PAI3. Seguridad del Software en el Desarrollo de Sistemas

Escuela Técnica Superior de
Ingeniería Informática

Índice

Alcance y Objetivos del Proyecto

El principal objetivo de este proyecto reside en detectar las vulnerabilidades sucedidas a lo largo del desarrollo de un proyecto. Sin embargo esto desentraña un problema más complejo, ya que las buenas prácticas implicarían la lectura línea a línea del código del proyecto desarrollado, pero esto resulta algo inabarcable debido a la cantidad de componentes distintos que se encuentran en nuestros proyectos, además, también supone la interacción del proyecto con otros sistemas del equipo, por lo que se requiere el uso de herramientas más sofisticadas.

En la actualidad existen multitud de herramientas que sirven para analizar código fuente para diversos lenguajes, basandose en los estándares de seguridad de mayor prestigio en la industria como OWASP, CVSS, CWE y WASC.

Además, también se pide dar una solución a las vulnerabilidades que consideremos críticas.

Recursos para el Proyecto

Recursos Humanos

El equipo de seguridad para esta asignatura se compone de los siguientes miembros:

- Barragán Candel, Marina - estudiante de Ing. Informática – Tecnología Informática, en la mención de Tecnologías de la información
- Calcedo Vázquez, Ignacio - estudiante de Ing. Informática – Tecnología Informática, en la mención de Sistemas de Información
- Polo Domínguez, Jorge - estudiante de Ing. Informática – Ingeniería de Computadores
- Sala Mascort, Jaime Emilio - estudiante de Ing. Informática – Tecnología Informática, en la mención de Computación

El equipo se organizará mediante la herramienta de Github, bajo el repositorio público SSII.

Recursos Tecnológicos

Debido a cuestiones monetarias y temporales, el equipo ha decidido realizar este proyecto basándose en dos herramientas, Visual Code Grepper y SonarQubee, que se explorarán más adelante en el informe.

Estudio Inicial

En primera instancia para entender el análisis de los proyectos, debemos diferenciar dos tipos de análisis, el análisis estático y el dinámico. El análisis estático consiste en a partir del código fuente de un sistema tratar de averiguar las posibles vulnerabilidades que pueda contener y el dinámico consiste en que a partir de la aplicación en una situación de testeo, sin acceso al código fuente, se traten de explotar las posibles vulnerabilidades que pueda entrañar.

Ambos formas combinadas ofrecen un análisis muy completo, ya que las estáticas son ideales para encontrar vulnerabilidades muy conocidas como criptografía débil, inyección SQL y buffer underflow y overflow, y las dinámicas prueban como sería desde un usuario externo haciendo uso de las interfaces.

Sin embargo, para este proyecto, se nos ha pedido el uso exclusivo de herramientas estáticas.

Actividades del Proyecto

Con el objetivo de alcanzar los objetivos arriba expuestos, se han desarrollado las siguientes actividades.

Análisis de Herramientas del Mercado

Debido a problemas organizativos por falta de tiempo y recursos del equipo, se ha estimado disminuir la cantidad de herramientas analizadas a dos, con el fin de poder dar una información más detallada sobre ellas en contra de ofrecer más herramientas con un análisis menos intensivo.

Visual Code Grepper

VisualCodeGreeper, una herramienta open source, y por tanto de acceso libre. Se basa en el análisis del código fuente, tanto de manera contextual como de posibles problemas potenciales o fallos de seguridad, incluso comentarios que indiquen código en mal estado. Además proporciona un gráfico circular para mayor interacción por parte del usuario. Su origen, según proclama su creador, parte de producir una herramienta que no devuelva un gran número de falsos positivos, así como identificar desbordamientos de buffer y otras vulnerabilidades.

Esta herramienta soporta lenguajes tales como C/C++, Java, PL/SQL, C#, VB, PHP y COBOL, por lo que podemos cubrir las peticiones del cliente en lo que se refiere a lenguaje para analizar.

Como objetivo de valorar la futura eficacia de la aplicación, hemos considerado respaldarnos en una serie de benchmarks que los desarrolladores de OWASP realizaron a esta herramienta, usando un paquete de 2740 test de pruebas, para medir su eficacia. De este se han obtenido la siguiente puntuación, 14,78 sobre 100, donde ha habido un ratio de True Positive de 53,51% y un ratio de False Positive de 38,73%, en el Anexo II: Benchmarks de OWASP se pueden encontrar los resultados detallados.

Como se puede observar, VisualCodeGrepper detectó más del 50% de las vulnerabilidades que se encontraban en los test, pero también falló un casi un 40% en alertar de positivos que no lo eran, por lo que no posee credibilidad muy elevada. Es también reseñable que los Falsos Negativos presentes en los resultados pueden ocurrir debido a la no configuración de la herramienta con la regla correspondiente a la vulnerabilidades que se hallan en cada test.

El coste temporal que presenta la aplicación, lo conformará el tiempo necesario para ejecutar el paquete de pruebas anterior. Ejecutando los test en una máquina cuyas características son un i5 8250U y 12GB de RAM DDR4, se ha obtenido un tiempo de ejecución necesario de 6 minutos y 23 segundos. Se trata de un tiempo adecuado dada la cantidad de test analizados y su proporción de vulnerabilidades.

En cuanto a la usabilidad que presenta la herramienta, desde el equipo podemos asegurar que es su punto fuerte, debido a simplicidad notable a simple vista. La instalación es fácil y de cortos pasos, su interfaz en forma de tabla es sencilla, y para ejecutarlo solo hace falta elegir lenguaje, fichero con las fuentes, y lanzar el escáner.

Además, su repositorio y código recibió actualizaciones hace un año, por lo que no es un herramienta tan olvidada por sus desarrolladores como otras presentes, para el análisis de código.

SonarQube

SonarQube, anteriormente conocido como Sonar, es una herramienta de análisis de código fuente libre. Su funcionamiento se basa en el uso de múltiples métricas de análisis de código estático para determinar vulnerabilidades del código y mejorar la calidad del mismo. Adicionalmente, resulta un software muy práctico ya que facilita mecanismos de Integración Continua (CI/CD).

Estudiando las alternativas que ofrece el producto, existe una versión open source totalmente gratuita que incluye una serie de lenguajes para analizar, bien sea por defecto o gracias a extensiones gratuitas del mismo software: Java, JavaScript, C#, TypeScript, Kotlin, Ruby, Go, Scala, Flex, Python, PHP, HTML, CSS, XML y VB.NET. Esta opción nos permite cubrir una parte de los requerimientos del

cliente, el código en Java. Sin embargo, para poder acceder al análisis de código estático de C, se requiere adquirir la versión "Developer" de SonarQube. De cara al estudio de alternativas para el cliente es necesario remarcarlo, ya que en última instancia resulta en una inversión monetaria que debe ser estudiada por el cliente.

Con el fin de valorar la futura eficacia de la aplicación, hemos considerado otra vez respaldarnos en una serie de benchmarks de OWASP que se realizaron con esta herramienta, usando un paquete de 2740 test de pruebas, para medir su eficacia. De este se han obtenido la siguiente puntuación, 33,34 sobre 100, donde ha habido un ratio de True Positive de 50,36% y un ratio de False Positive de 17,02%, en el Anexo II: Benchmarks de OWASP se pueden encontrar los resultados detallados.

A nivel de eficiencia y rendimiento, las pruebas de este benchmark indican que, de media, con una máquina con un Intel Core i5 3570 con un reloj de 3.40 GHz y una memoria RAM total de 16 Gigabytes se debería de analizar todo el benchmark en aproximadamente 3 minutos, lo cual es un tiempo considerablemente pequeño para un proyecto que contiene 2740 casos de prueba con todo tipo de vulnerabilidades.

Tratando la usabilidad, SonarQube presenta un formato muy cómodo a nivel de interfaz. Solo en la versión community ya se puede obtener una idea aproximada de las funcionalidades del producto. SonarQube se puede configurar para uso local como para uso en servidores, lo cual permite ajustarse al cliente en función de sus necesidades, teniendo además varias guías de instalación y una comunidad de soporte consistente.

La interfaz guía al usuario a lo largo de la configuración de proyectos para que este no tenga problemas a la hora de realizar escáneres de seguridad, y en caso de que los tuviera, siempre facilita el acceso a su guía de uso, la cual se organiza en función del lenguaje y de las tecnologías asociadas a los mismos.

Acerca del informe de resultados de la herramienta, se tiene evidencia gráfica de que los informes realizados por SonarQube son detallados y extensos, apuntando a muchos detalles tanto a nivel de seguridad como a nivel de desarrollo de código. El informe de resultados de la herramienta incluye un desglose de los distintos problemas que presenta el código a nivel estructural y las distintas vulnerabilidades del código a nivel de seguridad, pero adicionalmente la herramienta permite hacer una revisión del código a nivel gráfico, en el que se muestran los distintos fragmentos del código y su estado, y distintos gráficos sobre la evolución y evaluación del proyecto en los distintos ámbitos del mismo. Este nivel de detalle ya se puede conseguir desde la versión community del software, por lo que se puede deducir que la profundidad técnica de la herramienta es notable y muy provechosa para un buen analista. Finalmente, SonarQube facilita distintas gráficas mostrando el desarrollo del proyecto, los errores corregidos y otra información relevante sobre la proyección del mismo.

Conclusión del Análisis

Como conclusión del análisis de las herramientas, SonarQube resulta una herramienta relativamente intuitiva de usar, con un diseño excelente, un rendimiento aceptable y un despliegue de información completo y desglosado. Su integración de CI-CD y software de control de versiones y gestión de proyectos lo hacen una herramienta más que adecuada para el análisis de código para muchas empresas y una solución más que aceptable para muchos casos de negocio. Lamentablemente, para poder desbloquear todas sus funcionalidades es necesario acceder a las versiones developer y enterprise del software, incurriendo necesariamente en un desembolso económico. Sin embargo, el equipo de análisis considera que esto puede considerarse una inversión positiva y productiva en el largo plazo, pues el ahorro de costes por solución de problemas técnicos y de seguridad es más que notable y en muchos casos puede suponer una productividad en el entorno de desarrollo muy elevada, lo que a la larga producirá más beneficios y compensará por completo el coste de la licencia del producto, sin embargo, se usará VisualCodeGrepper debido a que es una herramienta gratuita, accesible, muy fácil de usar y con muchos lenguajes soportados. Aunque esté un paso por detrás en cuanto a fiabilidad, en comparación con SonarQube.

Análisis de Vulnerabilidades

En relación a la herramienta anteriormente escogida, se ha aplicado a los siguientes proyectos.

Proyecto de Prueba en Java

Se han extraído las seis vulnerabilidades que consideramos críticas.

Potential SQL Injection

Una vulnerabilidad de tipo SQL injection es una puerta trasera para todo aquel atacante que quiera obtener información de una base de datos, además de permitirle añadir, modificar y borrar contenido a voluntad. Encontrando estos fallos de seguridad en las aplicaciones y páginas web, un atacante es capaz de evitar las medidas de seguridad que controlan el acceso a dicha base de datos.

Encontramos esta vulnerabilidad en tres ocasiones en el proyecto.

En el fichero CsrfAccessDeniedHandler.java, exactamente en la línea 27:.

```
logger.info("[-] URL: " + request.getRequestURL() + "?" + request.getQueryString() + "\t" + …
```

En el fichero ficherto SQLI.java, línea 107 del código.

```
PreparedStatement st = con.prepareStatement(sql);
```

Aparece, de nuevo, en el fichero SQLI.java, en la línea 66.

```
        ResultSet rs = statement.executeQuery(sql);
```

Poor Input Validation

Estas tres vulnerabilidades son calificadas con riesgo alto, debido a que la aplicación hace una petición request de datos, sin que estos sean validados. La herramienta no ha localizado complementos de validación en los archivos XML de la aplicación que estamos analizando.

Cuya prioridad vendría tras las tres anteriores, pero que siguen siendo importantes.

Situada en el fichero OriginFilter.java, línea 43.

```
        "\tCurrent url:" + request.getRequestURL());
```

La encontramos en LoginFailureHandler.java, en la línea 24.

```
        logger.info("Login failed. " + request.getRequestURL() +
```

El fichero que la contiene es CRLFInjection.java, línea 25.

```
        String author = request.getParameter("test3");
```

Proyecto de Prueba en C

Se han extraído las seis vulnerabilidades que consideramos críticas.

Proyecto de la Consultoría 2

Soluciones Ofrecidas

Las soluciones que se ofrecen son las siguiente para los distintos proyectos

Proyecto en Java

Para tratar SQLInjection, se recomienda realizar previamente a la comunicación con la base de datos, la creación de declaraciones parametrizadas. Estas garantizan que los parámetros de entrada que se pasan a las declaraciones SQL sean seguras. Por ejemplo, evitaríamos crear la declaración para la base de datos mediante concatenaciones con parámetros, y en su lugar, especificaríamos el parámetro que deseamos al crear la declaración, y lo usaríamos al ejecutarla.

Y en cuanto a la mala validación de datos, simplemente recomendamos validar y sanear el uso de datos obtenidos mediante los request, puesto que pueden poner en riesgo la aplicación. La validación consiste en comprobar si los valores que recibimos son adecuados y dentro de los márgenes que son

coherentes con lo que necesitamos. EL saneamiento trata de analizar el código y eliminar todas aquellas llamadas input por parte del usuario, que sean peligrosas.

Proyecto de Prueba en C

Proyecto de la Consultoría 2

Informe Final

Una vez analizados los proyectos anteriores y ofrecido soluciones para tratar de evitar estas vulnerabilidades, el equipo se ve en la obligación de recomendar el uso de nuevas formas de análisis de código que no sean de tipo estático, ya que su fiabilidad no es la mejor y puede resultar un proceso arduo discernir entre falsos positivos y verdaderos positivos. Por ello recomendamos el uso de análisis dinámicos, donde a pesar de obtener un menor número de verdaderos positivos estos suelen relacionarse más con las amenazas reales, ya que son pruebas en un entorno más real ya que se actúa sobre una caja negra. Una combinación de ambas es una estrategia fuerte para proteger nuestras aplicaciones.

Finalmente, comentar que hoy en día, está habiendo un auge de un nuevo tipo de análisis, análisis interactivo, IAST, con un enfoque distinto al paradigma de la seguridad, donde se plantea un agente dentro del servidor de aplicaciones que ofrece detección inmediata de los problemas de seguridad gracias al análisis de flujo de ejecución de las aplicaciones. Además, no requiere modificar las aplicaciones, o ejecutar ningún tipo de test de penetración manual.

El análisis se ejecuta observando el comportamiento interno de la aplicación, que es un punto de vista ideal para este tipo de problemas. En particular, la implementación de un agente IAST introduce sensores en partes críticas de la aplicación.

Este tipo de análisis produce puntuaciones muy prometedoras en los benchmarks de OWASP donde se alcanza el 100 sobre 100, detectado el ratio del 100% en True Positive y un 0% de False Positive. Convirtiéndolo en una de las mejores vías para el análisis de los proyectos.

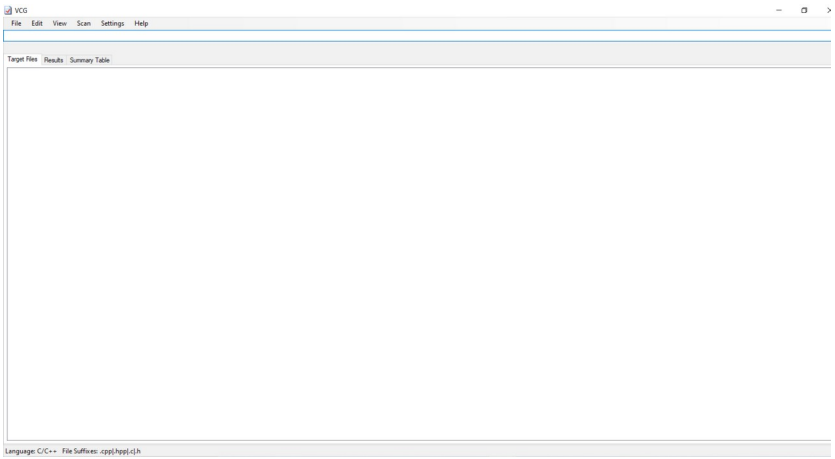
Anexo I: Manual de Uso de VisualCodeGrepper

Puesto que hemos utilizado esta VisualCodegrepper como herramienta de análisis, desde el quipo hemos visto conveniente la creación de este manual de instalación y uso de dicha herramienta.

Para comenzar, descargamos el archivo comprimido de la herramienta desde el siguiente enlace y ejecutamos el archivo VCG-setup.msi.

En la ventana de bienvenida, pulsamos Next y, a continuación, seleccionamos la carpeta donde queremos instalar la herramienta, y para que usuarios. Tras esto, nos confirma que el programa está listo para instalarse, y volvemos a pulsar en Next. Tras la instalación completa, pulsamos Close, y nos fijamos en nuestro escritorio, puesto que debe haberse creado un acceso directo para la aplicación. Lo ejecutamos, aceptamos el recordatorio de seleccionar un lenguaje antes de analizar, y ya tenemos disponible la herramienta.

Nos aparecerá la siguiente ventana de aplicación, donde a continuación explicaremos cada vista.



La pestaña File nos permite seleccionar el directorio que contiene los archivos que queremos analizar, elegir y analizar un único archivo, guardar los resultados en un fichero de texto, importar y exportar en formato XML, importar y exportar en formato CSV, y exportar metadatos de código en XML.

La pestaña Edit proporciona edición simple, tal como cortar, copiar, pegar, y buscar.

La pestaña View permite agrupar los resultados con texto enriquecido por su tipo de problema o por fichero al que afecte, además de poder seleccionar si queremos ver texto enriquecido que ofrezca más información de los resultados.

La pestaña Scan es donde podremos seleccionar el tipo de escaneo que queramos realizar, desde un escaneo completo, sólo de código ignorando comentarios o únicamente comentarios, además de otras opciones. También podemos ordenar los resultados por severidad de la vulnerabilidad, o por el nombre del fichero que se analiza. Por último, aparece una opción para filtrar los resultados por importancia de vulnerabilidad.

En la pestaña Settings elegiremos el tipo de lenguaje que queremos analizar, ya sea C/C++, java, o cualquier otro soportado por la herramienta.

La última pestaña, Help, muestra información sobre su creador, y versión de la herramienta que se está ejecutando.

Para su correcto uso, lo primero que haremos será seleccionar el tipo de lenguaje que vamos a analizar. A continuación, elegimos un directorio o fichero a analizar y activamos el escaneo que necesitemos, en este caso Full Scan. En la ventana que aparece, Visual Breakdown, podemos seleccionar que nos aparezca un gráfico informativo del análisis, tras cada escaneo.

Aparecerán dos ventanas, la primera con las vulnerabilidades encontradas en el código, cuya severidad queda marcada por un rango de colores, y un gráfico general del escaneo.

En la ventana con las vulnerabilidades, podremos observar el tipo de vulnerabilidad que presenta cada parte del código, el motivo, su localización y la línea afectada.

Para finalizar, el rango de colores que aplica la herramienta para los problemas encontrados, es el siguiente:

Magenta -> Vulnerabilidad crítica

Red -> Riesgo alto

Naranja -> Riesgo medio

Amarillo -> Riesgo estándar o normal

Azul claro -> Riesgo bajo

Verde -> Problema en potencia

Azul oscuro -> Comentario sospechoso que indica fallos en el código.

Esta información se puede encontrar en el propio enlace de descarga, y está proporcionada por el creador.

Anexo II: Benchmark de OWASP

Los benchmarks que se han obtenido a partir de OWASP han sido los siguientes:

VisualCodeGrepper

Statistics

Tool elapsed analysis time	Time not specified
Tool overall score (0-100)	14,78%
Total test cases	2740
Download raw results	Actual Results

Detailed Results

Category	CWE #	TP	FN	TN	FP	Total	TPR	FPR	Score
Command Injection	78	56	70	68	57	251	44,44%	45,60%	-1,16%
Cross-Site Scripting	79	0	246	209	0	455	0,00%	0,00%	0,00%
Insecure Cookie	614	36	0	0	31	67	100,00%	100,00%	0,00%
LDAP Injection	90	0	27	32	0	59	0,00%	0,00%	0,00%
Path Traversal	22	128	5	16	119	268	96,24%	88,15%	8,09%
SQL Injection	89	142	130	105	127	504	52,21%	54,74%	-2,54%
Trust Boundary Violation	501	27	56	35	8	126	32,53%	18,60%	13,93%
Weak Encryption Algorithm	327	97	33	116	0	246	74,62%	0,00%	74,62%
Weak Hash Algorithm	328	0	129	107	0	236	0,00%	0,00%	0,00%
Weak Random Number	330	193	25	223	52	493	88,53%	18,91%	69,62%
XPath Injection	643	15	0	0	20	35	100,00%	100,00%	0,00%
Totals*		**694**	**721**	**911**	**414**	**2740**			
Overall Results*							**53,51%**	**38,73%**	**14,78%**

SonarQube

Statistics

Tool elapsed analysis time	0:05:30
Tool overall score (0-100)	33,34%
Total test cases	2740
Download raw results	Actual Results

Detailed Results

Category	CWE #	TP	FN	TN	FP	Total	TPR	FPR	Score
Command Injection	78	107	19	16	109	251	84,92%	87,20%	-2,28%
Cross-Site Scripting	79	0	246	209	0	455	0,00%	0,00%	0,00%
Insecure Cookie	614	36	0	31	0	67	100,00%	0,00%	100,00%
LDAP Injection	90	27	0	0	32	59	100,00%	100,00%	0,00%
Path Traversal	22	0	133	135	0	268	0,00%	0,00%	0,00%
SQL Injection	89	0	272	232	0	504	0,00%	0,00%	0,00%
Trust Boundary Violation	501	0	83	43	0	126	0,00%	0,00%	0,00%
Weak Encryption Algorithm	327	130	0	116	0	246	100,00%	0,00%	100,00%
Weak Hash Algorithm	328	89	40	107	0	236	68,99%	0,00%	68,99%
Weak Random Number	330	218	0	275	0	493	100,00%	0,00%	100,00%
XPath Injection	643	0	15	20	0	35	0,00%	0,00%	0,00%
Totals*		**607**	**808**	**1184**	**141**	**2740**			
Overall Results*							**50,36%**	**17,02%**	**33,34%**

Leyenda

Common Weakness Enumeration (CWE), The primary MITRE CWE number for this vulnerability category.

True Positive (TP), Tests with real vulnerabilities that were correctly reported as vulnerable by the tool.

False Negative (FN), Tests with real vulnerabilities that were not correctly reported as vulnerable by the tool.

True Negative (TN), Tests with fake vulnerabilities that were correctly not reported as vulnerable by the tool.

False Positive (FP), Tests with fake vulnerabilities that were incorrectly reported as vulnerable by the tool.

True Positive Rate (TPR) = TP / (TP + FN), The rate at which the tool correctly reports real vulnerabilities. Also referred to as Recall, as defined at Wikipedia.

False Positive Rate (FPR) = FP / (FP + TN), The rate at which the tool incorrectly reports fake vulnerabilities as real.

Score = TPR – FPR, Normalized distance from the random guess line.