El análisis de código, fuente de seguridad

La complejidad creciente de los entornos de aplicaciones web hace cada día más arduo su desarrollo. Esto, unido al incremento de la interacción con los usuarios, provoca que puedan cometerse errores en el desarrollo que se conviertan en problemas de seguridad aprovechables por un atacante. Una de las mejores formas de detectar las debilidades de una aplicación es estudiándola desde dentro, esto es, analizando su código fuente. Para que esto sea posible, es recomendable una fase de análisis de código en el ciclo de vida del desarrollo de las aplicaciones apoyado con herramientas automáticas. Este artículo profundiza en los aspectos a tener en cuenta para introducir esta fase de análisis con éxito.

Joaquín Crespo Pérez

Por todos es conocido que en el mundo actual, la mayoría de los problemas de seguridad que afectan a las aplicaciones web publicadas en Internet se deben fundamentalmente a errores de desarrollo. Si se revisan las listas de vulnerabilidades más frecuentes elaboradas por las principales organizaciones (SANS [1], OWASP [2], WEBAPPSEC [3]), la mayoría de ellas corroboran esta afirmación. A consecuencia de esto durante los últimos años, en las pruebas de intrusión, la mayor parte del esfuerzo se ha dedicado a localizar posibles errores resultado de un desarrollo incorrecto de las aplicaciones, y no tanto a los problemas específicos de configuración de los servidores o vulnerabilidades conocidas. Aunque este último aspecto no se debe despreciar, ya que sigue siendo un posible foco de vulnerabilidades e incluso puede facilitar el aprovechamiento de otras.

Sin embargo, intentar descubrir las vulnerabilidades de una aplicación a través de pruebas de intrusión tiene ciertas desventajas. Por un lado, la cobertura de una prueba de intrusión nunca será total por no disponer de permisos al acceso de determinadas funcionalidades, y por la complejidad de las interacciones entre la aplicación y el interfaz web. Por otro lado, arreglar una aplicación ya en producción (o a punto de ponerse en producción) supone un coste muy elevado e, incluso, ciertas vulnerabilidades que surgen por un diseño incorrecto obligan prácticamente a rehacer la aplicación. Como consecuencia de esto, tras las pruebas de intrusión quedarán vulnerabilidades sin detectar y algunas otras vulnerabilidades deberán asumirse, como una fuente de riesgo a controlar, puesto que su corrección no es posible en ese estadio. Uno puede imaginarse (aunque hay pocos estudios cuantitativos que lo demuestren **[4]**) que será más fácil arreglar vulnerabilidades detectadas cuando la aplicación aún está en desarrollo (y, por tanto, es más fácil de modificar) que cuando ya está en un estado de paso a producción.

Teniendo en cuenta estos factores, muchas organizaciones empiezan a considerar que sería mejor tomar las medidas necesarias para detectar estos problemas, que serán la principal fuente de riesgo en un futuro, antes de dar la aplicación por finalizada, y someterla a pruebas de intrusión en los entornos de producción, o en los de pre-producción, tal y como se hace generalmente.

Una de las tareas fundamentales que se pueden realizar en nueva fase "de validación de seguridad", previa a la puesta en producción (o en pre-producción), es el **análisis de código** de las propias aplicaciones.

Generalmente las aplicaciones web actuales recaen sobre arquitecturas con varios niveles en las que cada elemento es independiente. Se tienen servidores que ofrecen el contenido estático, servidores que se encargan de ejecutar toda la lógica de presentación dinámica y servir como interfaz a la lógica de acceso a datos, y por último una capa de acceso a datos mantenidos en otros servidores distintos. Se trata de la estructura habitual de servidor web, servidor de aplicaciones y servidor de base de datos. Adicionalmente, hoy en día aparecen los tan promulgados *servicios web*, en los que prima el intercambio de mensajes en ficheros estructurados.

Todo este compendio de tecnologías implica un gran número de implementaciones distintas de código, generalmente con diversos lenguajes: código HTML estático, código dinámico (implementado en Java, .Net u otros lenguajes), sentencias SQL, ficheros XML, etc. Todos y cada uno de estos componentes de una aplicación son importantes en el análisis de código, porque todos son susceptibles de ser una fuente de los problemas de seguridad que la aplicación pueda tener en los entornos definitivos de producción.

Es muy difícil disponer de personal de desarrollo, responsable de cada una de las partes de una aplicación, con conocimientos suficientes en seguridad específicos sobre aplicaciones web y que conozca los problemas comunes (para evitarlos, claro está). Por otro lado, el coste[1] que supondría realizar un análisis de todo el código de una aplicación de forma manual, hace inviable introducir esta tarea en el ciclo de desarrollo. Por eso, es mucho más conveniente utilizar herramientas automatizadas de análisis de código, que minimicen la cantidad de recursos necesarios para su ejecución, reduzcan el tiempo de ejecución de esta tarea y optimicen el resultado del análisis con las dos visiones que se pretenden: expertos en desarrollo y expertos en seguridad.

Dado que se habla de la fase de análisis de código como parte del propio desarrollo de aplicaciones, la utilización de las herramientas y la aplicación de los procedimientos en esta fase deben ser responsabilidad del personal dedicado al desarrollo, sin que esto sea óbice para que

El uso de analizadores automáticos de código no son la panacea para garantizar el desarrollo de aplicaciones 100% seguras. Del mismo modo que las herramientas automáticas de búsqueda de vulnerabilidades no cubren todos los aspectos de una auditoría técnica, las herramientas de análisis de código no tienen el conocimiento suficiente para descubrir algunos de los problemas de seguridad más complejos.

En principio, se incluye en este análisis todo lo relacionado con la localización de problemas de seguridad en el código fuente de las aplicaciones. Ni que decir tiene, que además de las comprobaciones de seguridad, los objetivos del análisis pueden hacerse extensibles a otros aspectos de optimización del código de las aplicaciones. Por ejemplo, posibles mejoras de rendimiento, reutilización de código, etc.

Análisis de código: premisas

Aunque actualmente existen iniciativas que sientan las bases para conseguir un desarrollo seguro de aplicaciones web (ahí existen ejemplos tan útiles como el cada día más maduro proyecto *OWASP*), el análisis de código va un paso más allá, intentando, no sólo dar recomendaciones que ayuden a un desarrollo seguro, sino detectar además aquellas partes del código en las que estas recomendaciones se incumplen y ofrecer

[1] El coste no sólo es función de que el esfuerzo sea realizado por la empresa o por personal externo, sino también habría que considerar los retrasos en la puesta en producción derivados de la introducción de una actividad que lleva un tiempo considerable.

el personal dedicado a la seguridad dentro de la compañía participe en esta fase.

Así pues, en las herramientas de desarrollo de aplicaciones deben considerarse los siguientes mecanismos flujos de interacción:

• El primero, integrado en el entorno de desarrollo, se encarga de detectar errores en el código y ofrecer recomendaciones concretas a los desarrolladores para su resolución.

• El segundo produce informes generales sobre las debilidades detectadas en el conjunto del código de las aplicaciones y que podrán utilizar los responsables seguridad.

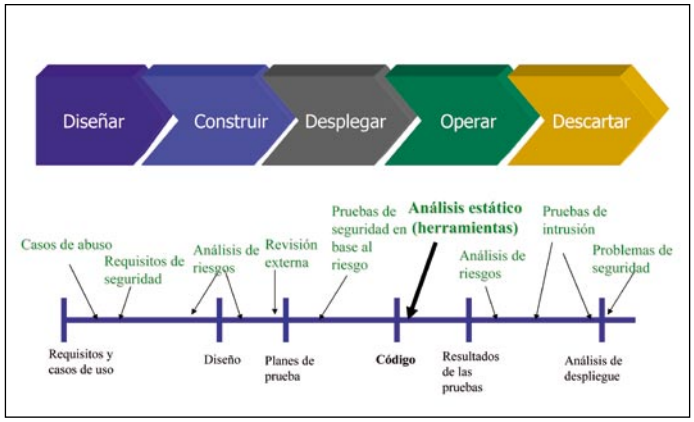

Figura 1

• A caballo entre las funciones asignadas a desarrolladores y responsables de seguridad, las herramientas de análisis deben permitir, además, imponer las restricciones que la compañía ha establecido para las aplicaciones que desarrolla, ajustando niveles de criticidad y añadiendo puntos de control personalizados. Se dice que esta relación afecta a ambos grupos porque el grupo de seguridad será el que imponga los controles a realizar así como su nivel de riesgo, y el de desarrollo será quien los materialice.

Como demuestra la experiencia, los errores en el desarrollo normalmente son repetitivos, con lo que los mismos problemas aparecen una y otra vez en todas las aplicaciones que desarrolla una misma compañía. Esto hace vital la formación del personal de desarrollo y su concienciación respecto de los problemas de seguridad desde el mismo momento del desarrollo.

La posibilidad de integración de herramientas de análisis de código con los Entornos Integrados de Desarrollo (IDE – *Integrated Development Environment*) hace que los equipos de desarrollo tengan siempre presente la seguridad en el código que generan. De esta forma, también se consigue la autoformación de los propios desarrolladores y su adaptación a los requisitos comunes de seguridad que pueda imponer la compañía en caso de contar con equipos de desarrollos heterogéneos (equipos de desarrollo internos y equipos de desarrollo con personal externo de distintas compañías).

Criterios de evaluación

Teniendo en cuenta las premisas del punto anterior, son vitales en la elección de una herramienta de análisis de código apropiada para una compañía dos aspectos fundamentales: los lenguajes de programación soportados y la interacción con los IDEs.

Lógicamente, los lenguajes soportados constituyen el factor que de entrada hará que una organización se incline por una u otra herramienta concreta. En este sentido, con el objetivo de tener un analizador de código centralizado, en la elección es importante que la herramienta en cuestión integre la mayor parte de los lenguajes utilizados en el global de la aplicación. Evidentemente, lo deseable es que soporte los que se utilizan en todos los niveles: presentación estática y dinámica, acceso a datos e interacción con otros servicios.

Además del soporte de la herramienta para los lenguajes que se utilizan en el momento de la selección, se debe tener en cuenta la adaptabilidad de ésta a otros lenguajes que, aunque no se usen actualmente, sí que podrían aparecer en un futuro. Este aspecto cada día cobra más fuerza debido a las posibilidades de abstracción del lenguaje que persiguen los nuevos entornos de servicios web.

Respecto de la integración con los IDEs, la capacidad de interacción de las herramientas con los entornos con los que se trabaja en la organización, determina el nivel de participación del personal de desarrollo en el ciclo de análisis, y con ello, la agilidad de esta fase. Un analizador automatizado de código integrado en un IDE permitirá a los desarrolladores realizar análisis parciales de sus secciones de código para conseguir así que ésta no contenga errores antes de integrarla con otros componentes y secciones de la aplicación.

Estas dos cuestiones implicarán la aceptación o descarte de entrada de un conjunto de herramientas de análisis. A partir de aquí, las funcionalidades específicas de cada herramienta servirán como factores de elección para inclinarse por una u otra solución concreta.

En primer lugar, hay que considerar la tecnología de análisis que utiliza la herramienta. Se distinguen inicialmente tres técnicas de análisis, que determinarán las capacidades de detección de problemas en las aplicaciones web:

• Análisis de flujo de datos: consiste en identificar los parámetros de entrada de las aplicaciones (por tanto, manipulables por un atacante) y detectar la ausencia de validaciones en cualquier punto de la aplicación en el que se trate.

• Análisis semántico: valida el uso de código potencialmente peligroso por sí mismo, esto es, llamadas a funciones y procedimientos vulnerables.

• Análisis estructural: se encarga de detectar problemas debidos a una incorrecta organización de sentencias en el código.

Otro aspecto de diferenciación entre herramientas es su capacidad de detección de problemas, entendiendo como tal el conjunto de vulnerabilidades que son capaces de descubrir, su categorización y la gestión que permiten sobre los problemas detectados. Es importante que, una vez realizado el análisis, se permitan gestionar de forma ágil los falsos positivos y riesgos asumidos que resultan de éste, para evitar que vuelvan a mostrarse en futuros análisis.

Además del descubrimiento de problemas, se debe considerar en la evaluación de cada herramienta de análisis las capacidades de administración de las comprobaciones encargadas de detectarlos. De esta forma, la interfaces de gestión de chequeos y la posibilidad de personalización de éstos, ya sea añadiendo nuevas comprobaciones o personalizando las incluidas por defecto, es otro punto de diferenciación entre los analizadores de código. Por supuesto, la herramienta debería permitir siempre la posibilidad de ajustar el nivel de riesgo de cada vulnerabilidad identificada (para descartar aquellas que hayan identificado mal), e incluso, poder aplicar plantillas diferentes a distintos análisis, ya que el nivel de riesgo o las comprobaciones a realizar podrán no ser las mismas para unas y otras aplicaciones.

Existen ciertas relaciones entre los componentes de las aplicaciones que un analizador automático no puede contemplar, y requieren de un análisis inteligente para detectar el problema y poder así ofrecer recomendaciones.

Una vez detectados los problemas, la calidad de las recomendaciones para su mitigación es otro factor a tener en cuenta. Lo deseable es que las recomendaciones que se ofrecen sean concretas y personalizadas para el lenguaje de programación al que afecta el problema en cuestión. Por ejemplo, ante la detección de una vulnerabilidad de inyección de código, no bastará con recomendar que se validen los parámetros de entrada, sino que además se tendrá que especificar cómo y dónde realizar esa comprobación de parámetros para el lenguaje utilizado.

Igualmente, hay una serie de factores añadidos que ayudan a decantarse por una u otra herramienta de análisis de código. Algunos de estas características auxiliares de enumeran a continuación:

• Capacidades de análisis incremental: evita el procesamiento de todo el código en cada análisis, teniendo en cuenta análisis previos.

• Calidad de informes generados: la herramienta debe ser capaz de proporcionar informes técnicos detallados con los problemas detectados e informes generales de carácter ejecutivo.

• Revisión evolutiva de los análisis: se tendrán en cuenta las posibilidades de estudiar los avances en la generación de código seguro por los equipos de desarrollo, comparar el resultado del análisis a varias aplicaciones o analizar tendencias en la seguridad de las aplicaciones de la organización.

• Posibilidades de gestión de informes: se considerarán los mecanismos de distribución de informes a todas las partes que intervienen en la fase de análisis. Se distinguen varias opciones como interfaces web autenticadas, envío por correo integrados en los entornos de desarrollo, etc.

• Referencias estandarizadas: el enlace de los problemas encontrados a referencias compatibles CWE (*Common Weakness Enumeration*) ayuda a identificar exactamente cada problema y facilita su investigación.

Situación actual

Desde la aparición del que se considera el primer analizador de código fuente (*Lint*) allá por 1979, estas herramientas han evolucionado mucho.

Hoy en día existe un amplio abanico de productos, propietarios y libres, de pago y gratuitos, que abarca un espectro extenso de funcionalidades y lenguajes. Tradicionalmente las herramientas de análisis de código se han centrado en el descubrimiento de problemas en código orientado a otros entornos distintos al de aplicaciones y servicios web en Internet, tales como sistemas operativos, desarrollo de aplicativos, etc. Sin embargo, en los últimos tiempos están apareciendo bastantes analizadores de código automatizados orientados a entornos web. Muchos de ellos son evolución de las herramientas automáticas enfocadas a realizar pruebas de intrusión en estos entornos, e incluso de los mismos fabricantes. Aunque la mayoría de estas herramientas ofrecen funcionalidades limitadas y soportan un conjunto de lenguajes y entornos de programación muy concretos, existen productos muy completos que permiten su perfecta integración en los ciclos de desarrollo de grandes compañías, tal y como se muestra en la **figura 2**.

Por otro lado, se produce otra corriente paralela a la aparición y perfeccionamiento de este tipo de productos, que pretende sentar las bases para conseguir unos requisitos mínimos a cumplir por este tipo de herramientas. Es el caso de iniciativas como el proyecto SAMATE (*Software Assurance Metrics And Tool Evaluation*), encabezada por el Grupo de Análisis y Pruebas de Conformidad de Software (*Software Diagnostics and Conformance Testing Division*) del NIST **[5].**

Este proyecto tiene como objetivo identificar un conjunto común de debilidades que suelen aparecer en la mayoría de los desarrollos, con el fin de fijar unos mínimos a cumplir por cualquier herramienta de análisis de código. Las especificaciones del propio proyecto hacen hincapié en que el cumplimiento de estos mínimos no garantiza que el código esté libre de vulnerabilidades, y que a partir de este límite, son las propias herramientas de análisis las encargadas de ofrecer mejoras en el descubrimiento de debilidades que permitan maximizar la seguridad del código final.

Herramienta	Fabricante	Lenguajes soportados	Integración en IDE
.TEST	Parasoft	C#, VB.NET, MC++	Microsoft Visual Studio
JTEST	Parasoft	Java	Eclipse, RAD, IntelliJ IDEA, Oracle JDeveloper
WebKing	Parasoft	HTML	WebSphere Studio Application Develope, Eclipse
CodeAssure	Secure Software	C, C++, Java	Eclipse, MS Visual Studio
CodeScan	CodeScan Labs	.ASP PHP	-
DevInspect	SPI Dynamics	C#, Visual Basic	Micrsoft Visual Studio 2003 y 2005, Eclipse, IBM Rational Application Developer 6 y 7.
Jlint	OpenSource	Java	-
K7	Klocwork	C, C++, Java	Eclipse, QNX Momentics, GVim, Emacs, Visual SlickEdit, Platform Builder, KDevelop, MetroWerks CodeWarrior, MS Visual Studio, Wind River Workbench
LAPSE	OpenSource	Java	Eclipse
PHP-SAT	OpenSource	PHP	-
PMD	OpenSource	Java	JDeveloper, Eclipse, JEdit, JBuilder, BlueJ, CodeGuide, NetBeans/ Sun Java Studio Enterprise/Creator, IntelliJ IDEA, TextPad, Maven, Ant, Gel, JCreator, Emacs, WebLogic Workshop 8.1.x
Prexis	Ounce Labs	C, C++, Java, JSP, J2EE, STRUTS	Microsoft Visual Studio, Eclipse
Resource Standard Metrics	M Squared Technologies	C, C++, C#, Java	
SoftCheck Inspector	SofCheck	Java	-
Source Code Analysis (SCA)	Fortify Software	ASP.NET, C, C++, C# y otros lenguajes .NET, Java, JSP, PL/SQL, T-SQL, VB.NET, XML	Microsoft Visual Studio, Eclipse, WebSphere Application Developer,IBM Rational Application Developer
Prevent	Coverity	C, C++	Micrsoft Visual Studio, Eclipse, Wind River Workbench, Sun CC

Figura 2

Conclusiones

La complejidad de los entornos de aplicaciones web actuales y futuros, y las dificultades y riesgos asociados con el arreglo de problemas de seguridad cuando las aplicaciones ya están en producción, hace necesario introducir una fase de análisis de código en el ciclo de vida del desarrollo de estas aplicaciones. Esta tarea es eficiente únicamente mediante el uso de herramientas automatizadas, que deberán adaptarse al entorno específico de cada organización y a las aplicaciones que ésta desarrolla.

Esto plantea un reto en dos vertientes. El primero, que las propias herramientas cumplan con lo que se espera de ellas, ofreciendo un análisis de código con calidad y adaptándose a los nuevos recursos que aparecen cada día en el mundo cambiante de las aplicaciones web en Internet. El segundo, orientado a los departamentos de seguridad, para que sean capaces de introducir una nueva fase de control relacionada con la seguridad en los ciclos de desarrollo de sus organizaciones.

Cumplir con estos retos no es fácil, y exige un cambio de mentalidad en el que la seguridad ya no está "al final" sino "durante" el proceso. Las organizaciones tardarán en introducir este cambio, sólo hace falta recordar lo que ha costado introducir la necesidad de ejecutar pruebas de intrusión previas a los pasos a producción con el objeto de validar la seguridad de las aplicaciones.

En cualquier caso, no hay que pensar que el uso de analizadores automáticos de código es la panacea para garantizar el desarrollo de aplicaciones 100% seguras. Del mismo modo que las herramientas automáticas de búsqueda de vulnerabilidades no cubren todos los aspectos de una auditoría técnica de seguridad, las herramientas de análisis de código no tienen el conocimiento suficiente para descubrir algunos de los problemas de seguridad más complejos que pueden afectar a una aplicación y que pueden surgir de un diseño incorrecto. Existen ciertas relaciones entre los componentes de las aplicaciones que un analizador automático no puede contemplar, y requieren de un análisis inteligente para detectar el problema y poder así ofrecer recomendaciones. Y, por supuesto, estas herramientas poco sabrán del entorno legal y normativo que algunas compañías deben cumplir. ■

Joaquín Crespo Pérez.
División de Seguridad e Infraestructuras de Redes Corporativas
GERMINUS XXI (GRUPO GESFOR)
jcrespo@germinus.com

REFERENCIAS

[1] SANS Top-20 Internet Security Attack Targets, disponible en http://www.sans.org/top20/
[2] Proyecto OWASP Top Ten, disponible en: http://www.owasp.org/index.php/OWASP_Top_Ten_Project
[3] The Web Hacking Incidents Database, del Web Application Security Consortium, disponible en: http://www.webappsec.org/projects/whid/
[4] "Secure Business Quaterly", volumen uno, número 2, Q4 2001: "Tangible ROI through software engineering", por Kevin Soo Hoo, Andrew W. Sudbury y Andrew R. Jaquith
[5] SAMATE - Software Assurance Metrics And Tool Evaluation, disponible en: http://samate.nist.gov.
[6] Artículos relacionados con el proceso de desarrollo seguro en el ciclo de vida de aplicaciones, elaborados por el *Software Engineering Institute*, disponibles en: https://buildsecurityin.us-cert.gov/daisy/bsi/articles/knowledge/sdlc.html
[7] Artículo "Seguridad de aplicaciones: una mirada al proceso de desarrollo" de Luis Rodríguez Berzosa. Revista SIC nº 73.