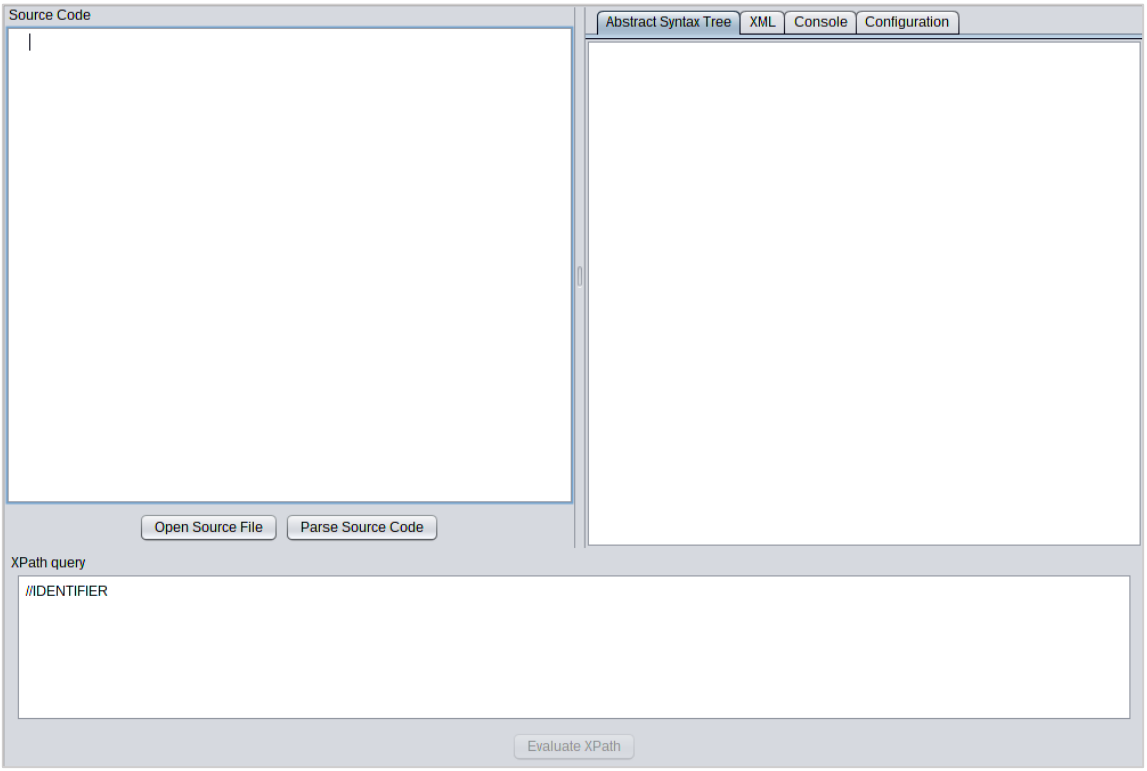

Fig. 30. Interfaz de SSLR Toolkit

En la Figura 17 se puede apreciar la interfaz gráfica de SSLR Toolkit. Esta interfaz está compuesta por siete componentes:

Source Code: Código fuente en Python cuyo AST se desea obtener.

AST: Cuadro en el que se representa el AST del código fuente insertado en Source Code.

XML: Cuadro en el que se representa el XML del código fuente insertado en Source Code.

Console: Representa la salida por consola (si existe alguna) del código fuente insertado en Source Code.

Configuration: Cuadro que permite configurar la codificación del proyecto. Por defecto será UTF-8.

XPath query: Consulta escrita en el lenguaje XPath para seleccionar uno o más nodos del AST.

Open Source File/Parse Source Code: Opciones que permiten abrir un fichero .py y obtener el AST/XML/Console del código fuente insertado en Source Code, bien a mano o bien mediante la opción Open Source File.

Para obtener una consulta XPath válida, primero se debe pensar en las líneas de código fuente que pueden representar la vulnerabilidad que se desea detectar. En este caso, se van a crear dos reglas para detectar la inyección: una que detecta la inyección mediante la entrada de datos del usuario y otra que detecta la inyección SQL.

Datos de entrada del usuario o User Input

Existen diferencias entre la forma en que Python 2 y 3 manejan los datos de entrada del usuario. La siguiente línea de código arroja diferentes resultados dependiendo de si se ejecuta en Python 2 o en Python 3:

nombre = input('Introduce un nombre: ')

print(nombre)

Python 3 evalúa la expresión anterior como se esperaría que lo hiciese, mientras que Python 2 evalúa el valor de entrada como un nombre de variable. Es decir, Python 2 espera que el valor guardado en *nombre* sea el nombre de una variable y, por lo tanto, arrojará un *NameError* si no existe ninguna variable cuyo nombre coincida con ese valor.

Por lo tanto, utilizar la función input() en Python 2 supone un riesgo para la seguridad de la aplicación, pues el usuario podría acceder a datos sensibles de esta.

Para solucionar este problema en Python 2, se debe usar la función raw_input() en vez de input(), puesto que esta función analiza el dato de entrada y lo purga.

Tras aclarar el problema existente, se procede a crear la regla en XPath haciendo uso de SSLR Toolkit.

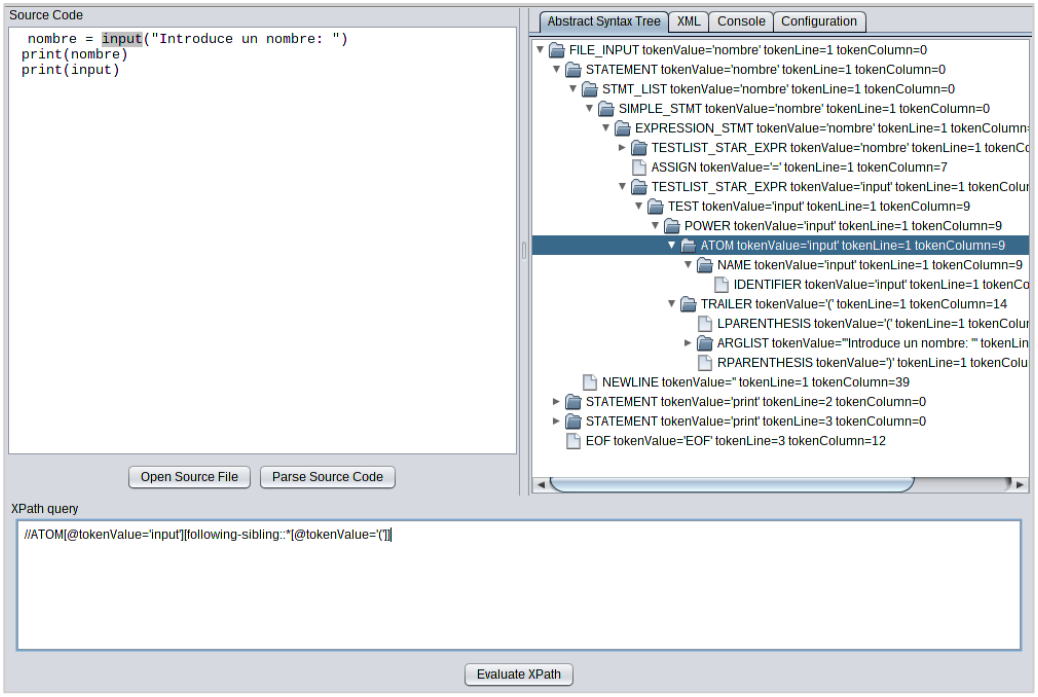

Fig. 31. Regla User Input Injection en SSLR Toolkit

En la Figura 18 se aprecia la consulta XPath que detecta esta vulnerabilidad. El código fuente examinado se compone de tres líneas: la primera guarda en la variable *nombre* el dato introducido por el usuario, la segunda imprime por consola la variable *nombre* y la tercera imprime por consola la variable *input*. Esta tercera línea se ha añadido para verificar que la consulta XPath detecta únicamente las palabras input que van seguidas de un paréntesis izquierdo, es decir, la función input().

//ATOM[@tokenValue='input'][following-sibling::[@tokenValue='(']]*

La consulta anterior selecciona cualquier nodo *ATOM*, indiferentemente de su localización en el AST, cuyo atributo *tokenValue* tiene el valor *input* y cuyos hermanos siguientes tienen un atributo *tokenValue* con un paréntesis izquierdo como valor.

Datos de entrada de la base de datos o Database Input

Si los datos de entrada de una sentencia SQL no se purgan correctamente, podría suponer un riesgo para la seguridad de la aplicación, pues el usuario podría acceder a datos de la BD, modificarlos e incluso eliminarlos.

nombre = raw_input('Introduce el nombre: ')

*cur.execute("SELECT * FROM usuarios WHERE usuario = '%s';" % nombre)*

La variable *nombre* no se purga/filtra en ningún momento, por lo que el usuario podría introducir el siguiente valor y eliminar la tabla de *usuarios* entera:

Pedro'; DROP TABLE usuarios;

Para solucionar este problema, se debe modificar la sentencia SQL de la siguiente manera para purgar el dato de entrada:

*cur.execute("SELECT * FROM usuarios WHERE usuario = %s;", (nombre,))*

Tras aclarar el problema existente, se procede a crear la regla en XPath haciendo uso de SSLR Toolkit.

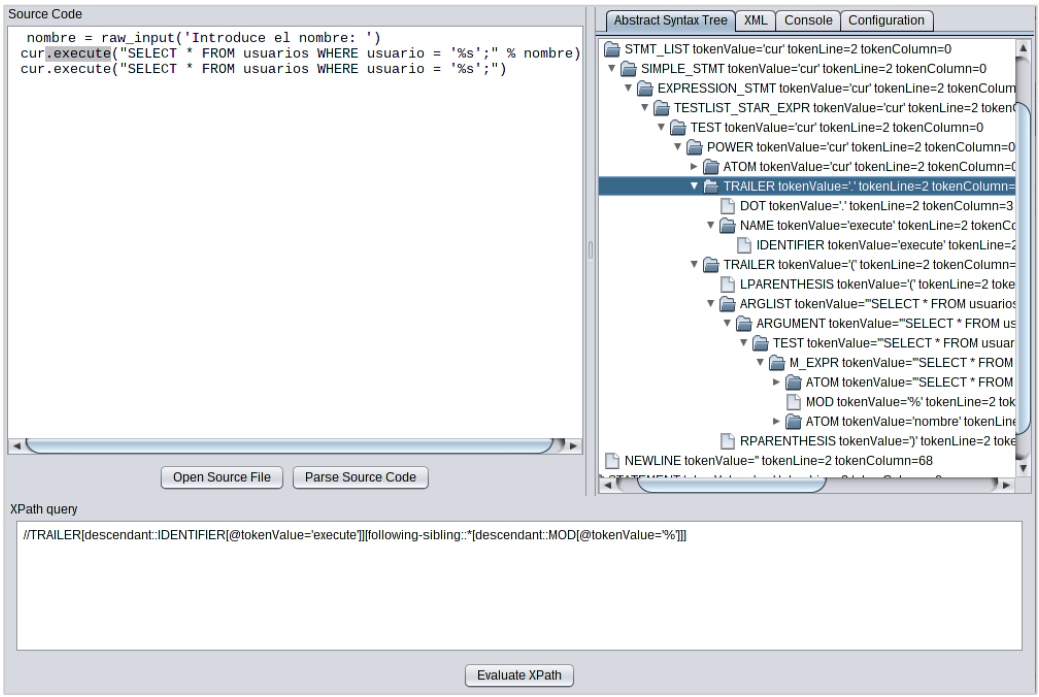

Fig. 32. Regla SQL Injection en SSLR Toolkit

En la Figura 19 se aprecia la consulta XPath que detecta esta vulnerabilidad. El código fuente examinado se compone de tres líneas: la primera guarda en la variable *nombre* el dato introducido por el usuario, la segunda ejecuta una consulta SQL con el nombre introducido por el usuario y la tercera es exactamente igual que la segunda pero evitando el signo módulo. Esta tercera línea se ha añadido para verificar que la consulta XPath detecta únicamente las consultas que toman como parámetro el dato introducido por el usuario.

//TRAILER[descendant::IDENTIFIER[@tokenValue='execute']][following-sibling::[descendant::MOD[@tokenValue='%']]]*

La consulta anterior selecciona cualquier nodo *TRAILER*, indiferentemente de su localización en el AST, cuyos descendientes de tipo *IDENTIFIER* tienen un atributo *tokenValue* con valor *execute*, y cuyos hermanos siguientes tiene descendientes de tipo *MOD* con un atributo *tokenValue* y un signo *%* como valor de ese atributo.

Una vez generadas las consultar XPath con la herramienta SSLR Toolkit, solo queda crear dos reglas, usando la plantilla adecuada, dentro del perfil de calidad (‘*Modulo5*’) en la interfaz web de SonarQube. Para hacerlo, se deben seguir las siguientes instrucciones:

- Abrir la vista *Quality Profiles*.

- Seleccionar el perfil de calidad creado anteriormente (‘*Modulo5*’).
- Seleccionar la opción ‘*Activate More’*.
- Seleccionar la regla ‘*Track breaches of an XPath rule*’.
- Seleccionar la opción ‘*Create*’ en el apartado inferior.
- Rellenar el formulario con los siguientes datos:
    - *Name*. Descripción corta del problema.
    - *Key*. Identificación única de la regla. Se puede dejar el nombre como valor.
    - *Description*. Explicación extensa del problema que se intenta detectar/evitar.
    - *Type*. Seleccionar el tipo de problema (bug, vulnerabilidad o code smell).
    - *Severity*. Seleccionar la gravedad del problema.
    - *Status*. Seleccionar si la regla está lista para ser usada, es una versión beta o está obsoleta.
    - *message*. A qué línea de código corresponde la consulta en XPath.
    - *xpathQuery*. Consulta escrita en XPath que identifica el problema dentro del código.
- Confirmar la creación de la regla.
- Seleccionar la nueva regla en la lista de reglas.
- Seleccionar la opción ‘*Activate*’, situada en el apartado ‘*Quality Profiles*’ de la regla, y confirmar la activación de la regla en el perfil creado.

Para la primera consulta Xpath creada, el formulario se rellenaría como sigue:

Name: input() no debería usarse en Python 2.X

Key: se rellena automáticamente con Name

Description: La función input() no debería usarse en Python 2.X (en Python 3.X sí se puede usar) porque el dato que se introduce por entrada se considera como si fuese el nombre de una variable, por lo que el usuario podría tener acceso al valor de las variables de la aplicación o podría invocar funciones.

Noncompliant Code Example

``nombre = input(‘Introduce un nombre: ’) ``

``print(nombre)``

Compliant Solution

``nombre = raw_input(‘Introduce un nombre: ’) ``

``print(nombre)``

Type: Vulnerability

Severity: Major

Status: Ready

message: input()

xpathQuery://ATOM[@tokenValue='input'][following-sibling::*[@tokenValue='(']]

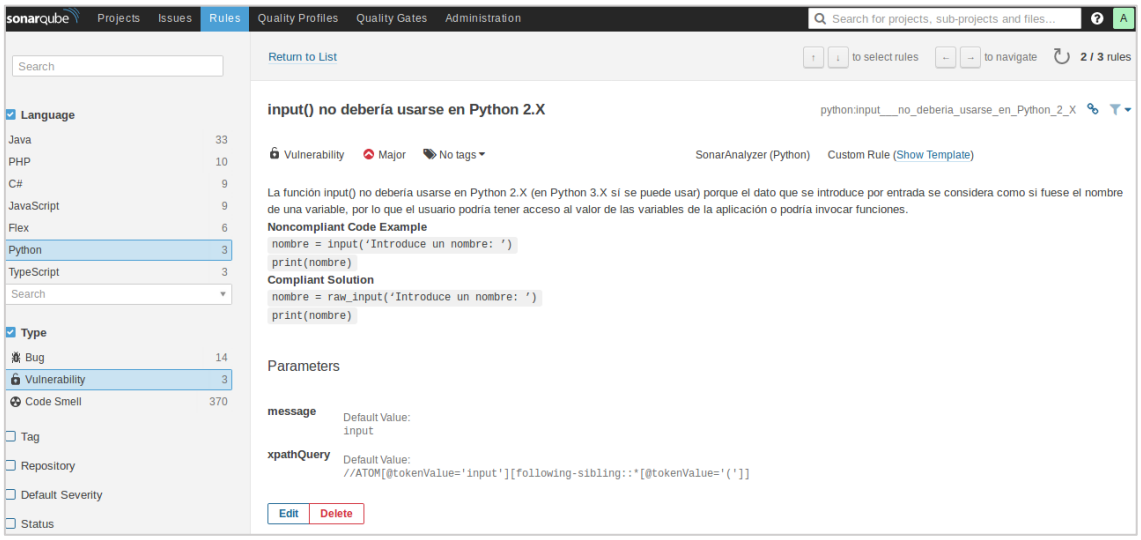

Fig. 33. Regla User Input Injection en SonarQube

En la Figura 20 se aprecia la primera regla una vez que ha sido agregada al sistema y activada en el nuevo perfil de calidad.

Para la segunda consulta Xpath creada, el formulario se rellenaría como sigue:

Name: el parámetro de execute() debe ser purgado

Key: se rellena automáticamente con Name

Description: Los datos de entrada que se usan como parámetros en las consultas SQL deben ser purgados/filtrados para evitar que el usuario pueda realizar cualquier acción dentro de la base de datos u obtenga cualquier dato de manera indebida.

Noncompliant Code Example

``nombre = raw_input('Introduce el nombre: ')``

``cur.execute("SELECT * FROM usuarios WHERE usuario = '%s';" % nombre)``

Compliant Solution

``nombre = raw_input('Introduce un nombre: ')``

``cur.execute("SELECT * FROM usuarios WHERE usuario = %s;", (nombre,))``

Type: Vulnerability

Severity: Major

Status: Ready

message: .execute()

xpathQuery://TRAILER[descendant::IDENTIFIER[@tokenValue='execute']][following-sibling::*[descendant::MOD[@tokenValue='%']]]

Fig. 34. Regla SQL Injection en SonarQube

En la Figura 21 se aprecia la segunda regla una vez que ha sido agregada al sistema y activada en el nuevo perfil de calidad.

Una vez finalizado este módulo, se considera que el lector de este documento tiene los conocimientos necesarios para manejar sin problemas tanto SonarQube como SonarQube Scanner, analizar cualquier proyecto, evaluar los resultados, actuar en consecuencia y seguir adquiriendo experiencia en esta plataforma.

NOTA:

SSLR Python Toolkit

http://downloads.sonarsource.com/plugins/org/codehaus/sonar-plugins/python/sslr-python-toolkit/1.5/sslr-python-toolkit-1.5.jar

W3Schools

https://www.w3schools.com/xml/xpath_intro.asp

ZVON

http://www.zvon.org/xxl/XPathTutorial/General_spa/examples.html

5. EVALUACIÓN

5.1. Método de evaluación

Una vez terminada la redacción y revisión del curso, y con el objetivo de evaluar la solución diseñada, se ha elaborado una encuesta para conocer la opinión algunos posibles futuros alumnos de este curso.

En esta encuesta han participado un total de tres usuarios, todos ellos con un perfil de ingeniero informático. Además, todos han tenido contacto y conocen el lenguaje Python, requisito imprescindible pues los proyectos que se analizan en el curso están escritos en este lenguaje.

Los tres usuarios han recibido un documento, en formato pdf, que contenía el capítulo cuatro y el anexo con las respuestas de la autoevaluación. Así mismo, se les ha proporcionado un documento, en formato txt, con el contenido mostrado en la Figura 35.

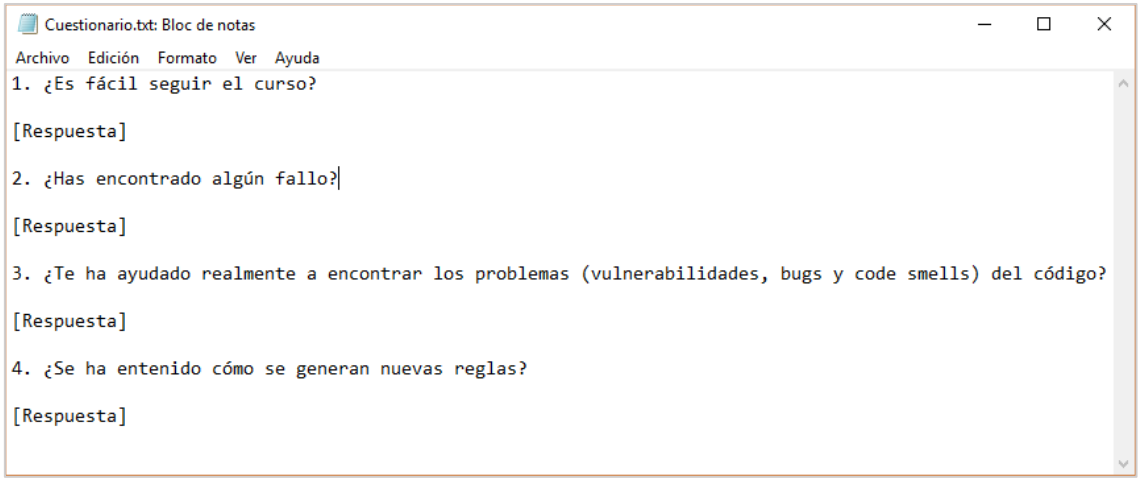

Fig. 35. Cuestionario de evaluación

Se ha elegido este método de evaluación debido a que es otorga más libertad a los usuarios para explicar en detalle su respuesta y, de esta manera, poder realizar una evaluación de calidad.

5.2. Resultados de la evaluación

Una vez que se ha recibido la respuesta de todos los usuarios, se ha procedido a analizar las respuestas que han proporcionado cada uno a las cuatro preguntas del cuestionario. El análisis realizado consiste en marcar con un color verde los comentarios positivos y con un color rojo los comentarios negativos del usuario. En las Figuras 36-38 se puede observar este método de análisis:

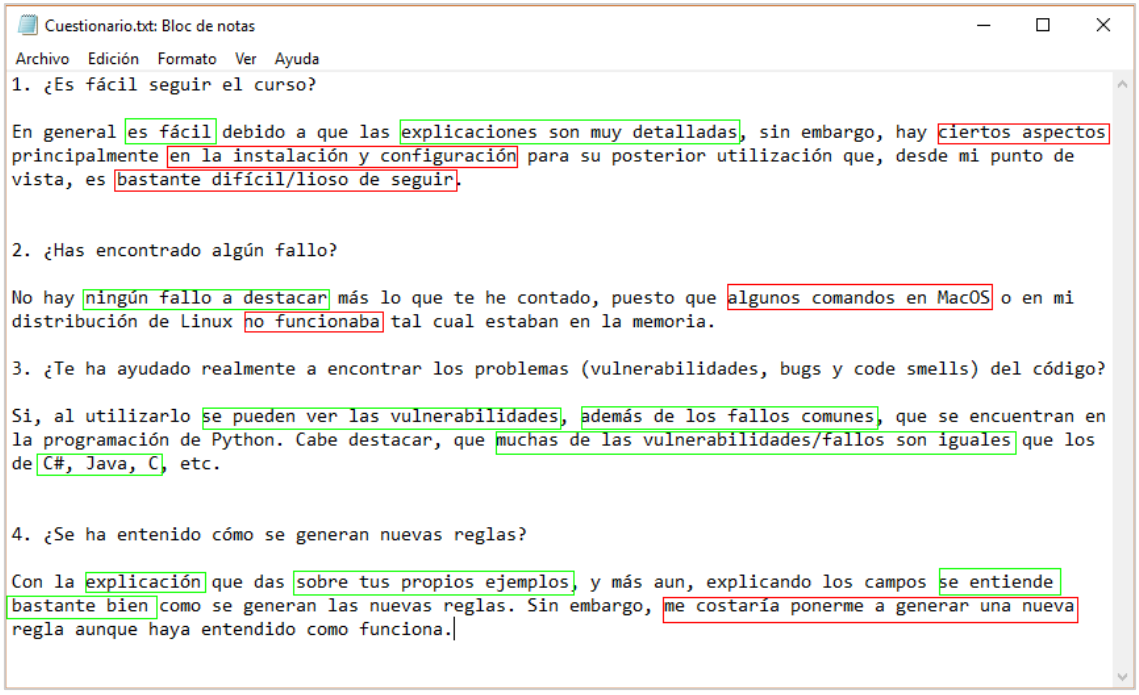

Fig. 36. Respuestas del primer usuario

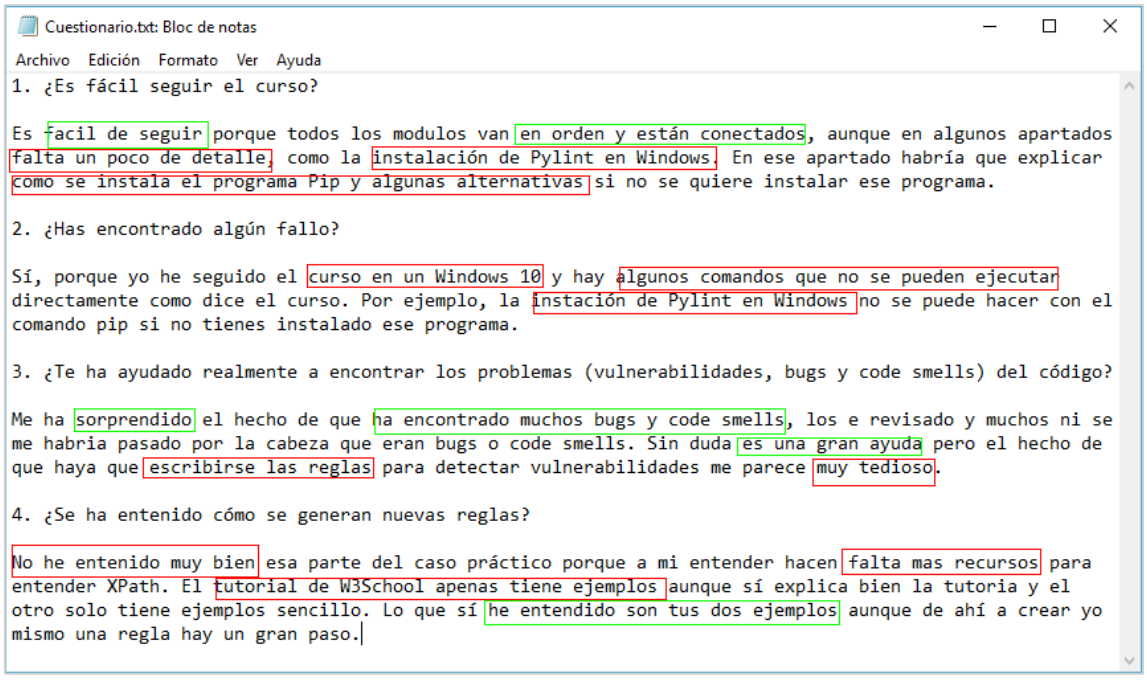

Fig. 37. Respuestas del segundo usuario

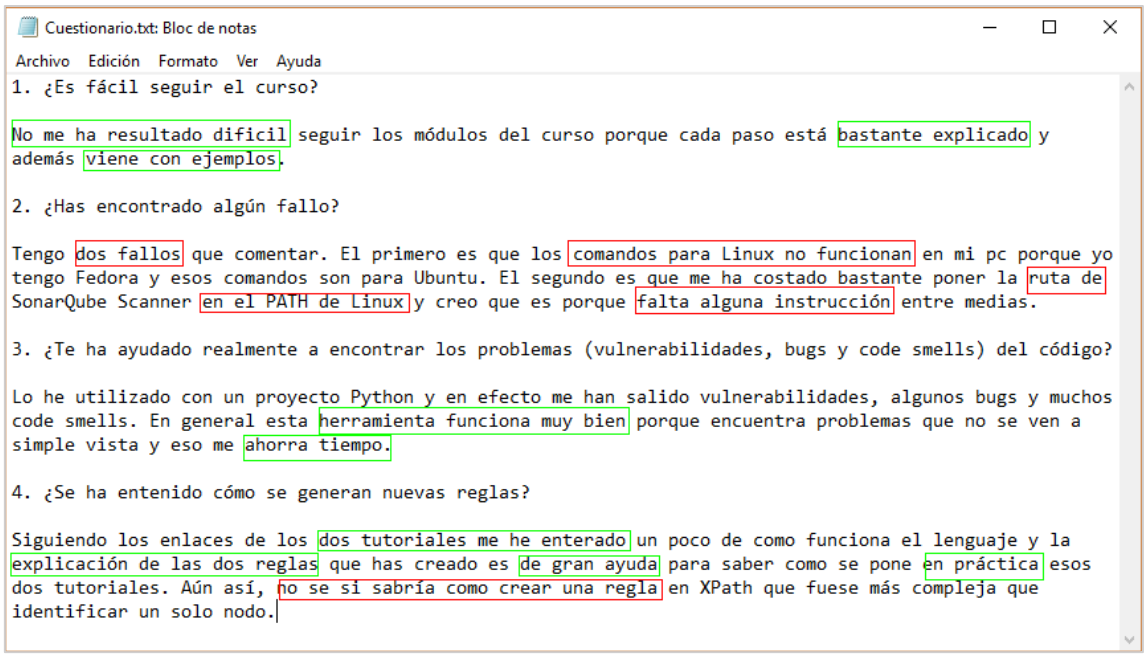

Fig. 38. Respuestas del tercer usuario

Una vez marcados los comentario positivos y negativos, se pueden juntar en una única lista para ver si existen puntos en común.

Lista de comentarios positivos:

- El curso es fácil de seguir
- Las explicaciones son muy detalladas
- Ningún fallo destacable
- Se pueden ver las vulnerabilidades, además de fallos comunes
- Muchas vulnerabilidades son iguales que en otros lenguajes como C o Java
- Con la explicación sobre los ejemplos propios se entiende bastante bien como se generan nuevas reglas
- No ha resultado difícil seguir el curso
- El curso está bastante explicado
- El curso viene con ejemplos
- La herramienta funciona muy bien
- La herramienta ahorra tiempo
- Los dos tutoriales son de ayuda para crear reglas
- La explicación de las dos reglas es de gran ayuda en la práctica
- El curso es fácil de seguir
- El curso está en orden y sus partes están conectadas
- Sorprendido por encontrar muchos bugs y code smells
- La herramienta es una gran ayuda
- Se han entendido los dos ejemplos

Lista de comentario negativos:

- Ciertos aspectos de la instalación y configuración son bastante difíciles de seguir

- Algunos comandos no funcionan en MacOS
- Cuesta ponerse a generar una nueva regla
- Comandos para Linux no funcionan en Fedora
- Ha costado poner la ruta de Scanner en el Path de Linux
- No sabría cómo crear una regla en Xpath
- Falta detalle en el curso
- Falta explicar la instalación de Pylint en Windows
- El curso en Windows 10 no funcionan comando para instalar Pylint
- Escribir reglas es tedioso
- No ha entendido muy bien cómo crear reglas
- Faltan recursos para crear reglas
- Tutoriales recomendados no sirven

Si se observa cada una de las listas anteriores se pueden sacar las siguientes conclusiones generales en cuanto a la evaluación:

- El curso en general es fácil de seguir, con instrucciones detalladas y en orden.
- La herramienta SonarQube es un buen apoyo para generar código de calidad.
- Hay que mejorar las instrucciones tanto de Linux como de Windows.
- Poner a disposición del alumno más material para aprender el lenguaje XPath.

6. PLANIFICACIÓN Y PRESUPUESTO

6.1. Planificación del proyecto

En este epígrafe se describen las fases que se han seguido para desarrollar este trabajo. Cada una de las fases está compuesta por diferentes tareas para una mejor comprensión del proceso de desarrollo del trabajo.

A continuación, se proporciona la estructura detallada donde se puede apreciar cada una de las fases, sus tareas asignadas y el número de horas necesarias para llevarlas a cabo.

1. **Documentación previa**
    1) Estudio de técnicas de prevención y descubrimiento de vulnerabilidades (20 horas)
    2) Estudio de la técnica de análisis estático de código fuente (10 horas)
    3) Estudio de las herramientas SAST disponibles (10 horas)
    4) Estudio y evaluación de SonarQube (25 horas)
2. **Diseño y elaboración del curso**
    1) Identificación de los resultados del aprendizaje (2 horas)
    2) Diseño de la estructura del curso (5 horas)
    3) Elaboración del material del curso (50 horas)
3. **Evaluación**
    1) Elaboración del cuestionario (2 horas)
    2) Distribución del cuestionario (1 hora)
    3) Análisis de los resultados (3 horas)
4. **Memoria**
    1) Elaboración de la memoria (40 horas)
    2) Revisión de la memoria (10 horas)

En la Tabla 6.4. se puede observar el número total de horas invertidas en el desarrollo de este trabajo.

TABLA 6.4.TOTAL HORAS DE DESARROLLO

Fase	Tarea	Horas	Horas/Fase
1. Documentación previa	Estudio de técnicas de prevención y descubrimiento de vulnerabilidades	20	65
	Estudio de la técnica de análisis estático de código fuente	10	
	Estudio de las herramientas SAST disponibles	10	
	Estudio y evaluación de SonarQube	25	
	Identificación de los resultados del aprendizaje	2	57

Fase	Tarea	Horas	Horas/Fase
2. Diseño y elaboración del curso	Diseño de la estructura del curso	5	
	Elaboración del material del curso	50	
3. Evaluación	Elaboración del cuestionario	2	6
	Distribución del cuestionario	1	
	Análisis de los resultados	3	
4. Memoria	Elaboración de la memoria	40	50
	Revisión de la memoria	10	
TOTAL			**178**

En las Figuras 35-37 se puede observar esta misma estructura en un diagrama de Gantt, donde se puede apreciar el desarrollo de cada tarea en el tiempo. Para una mejor visualización del diagrama, se ha separado el diagrama en tres partes.

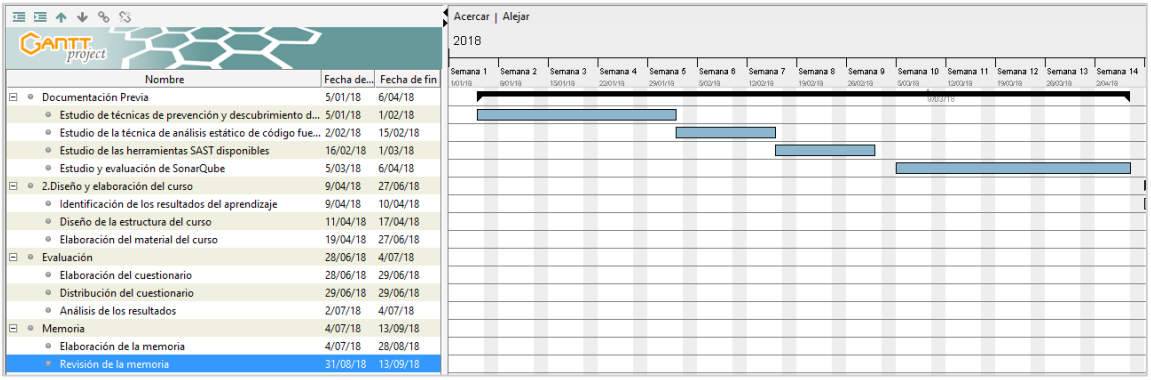

Fig. 39. Diagrama de Gantt (fase 1)

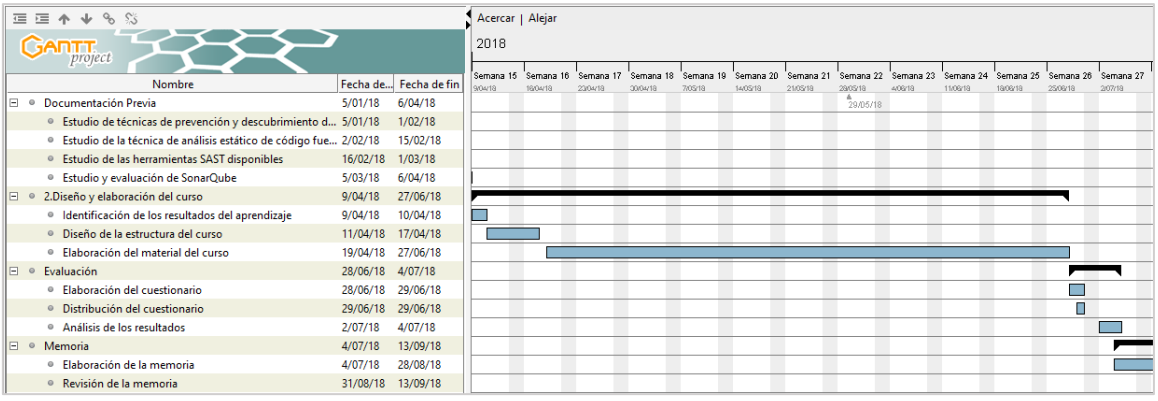

Fig. 40. Diagrama de Gantt (fase 2 y 3)

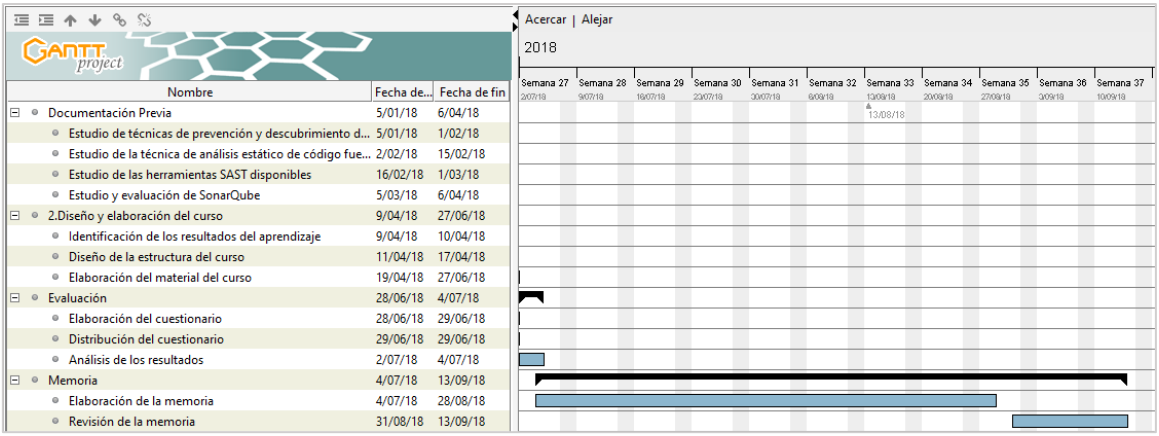

Fig. 41. Diagrama de Gantt (fase 4)

6.2. Presupuesto

En este epígrafe se detalla cada uno de los costes generados durante el transcurso de este trabajo, desde la primera fase hasta la última.

6.2.1. Coste de personal

Para llevar a cabo el desarrollo de este trabajo se requerido la implicación de un total de cinco personas, entre el autor de este, la tutora del trabajo y los tres usuarios que han probado y evaluado el curso.

- El autor de este documento se ha encargado del diseño y elaboración del curso, así como de la posterior redacción de la memoria. El cálculo se hará teniendo en cuenta las horas especificadas en Tabla 6.4.
- La tutora del trabajo ha supervisado cada una de las cuatro fases del trabajo, aportando información y conocimientos durante todo el proceso. El cálculo se hará teniendo en cuenta las horas de tutorías y el tiempo dedicado para responder a los emails. Se han estimado un total de 30 horas.
- Los tres usuarios, con un perfil de ingeniero informático, se han encargado de realizar el curso y, posteriormente, evaluarlo mediante el uso del cuestionario proporcionado. El cálculo se hará teniendo en cuenta las horas estimadas en la Tabla 4.2.

Por lo tanto, en el coste final del personal se incluirán los cinco participantes. Para calcular el coste por hora de cada participante, se ha hecho una media de los resultados arrojados por diversos motores de búsqueda de empleo en España. En la Tabla 6.5. se puede observar el resultado obtenido:

TABLA 6.5.COSTE DEL PERSONAL

Personal	Tiempo (horas)	Coste/Hora (€)	Total (€)
Autor del trabajo	178	24,58	4.375,24
Tutora del trabajo	35	35,50	1.242,50
Usuario 1	20	16,97	339,40
Usuario 2	20	16,97	339,40

Personal	Tiempo (horas)	Coste/Hora (€)	Total (€)
Usuario 3	20	16,97	339,40
TOTAL			**6.635,94**

Una vez realizados los cálculos, se puede apreciar que el coste total del personal asciende a la cantidad de 6635,94 euros.

6.2.2. Coste de hardware

En este apartado se calcula el coste que suponen todos los equipos hardware utilizados durante el desarrollo de cada una de las fases. Para calcular la amortización del equipo se hace uso de la siguiente fórmula:

$$\frac{\boldsymbol{A}}{\boldsymbol{B}}(\boldsymbol{C})(\boldsymbol{D}) \qquad (\mathbf{6.1})$$

***A**: número de meses que se ha utilizado el equipo*

***B**: periodo de depreciación (60 meses)*

***C**: coste unitario del equipo*

***D**: porcentaje de uso del hardware que se ha dedicado para el proyecto*

En la Tabla 6.6 se detallan cada uno de los equipos utilizados, su coste unitario, número de meses que se ha utilizado el equipo y el coste, calculado con la fórmula anterior, que supone para el proyecto.

TABLA 6.6.COSTE HARDWARE

Hardware	Unidades	Coste unitario (€)	Meses utilizados	Uso (%)	Total (€)
Portátil Acer Aspire	1	799,00	9	15	17,98
Ratón inalámbrico	1	10,00	9	10	0,15
SSD 120GB	1	65,94	9	14	1,38
TOTAL					**19,51**

Una vez realizados los cálculos, se puede apreciar que el coste total del hardware asciende a la cantidad de 19,51 euros.

6.2.3. Coste de software

En la Tabla 6.7 se detallan cada una de las herramientas software utilizadas durante el desarrollo del trabajo, así como el coste de las licencias.

TABLA 6.7.COSTE SOFTWARE

Software	Unidades	Coste unitario (€)	Total (€)
SonarQube	1	0,00	0,00
SonarQube Scanner	1	0,00	0,00
Pylint	1	0,00	0,00
Microsoft Windows 10	1	0,00	0,00
Elementary OS	1	0,00	0,00

Software	Unidades	Coste unitario (€)	Total (€)
Gantt Project	1	0,00	0,00
Microsoft Office	1	0,00	0,00
TOTAL			0,00

La herramienta utilizada para el análisis es de código abierto, la licencia del sistema operativo Windows y de la herramienta Microsoft Office ha sido adquirida a coste 0 gracias al acuerdo que tiene la universidad con la plataforma Microsoft Imagine, el sistema operativo Linux utilizado durante el desarrollo es de código abierto y las demás herramientas software también.

Por lo tanto, una vez realizados los cálculos, se puede apreciar que el coste total del software asciende a la cantidad de 0 euros.

6.2.4. Coste indirecto

En este apartado se calcula el coste que supone el gasto de electricidad de los equipos hardware utilizados durante los nueve meses que ha durado el proyecto.

Para calcular el gasto de electricidad se hace uso de la siguiente fórmula:

$$\frac{(\boldsymbol{A})(\boldsymbol{B})}{\mathbf{1000}}(\boldsymbol{C}) \qquad (\mathbf{6.2})$$

***A**: potencia en Wh consumida por el equipo*

***B**: número de horas que se ha utilizado el equipo*

***C**: precio en euros por cada kWh consumido*

En la Tabla 6.8 se detallan cada uno de los equipos hardware utilizados, la potencia que consumen por hora, el número de horas que se ha utilizado el equipo y el coste total indirecto.

TABLA 6.8.COSTE INDIRECTO

Hardware	Potencia (Wh)	Uso (horas)	Precio (€/kWh)	Total (€)
Portátil Acer Aspire	73,44	178	0,12237	1,60
Ratón inalámbrico	1	178	0,12237	0,02
SSD 120GB	2,5	178	0,12237	0,05
TOTAL				1,68

Una vez realizados los cálculos, se puede apreciar que el coste total indirecto asciende a la cantidad de 1,68 euros.

6.2.5. Coste de material fungible

En este apartado se calcula el coste que supone el gasto de material fungible durante el proyecto.

En la Tabla 6.9 se detallan cada uno de los materiales utilizados durante el desarrollo del trabajo, el número de unidades y el precio por unidad.

TABLA 6.9.COSTE MATERIAL

Material	Unidades	Coste unitario (€)	Total (€)
Hojas A4	30	0,0058	0,17
Bolígrafo	1	0,50	0,50
TOTAL			0,67

Una vez realizados los cálculos, se puede apreciar que el coste total del material asciende a la cantidad de 0,67 euros.

6.2.6. Coste total

En este apartado se calcula el coste total que ha supuesto el desarrollo de las cuatro fases del trabajo.

TABLA 6.10.COSTE TOTAL

Coste parcial	Total (€)
Coste de personal	6635,94
Coste de hardware	19,51
Coste de software	0,00
Coste indirecto	1,68
Coste de material fungible	0,67
TOTAL	6657,80

Una vez realizados los cálculos, se puede apreciar que el coste total del proyecto asciende a la cantidad de 6657,80 euros.

7. CONCLUSIONES

7.1. Objetivos cumplidos

En este epígrafe se hace un análisis para conocer qué objetivos, de los planteados inicialmente, se han cumplido una vez finalizada la parte de desarrollo y evaluación del trabajo.

En el primer capítulo de este documento se plantearon varios objetivos con respecto a lo que se pretendía conseguir con este trabajo, aunque se pueden resumir en dos grandes objetivos. A continuación, se analiza cada uno ellos:

"…crear un documento en el que se proporcionen las instrucciones necesarias para conocer en detalle las características de una herramienta de análisis estático de código, su funcionamiento básico y su aplicación a un proyecto real."

En lo que respecta a este objetivo, se puede considerar cumplido pues así lo avalan las conclusiones extraídas del análisis que se ha realizado a las respuestas de evaluación recibidas. Dos de las conclusiones generales extraídas en el epígrafe 5.2 confirman que se ha conseguido crear un documento con instrucciones ordenadas y bien detalladas que ayudan al usuario a conocer el funcionamiento de una herramienta de análisis estático de código. Así mismo, las conclusiones generales también indican que los usuarios que siguen el material son capaces de ver los beneficios de usar la herramienta en cualquier entorno.

"… se proporcionará la documentación necesaria para hacer un uso avanzado de las funcionalidades de este tipo de herramientas."

En lo que respecta a este objetivo, las conclusiones generales extraídas en el epígrafe 5.2 imposibilita considerar este objetivo cumplido. La razón es que todos los usuarios que han evaluado el curso coinciden en afirmar que se necesita material adicional para aprender a crear reglas personalizadas. Esta afirmación imposibilita, por tanto, considerar el objetivo como cumplido puesto que la creación de reglas es una funcionalidad avanzada que intenta enseñarse en el curso.

7.2. Líneas futuras de trabajo

Tras realizar la evaluación y la revisión de objetivos, se ha llegado a algunas conclusiones sobre posibles mejoras del curso e incluso nuevos cursos. A continuación, se describen algunas ideas sobre los trabajos que se podrían realizar en un futuro:

- Arreglar los errores cometidos en las instrucciones del curso, como la diferenciación entre las diferentes distribuciones Linux o la explicación de cómo instalar Pylint en Windows.
- Ampliar el curso con instrucciones para integrar SonarQube en entornos de desarrollo integrado, como Eclipse, o en software de integración continua, como Jenkins.

- Ampliar el apartado de creación de reglas personalizadas con información adicional externa o propia, o incluso crear un curso adicional para aprender el lenguaje XPath en profundidad.
- Ampliar el curso con casos prácticos adicionales para que los alumnos puedan afianzar y adquirir nuevos conocimientos.
- Crear un plugin donde se incluyan reglas para detectar vulnerabilidades particulares de Python.

7.3. Impacto socio-económico

Para analizar el impacto socio-económico que tendrá el uso de este trabajo hay que tener en cuenta todos los estudios mencionados en el capítulo *Estado de la cuestión* y los resultados de estos estudios.

Como ya se ha comentado, en los últimos años una considerable cantidad de organizaciones han sido víctimas de ataques cibernéticos, como es el caso de Yahoo, Ebay, Anthem y Sony PlatStation Network.

Así mismo, también se ha indicado que según el Centro de Recursos para el Robo de Identidad [11], en los últimos cuatro años se han producido más de 9395 ataques cibernéticos, llegando a perder cerca de 1 billón de registros con información confidencial.

Por otro lado, según el último informe del Ponemon Institute [13], el número de ataques aumenta cada año un 27.4%, lo que ha provocado que el coste de la seguridad aumente un 22.7% en el último año hasta situarse en unos 11.7 millones de dólares.

Teniendo en cuenta estos datos, puede afirmarse que este trabajo tiene un impacto directo e influye positivamente tanto en la sociedad como en la economía.

Por un lado, mediante la realización del curso diseñado, el usuario tendrá los conocimiento y capacidades necesarias para identificar los bugs y vulnerabilidades de seguridad existentes en un sistema software y corregirlos, reduciendo de esta manera el impacto de un posible ataque cibernético o incluso evitándolo.

De esta manera, si el usuario trabaja actualmente, o en el futuro, como programador de sistemas software, podrá aplicar estos conocimientos y capacidades en su entorno laboral, evitando que su organización tenga que invertir dinero en medidas de seguridad software o en resolver brechas de seguridad ya explotadas por los atacantes.

Por otro lado, si el usuario trabaja en su entorno laboral con datos confidenciales de pacientes o consumidores, aplicando estos conocimientos y capacidades podrá evitar que los atacantes puedan comprometer la confidencialidad de esos datos y por tanto a los propios pacientes o consumidores.

BIBLIOGRAFÍA

[1] "Yahoo Discloses 2013 Breach that Exposed Over One Billion Accounts," 2016. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/yahoo-discloses-2013-breach-exposed-over-1billion-accounts.

[2]. "*EQUIFAX'S STATEMENT FOR THE RECORD REGARDING THE EXTENT OF THE CYBERSECURITY INCIDENT ANNOUNCED ON SEPTEMBER 7, 2017*" 2017. [Online].
Available: https://www.sec.gov/Archives/edgar/data/33185/000119312518154706/d583804dex991.htm.

[3] "Equifax Reveals Extent of 2017 Data Breach, Details Number of Stolen Records," 2018. [Online]. Available: https://www.trendmicro.com/vinfo/ph/security/news/cyber-attacks/equifax-reveals-extent-of-2017-data-breach-number-of-stolen-records.

[4] C. Riley, "Insurance giant Anthem hit by massive data breach," 2015. [Online]. Available: https://money.cnn.com/2015/02/04/technology/anthem-insurance-hack-data-security/.

[5] "California Department of Insurance," vol. 66, no. 9, p. 33, Mar. 2012.

[6] M. Kolbasuk McGee, "A New In-Depth Analysis of Anthem Breach," 2017. [Online]. Available: https://www.bankinfosecurity.com/new-in-depth-analysis-anthem-breach-a-9627.

[7] "US OPM Hack Exposes Data of 4 Million Federal Employees," 2015. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/us-opm-hack-exposes-data-of-4-million-federal-employees.

[8] D. Chiacu and M. Spetalnick, "Cyber attack hits 4 million current, former U.S. federal workers," 2015. [Online]. Available: https://www.reuters.com/article/us-cybersecurity-usa/cyber-attack-hits-4-million-current-former-u-s-federal-workers-idUSKBN0OK2IK20150604?feedType=RSS&feedName=technologyNews.

[9] K. McCarthy, "5 Colleges With Data Breaches Larger Than Sony's in 2014," 2015. [Online]. Available: https://www.huffingtonpost.com/kyle-mccarthy/five-colleges-with-data-b_b_6474800.html?guccounter=1.

[10] "Privacy Rights Clearinghouse," 2018. [Online]. Available: https://www.privacyrights.org/data-breaches.

[11] "Identity Theft Resource Center," 2018. [Online]. Available: https://www.idtheftcenter.org/data-breaches/.

[12] N. Huq, "Follow the Data: Analyzing Breaches by Industry," 2015. [Online]. Available: https://www.trendmicro.de/cloud-content/us/pdfs/security-intelligence/white-papers/wp-analyzing-breaches-by-industry.pdf.

[13] "2017 COST OF CYBER CRIME STUDY," 2018. [Online]. Available: https://www.accenture.com/us-en/insight-cost-of-cybercrime-2017.

[14] P. Black, M. Kass, and M. Koo, “Source Code Security Analysis Tool Functional Specification Version 1.0,” 2007. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-268.pdf.

[15] “Software Fail Watch 5th Edition,” 2018. [Online]. Available: https://tricentis-com-tricentis.netdna-ssl.com/wp-content/uploads/2018/02/20180207_Software-Fails-Watch.pdf.

[16] L. Futcher and R. von Solms, "SecSDM: A model for integrating security into the software development life cycle," in *Fifth World Conference on Information Security Education*Anonymous 2007, . DOI: 10.1007/978-0-387-73269-5_6.

[17] “Application Security Testing and Static Code Analysis,” 2018. [Online]. Available: ttps://www.checkmarx.com.

[18] K. Goseva-Popstojanova and A. Perhinschi, "On the capability of static code analysis to detect security vulnerabilities," *Information and Software Technology,* vol. 68, pp. 18-33, 2015. Available: https://www.sciencedirect.com/science/article/pii/S0950584915001366. DOI: 10.1016/j.infsof.2015.08.002.

[19] W. R. Jimenez Freitez, A. Mammar and A. R. Cavalli, "Software vulnerabilities, prevention and detection methods : A review," in 2009, Available: https://hal.archives-ouvertes.fr/hal-01367445.

[20] “Security Code Review in the SDLC,” 2010. [Online]. Available: https://www.owasp.org/index.php/Security_Code_Review_in_the_SDLC.

[21] “VeraCode,” 2018. [Online]. Available: https://www.veracode.com/.

[22] “Kiuwan,” 2018. [Online]. Available: https://www.kiuwan.com/.

[23] “MicroFocus,” 2018. [Online]. Available: https://software.microfocus.com.

[24] “Interactive: The Top Programming Languages 2018,” 2018. [Online]. Available: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018.

[25] “TIOBE Index for September 2018,” 2018. [Online]. Available: https://www.tiobe.com/tiobe-index/.

[26] “SonarQube,” 2018. [Online]. Available: https://www.sonarqube.org/.

[27] R. van Riel and P. Morreale, “Documentation for Kernel Linux,” 2008. [Online]. Available: https://www.kernel.org/doc/Documentation/sysctl/vm.txt.

[28] “Virtual Memory.” [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/6.0/vm-max-map-count.html.

[29] G. Mourani, “Securing and Optimizing Linux: RedHat Edition -A Hands on Guide,” 2000. [Online]. Available: http://www.faqs.org/docs/securing/chap6sec72.html.

[30] M. Kerrisk, “Linux/UNIX system programming training,” 2018. [Online]. Available: http://man7.org/linux/man-pages/man3/ulimit.3.html.

[31] “System resource limits and the ulimit command,” 2018. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSYKE2_7.0.0/com.ibm.java.aix.70.doc/user/ulimits.html.

[32] “Number of threads.” [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/5.6/max-number-of-threads.html.

[33] “OWASP Top 10 - 2017 Los diez riesgos más críticos en Aplicaciones Web,” 2017. [Online]. Available: https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf.

ANEXO A. GLOSARIO

CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
GCJ	GNU Compiler for Java
JAR	Java ARchive
JVM	Java Virtual Machine
IDE	Integrated Development Environment
LTS	Long-Term Support
MOOC	Massive Open Online Course
NSS	Número de la Seguridad Social
OWASP	Open Web Application Security Project
RAM	Random Access Memory
SaaS	Software as a Service
SDLC	Systems Development Life Cycle
SEC	U.S. Securities and Exchange Commission
SO	Sistema Operativo
SSLR	SonarSource Language Recognizer
XSS	Cross-Site Scripting

ANEXO B. RESPUESTAS AUTOEVALUACIÓN

Módulo 1. Requisitos previos

1. a.

Los parámetros modificados en la sección dos del primer módulo representan la posibilidad de que un proceso pueda trabajar con una cantidad u otra de memoria, ficheros y/o threads.

vm.max_map_count. De acuerdo con la documentación tanto del kernel de Linux [27] como la proporcionada por Elastic [28], este parámetro representa el número máximo de áreas del mapa de memoria del que puede disponer un proceso.

fs.file-max. De acuerdo con la documentación de Faqs [29], este parámetro representa el número máximo de descriptores de fichero abiertos a nivel de sistema.

ulimit -n. De acuerdo con la documentación tanto de Linux [30] como la proporcionada por IBM sobre la ejecución de aplicaciones Java [31], este parámetro representa el número máximo de descriptores de fichero abiertos a nivel de usuario.

ulimit -u. De acuerdo con la documentación proporcionada por Elastic [32], este parámetro representa el número máximo de threads que puede crear un proceso.

1. b.

Sí, existe una forma de fijar el valor de dichos parámetros de forma permanente y para hacerlo se deben seguir los siguientes pasos:

1. Para fijar los dos primeros parámetros (*vm.max_map_count*, *fs.file-max*) se debe editar el fichero */etc/sysctl.d/99-sysctl.conf*, o en su defecto */etc/sysctl.conf*, y añadir las siguientes líneas al final del mismo:

   sysctl -w vm.max_map_count=262144

   sysctl -w fs.file-max=65536

2. Para fijar los dos últimos parámetros (*ulimit -n*, *ulimit -u*) se debe editar el fichero */etc/limits.conf*, o en su defecto */etc/security/limits.conf*, y añadir las siguientes líneas al final de este:

   sonarqube - nofile 65536

   sonarqube - nproc 2048

2. a.

Sí, existe una característica, utilizada por un componente de SonarQube, denominada *seccomp filter* (SECure COMPuting with filters). Esta característica está activada, en el kernel, en la mayoría de las distribuciones Linux, aunque en algunas como Red Hat Linux 6 está desactivada. Para saber si esta característica está desactivada en el sistema y, en caso afirmativo, como activarla, se deben seguir los siguientes pasos:

1. Abrir un terminal y ejecutar el siguiente comando:

```
grep SECCOMP /boot/config-$(uname -r)
```

   Si, tras ejecutar el comando, el terminal imprime las siguientes líneas, se debe omitir el paso 2:

   CONFIG_HAVE_ARCH_SECCOMP_FILTER=y

   CONFIG_SECCOMP_FILTER=y

   CONFIG_SECCOMP=y

2. Abrir el fichero *sonar.properties*, situado en el directorio */opt/sonarqube/conf/*, y modificar el siguiente parámetro:

   sonar.search.javaAdditionalOpts=-Dbootstrap.system_call_filter=false

Módulo 2. SonarQube

1. a.

Dentro del directorio raíz de SonarQube existe un subdirectorio denominado *'logs'* en el que se puede consultar una serie de registros, entre los que destacan los siguientes:

sonar.log. Contiene información sobre el estado del arranque y la finalización del proceso principal.

web.log. Contiene información sobre la conexión inicial con la base de datos, la migración de esta y el proceso de reindexado.

ce.log. Contiene información sobre el procesamiento de las tareas ejecutadas en segundo plano.

es.log. Contiene información sobre las operaciones del motor de búsqueda, cambios en el estado de la instancia, operaciones a nivel de clúster, etc.+

Módulo 3. SonarQube Scanner

1. a.

En efecto, existe una alternativa a tener que crear el fichero *sonar-project.properties* dentro del directorio raíz del proyecto. Esta alternativa pasa por especificar las propiedades, del proyecto a analizar, directamente en el terminal. Por lo tanto, lo único que se debe hacer es abrir un terminal y ejecutar el siguiente comando:

```
sonar-scanner -Dsonar.projectKey=myproject -Dsonar.projectName=MyProject -Dsonar.projectVersion=1.0 -Dsonar.sources=.
```

2. a.

El problema más recurrente es recibir un *Java heap space error* o *java.lang.OutOfMemoryError* debido a una cantidad insuficiente de memoria asignada a

la ejecución de SonarQube Scanner. Para solucionar este problema, se deben seguir los siguientes pasos:

 # Windows # - Seguir los siguientes pasos:

- Agregar la variable *SONAR_SCANNER_OPTS* a las variables de entorno del sistema asignando el siguiente valor:

```
-Xmx512m
```

 # Linux # - Seguir los siguientes pasos:

- Editar el fichero *~/.bash_profile* (o *~/.profile*) o el fichero */etc/profile*, para incorporar la siguiente variable al entorno del S.O.:

```
export SONAR_SCANNER_OPTS="-Xmx512m"
```

Módulo 4. Interfaz de usuario

1. a.

Para cambiar la contraseña de la cuenta con la que se inicia sesión en la interfaz de SonarQube, se deben seguir los siguientes pasos:

1. Iniciar sesión en la interfaz de SonarQube.
2. Acceder a la vista de *Administration*.
3. Seleccionar la opción *Security* y posteriormente *Users*.
4. Elegir la cuenta, cuya contraseña se quiere cambiar, y seleccionar la opción *Change Password* (icono representado por un candado).
5. Rellenar los datos requeridos y confirmar el cambio de contraseña.

2. a.

Para actualizar la funcionalidad (en este caso un plugin) de SonarPython, se deben seguir los siguientes pasos:

1. Iniciar sesión en la interfaz de SonarQube.
2. Acceder a la vista de *Administration*.
3. Seleccionar la opción *Marketplace.*
4. Filtrar los resultados por las funcionalidades que tienen actualizaciones (*Updates Only*).
5. Localizar la funcionalidad en la lista de resultados.
6. Seleccionar la opción *Update* de la funcionalidad.
7. Esperar a que la actualización finalice y seleccionar la opción *Restart* para que la interfaz haga efectiva la nueva funcionalidad.

ANEXO C. RESUMEN EN INGLES

ABSTRACT

The continuous evolution of technology, added to the growing size and complexity of software systems, has taken the development of software products to a point where it is difficult to ensure its reliability. As a consequence of this trend, software security has become an essential requirement in the development process, even becoming an important factor in determining whether the code of a software system is of quality.

Throughout this document we will analyze one of the possible techniques that can be applied to guarantee, as far as possible, the security and therefore the quality of a software system. Also, sufficient didactic material will be provided to start using one of the tools that apply this technique known as static source code analysis.

ACKNOWLEDGEMENTS

Someone once said that life is a series of rooms and those with whom we agree in them shape our lives. With this analogy well present, and the time has come to close this chapter of my life, I realize the great importance that this stage has had both in my personal life and in my professional career.

There are no words that can express my gratitude to all those who have accompanied and helped me during this journey, so I will try to do justice to them with those that do exist.

With the only words I have been able to find, I want to dedicate this work to my family, who have always been by my side and have sacrificed for me so many times, to my fellow students with whom I have shared so many experiences, to the tutor who has supported me in the final stretch of this journey and to all the teachers I have had the good fortune to meet.

INTRODUCTION

Work Motivation

Today's economy relies heavily on computer systems and networks, and many services, such as finance, e-commerce, transport, energy and healthcare, cannot function without these systems. The growth of online commerce, related transactions, as well as the increasing volume of confidential information accessible online, have led to an increase in cyberattacks.

Currently, to develop a system, the SDLC or System Development Life Cycle is applied. This method enables the development of an IS or Information System, offering support so that the design and implementation processes can be planned by the project manager, specifying the investment time of the budget and the development time. Although it is a model that covers the life cycle of a system, from the moment the idea is put forward until its expiration date, it does not include a specific cycle for security.

Considering these facts, it seems more than evident the urgent need to take into account the safety of any product or system that is under development. This security must be applied during the development itself, and not after its completion, in order to have the opportunity, in the face of any security problem, to apply a solution commensurate with its seriousness.

Objectives

As mentioned in the previous section, the continuous evolution of technology means an advance in the ease of use of this technology, but it also means an increase in security problems in most cases. A small failure in the security of a system could lead to a massive leakage of sensitive information.

The purpose of the proposal described in this document is to prepare instructive material to provide a viable solution and that any user, who is developing a system, avoids, as far as possible, the security problems present in the system. These security problems range from misuse of the programming language, through bugs, to vulnerabilities. These vulnerabilities are the gateway to any cyber attack, so the proposal described focuses on this particular type of problem.

Therefore, one of the objectives of this work is to create a document that provides the necessary instructions to know in detail the characteristics of a static code analysis tool, its basic operation and its application to a real project. In addition, the necessary documentation will be provided to make advanced use of the functionalities of this type of tool.

Regulatory Framework

There are different technical norms and standards, applicable to this work, that mark the way of using some appropriate measures for the security of a system and the protection of sensitive data. Some of these norms and standards are detailed below:

The ISO/IEC 27000 family of standards. This family of standards, developed by the International Organization for Standardization in collaboration with the International

Electrotechnical Committee, helps organizations keep information secure. The use of these standards makes it easier for the organization to manage information security, such as financial information, intellectual property, employee data, or third party information. For example, ISO/IEC 27001, the most widely known international standard, establishes the requirements that a system that manages information security must meet.

Applying this set of rules to the environment of a static code analysis tool prevents any user other than the developer or project manager from having access to the administration of the tool and from knowing the detected vulnerabilities that have not yet been resolved.

CISQ. The Consortium for Software Quality in Information Technologies is a leadership group that develops international standards to automate the process of measuring the size and quality of software systems.

These standards are used to manage the properties of security, reliability, performance efficiency and risk maintainability of a software system. The security standard is based on the most widespread and exploited security vulnerabilities today, such as those found in OWASP Top 10, CWE, and SANS Top 25.

Applying these standards to the environment of a static code analysis tool improves the detection of bugs, vulnerabilities and bad programming habits, as well as strengthens the quality standards of the project.

RFC 2196. Memorandum published by the Internet Engineering Working Group, better known as IETF, to develop procedures and security policies for information systems that are connected to the Internet. This document provides an overview of network and information security, the response to possible conflicts and the security policies to be applied.

Applying these security policies and procedures to the environment of this work achieves a more realistic view of the possible situations that may occur in a work environment, such as the procedure to follow if the system has a security breach or is under cyber attack.

BACKGROUND

Current situation

Technology has never been more useful or more dangerous for our information, identity and assets than it is today. This technology is the basis of all services in the tertiary and quaternary sector of the economy and is therefore the main target of all cyber attacks that occur today. Important services such as information, business, health care, government, military and education have been compromised in recent years due to security breaches in their systems. Some of these examples are:

Information. In September 2016, Yahoo! Inc. revealed that it had suffered a security breach in 2014, in which more than 500 million user accounts were stolen. A month later, in December 2014, this same company revealed that they had suffered another security breach, in August 2013, in which around one billion user accounts were exposed [1]. Taking advantage of this security gap, attackers obtained confidential data such as user names, email addresses, phone numbers, unencrypted passwords, clear security questions and answers.

Business. The company Equifax, which collects and manages consumer credit reports, informed the U.S. Securities and Exchange Commission [2] in September 2017 of the scope of the cyber attack they had suffered in May 2017. According to this report, the attackers stole sensitive data from nearly 150 million U.S. consumers and 16 million British consumers [3]. These include name, date of birth, NSS, mailing address and bank details.

Medical care. The second-largest U.S. health insurance company, Anthem Insurance Companies, saw data for approximately 80 million of its customers compromised in February 2015 [4]. Seven state insurance commissioners produced a report [5] in December 2016 detailing the cause of the security breach and the severity of the attack in terms of confidential stolen data. These commissioners reached an agreement with the insurance company urging the company to invest significantly in security. As a result of this problem, the company has invested more than 260 million dollars in security measures [6].

Government. In June 2015, the U.S. Office of Personnel Management announced that it had been the victim of a massive information leak involving 4 million government employees [7]. An internal U.S. police source told a media outlet [8] that the government had reason to believe that the attack came from a foreign entity or government.

Education. According to information published in the Huffington Post [9], in March 2014 the University of Maryland suffered the theft of more than 300,000 student, faculty and employee records [9]. According to the Privacy Rights Clearinghouse [10], there have been 854 security breaches and more than 25 million records of stolen data in the U.S. educational service alone since 2005.

The Ponemon Institute [13], in collaboration with Accenture, produces an annual report in which they evaluate the responses of 2182 interviewees from 254 companies in seven countries, namely: Australia, France, Germany, Italy, Japan, the United Kingdom and the

United States. The purpose of this report is to explain the impact of cybercrime on a country or sector and how to stay away from these cyberthreats.

According to the latest report, from 2017, the average annual cost of cybercrime for these 254 companies was $11.7 million, with an average of 50 days to contain an attack. In addition, it is estimated that the number of attacks increases on average each year by 27.4% and, as a result, the cost of security has increased by 22.7% in the last year. This report also reveals that financial services have the highest security cost, despite the fact that the health care service receives the most attacks.

Due to the increase in cyber attacks and the high cost of patching security breaches, both the public and private sectors are incorporating security throughout all stages of the SDLC [16]. Therefore, it is becoming imperative to take security into account during the design and development of software systems, as well as to expand verification and validation capabilities to cover all information security concerns [18].

Incorporating security during the SLC requires the use of a variety of vulnerability prevention and discovery techniques [19]. One of these vulnerability discovery techniques is static source code analysis. This technique provides a scalable form for code review, can be applied at the beginning of the life cycle, does not require the system to be finished, and can be applied to parts of the code individually [20].

Current framework

Source code analysis tools, also known as static application security testing (SAST) tools, are designed to analyze the source code, or a compiled version of that code, and help find security flaws.

SAST has evolved rapidly over the last decade, from simple lexical analysis to much more complex techniques. In general, however, static analysis problems depend on the complexity of the security errors and the knowledge of the code author.

Therefore, SASTs do not detect all vulnerabilities in the source code (also known as false negatives) and are prone to report bugs that upon closer examination turn out not to be security vulnerabilities (also known as false positives). To be of practical use, a static code analysis tool must find as many vulnerabilities as possible and with as few false positives as possible.

As with any other technology, this tool has advantages and also has drawbacks and more when applied in project development.

The advantages of SAST are:

- Good scalability. They are designed to be used in continuous integration environments where the code and the project itself is frequently modified.
- Useful for known problems. They are very useful, especially when it comes to finding frequent bugs that leave no room for doubt, such as buffer overflow, SQL injection, etc.
- Good feedback for developers. The information provided to the developer, in relation to the bug, is very accurate. This information ranges from the exact

location of the bug, through the severity of the bug and even possible solutions.

The disadvantages of SAST are:

- High rate of false positives. The complexity of the code and the facility to detect the failure are determining factors to obtain a reduced rate of false positives.
- Difficulty in verifying whether a failure is a vulnerability. It is difficult to prove that the bug found is a vulnerability if it is not frequent.
- Difficulty in finding configuration faults. If the configuration is not present in the code, the tool cannot detect if there are problems in it.

PROBLEM ANALYSIS

Description

In the previous chapter we have described the main problem (cyber attacks) that technology currently has and that any software systems developer has to deal with. It has also described one of the techniques that these developers have (SAST), its advantages and disadvantages.

After an exhaustive analysis of the possible solutions that can be carried out, it has been decided to take a course, imitating as much as possible the structure of a MOOC, in which it is taught how to use one of these tools for static analysis of source code. It should be noted, however, that this course does not include all the components (videos, readings, questionnaires) available in an MOOC, nor is it intended for a mass audience.

Considering these factors, it has been decided that focusing the course on the Python language would be a greater contribution than doing it for any other language.

The tool that will be used in this course is called SonarQube and is developed by the company SonarSource. This tool, unlike those mentioned in the previous chapter, has an open source version (Community Edition) and also has three versions under commercial license. The following section will describe in more detail the features of this tool.

SOLUTION DESIGN

Learning outcomes

The objective of this course is for the student to acquire a series of knowledge and skills, which are described below:

- The ability to read, analyze, and understand the instructions in a course.
- The ability to locate, identify, and select key concepts in a lesson.
- The ability to examine the documentation of a software system.
- The ability to complete an assessment test based on the knowledge acquired.
- A comprehensive knowledge of the functionalities offered by a static source code analysis tool.
- The ability to inspect and test a static source code analysis tool.
- The ability to run an analysis on a software project.
- The ability to evaluate and classify the result of a source code analysis.
- The ability to apply a solution to problems encountered during analysis.
- The ability to prepare, design and elaborate a custom analysis rule.

Structure of the solution

This first section describes the structure of the solution designed for the problem mentioned in the previous chapters.

This course is structured in five modules, along which each one of the SonarQube functionalities will be known. The aim of the first four modules is to prepare the working environment and show the interface of each tool to be used, while the fifth aims to put into practice what has been learnt in the previous modules, in a real case.

The content of each module is described below:

Module 1. Prerequisites: Information and instructions are provided to carry out previous configurations and installations.

Module 2. SonarQube: Information and instructions are provided to carry out the installation and configuration of the SonarQube tool.

Module 3. SonarQube Scanner: Provides information and instructions for installing and configuring the SonarQube Scanner tool.

Module 4. User interface: Information is provided on the different views that make up the web interface of the SonarQube tool and the possible functions of each one.

Practical Case: Detailed instructions are provided to test the main functionalities of SonarQube on a real project. These functionalities include analysing a project, evaluating the results, solving problems encountered and creating customised analysis rules to detect one of the main vulnerabilities that currently exist.

Each of these modules is divided into one or more sections, which provide the information and instructions necessary to understand the concepts that are intended to convey in the module.

At the end of the first four modules there is one or more Self-Assessment questions, which are recommended to consolidate the acquired knowledge and whose solution can be found in Annex B.

Software and hardware Requirements

This section details the technical requirements that the system where both the SonarQube and the SonarQube Scanner will be installed and run must meet.

To run and use the SonarQube server on a small scale (small group) you need at least 2GB of RAM for the server instance and 1GB of RAM for the OS.

The amount of space needed on the hard disk depends exclusively on how much code you want to analyze and the frequency of the scans, although to get the most out of the SonarQube server, it must be installed on a hard disk with a high read and write speed.

ASSESSMENT

Evaluation method

Once the writing and revision of the course was completed, and with the aim of evaluating the solution designed, a survey was drawn up to find out the opinion of some possible future students of this course.

A total of three users participated in this survey, all of them with a profile of computer engineer. In addition, all have had contact and know the Python language, an essential requirement since the projects analysed in the course are written in this language.

The three users have received a document, in pdf format, containing chapter four and the annex with the answers to the self-evaluation. They have also been provided with a document in txt format.

Evaluation results

Once all the users' answers have been received, the answers to the four questions in the questionnaire have been analysed. The analysis consists of marking the positive comments with a green colour and the negative comments of the user with a red colour.

Once the positive and negative comments have been marked, they can be put together into a single list to see if there are any common points.

If you look at each of the above lists you can draw the following general conclusions about the evaluation:

- The course in general is easy to follow, with detailed instructions and in order.
- The SonarQube tool is a good support to generate quality code.
- Both Linux and Windows instructions need to be improved.
- Provide the student with more material to learn the XPath language.

OUTCOME

Objectives met

In this epigraph an analysis is made in order to know which objectives, of those initially proposed, have been fulfilled once the development and evaluation part of the work has been completed.

In the first chapter of this document, several objectives were set with respect to what this work was intended to achieve, although they can be summed up in two main objectives. Each is then analysed:

"*...to create a document in which the necessary instructions are provided to know in detail the characteristics of a static code analysis tool, its basic functioning and its application to a real project*".

As far as this objective is concerned, it can be considered to have been achieved, since this is endorsed by the conclusions drawn from the analysis that has been carried out on the evaluation responses received. Two of the general conclusions drawn in section 5.2 confirm that it has been possible to create a document with ordered and well detailed instructions that help the user to know how a static code analysis tool works. Likewise, the general conclusions also indicate that users who follow the material are able to see the benefits of using the tool in any environment.

"*...the necessary documentation will be provided to make advanced use of the functionalities of this type of tool*."

As far as this objective is concerned, the general conclusions drawn in section 5.2 make it impossible to consider this objective fulfilled. The reason is that all users who have evaluated the course agree that additional material is needed to learn how to create custom rules. This statement therefore makes it impossible to consider the objective as accomplished since rule creation is an advanced functionality that is intended to be taught in the course.

Future lines of work

After the evaluation and review of objectives, some conclusions have been reached about possible improvements to the course and even new courses. The following are some ideas about the work that could be done in the future:

- Fix errors made in the course instructions, such as differentiating between different Linux distributions or explaining how to install Pylint on Windows.
- Expand the course with instructions for integrating SonarQube in integrated development environments, such as Eclipse, or in continuous integration software, such as Jenkins.
- Expand the custom rule creation section with additional external or proprietary information, or even create an additional course to learn the XPath language in depth.

- Extend the course with additional case studies so that students can consolidate and acquire new knowledge.
- Create a plugin that includes rules to detect particular Python vulnerabilities.

Socio-economic impact

In order to analyse the socio-economic impact that the use of this work will have, it is necessary to take into account all the studies mentioned in the chapter State of the question and the results of these studies.

As already mentioned, in recent years a considerable number of organizations have been victims of cyber attacks, such as Yahoo, Ebay, Anthem and Sony PlatStation Network.

It has also been indicated that according to the Identity Theft Resource Center [11], in the last four years there have been more than 9395 cyber attacks, losing nearly 1 billion records with confidential information.

On the other hand, according to the latest report from the Ponemon Institute [13], the number of attacks increases every year by 27.4%, which has caused the cost of security to increase by 22.7% in the last year to stand at about 11.7 million dollars.

Considering these data, it can be said that this work has a direct impact and has a positive influence on both society and the economy.

On the one hand, through the course designed, the user will have the knowledge and skills necessary to identify bugs and security vulnerabilities in a software system and correct them, thus reducing the impact of a possible cyber attack or even avoiding it.

In this way, if the user works currently, or in the future, as a software systems programmer, he will be able to apply this knowledge and skills in his working environment, avoiding that his organization has to invest money in software security measures or in solving security gaps already exploited by the attackers.

On the other hand, if the user works in his work environment with confidential data of patients or consumers, applying this knowledge and skills may prevent attackers from compromising the confidentiality of such data and therefore the patients or consumers themselves.