Análisis de las Herramientas para Realizar Pruebas Estáticas de Seguridad de las Aplicaciones

Jorge Eterovic; Valeria Silvestri; Andrea Vera; Martin Zeballos; Alesio Esteban Sinopoli
Departamento de Ingeniería e Investigaciones Tecnológicas Universidad Nacional de La Matanza
Florencio Varela 1903 (B1754JEC), San Justo, (5411) 4480-8900

{eterovic; vsilvestri; avera; mzeballos}@unlam.edu.ar;
asinopoli@alumno.unlam.edu.ar

RESUMEN

Las pruebas estáticas de seguridad de las aplicaciones que se utilizan para proteger el software se denominan SAST (Static Application Security Testing) y consisten en la revisión automática del código fuente para identificar patrones vulnerables.

Las herramientas SAST permiten automatizar la detección de vulnerabilidades y se pueden integrar al sistema de CI/CD (integración continua / distribución continua) para que detecten vulnerabilidades en etapas tempranas del ciclo de vida. Esto ayuda al equipo de Seguridad de las Aplicaciones a implementar un ciclo de vida del desarrollo de software seguro.

CI/CD es un método para distribuir las aplicaciones a los clientes mediante el uso de la automatización en las etapas del desarrollo de las aplicaciones. En este contexto, cualquier equipo de Seguridad de las Aplicaciones se enfrentará al desafío de automatizar los chequeos de seguridad y encontrará la solución en herramientas SAST.

Integrar una herramienta SAST al proceso de CI/CD permite detectar vulnerabilidades en la etapa de desarrollo, en vez de esperar a la etapa de prueba o que se detecten directamente en producción.

Hay disponibles herramientas SAST gratuitas para los repositorios open-source y pagas para los repositorios privados. Algunas son open-source, otras usan un motor privado pero las reglas son open-source y algunas pocas son totalmente privadas.

Este proyecto de investigación propone hacer un análisis de estas herramientas SAST y aportar los resultados obtenidos a la comunidad open-source para mejorar la seguridad de los repositorios de proyectos de desarrollo de software que las utilizan.

Palabras Clave: *Herramienta SAST; Detección de Vulnerabilidades; Análisis Estático; Seguridad de las Aplicaciones..*

CONTEXTO

Este proyecto de investigación se desarrolla en el marco de un Programa de Incentivos a Docentes Investigadores de la Secretaría de Políticas Universitarias (PROINCE) del Ministerio de Educación, y se ejecuta en el Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza.

El proyecto es financiado por el propio Departamento y es del tipo investigación aplicada. El mismo propone hacer un análisis de las herramientas SAST y aportar los resultados obtenidos a la comunidad open-source. Los trabajos de campo y relevamientos realizados aportaron información valiosa y sirvieron como base para el presente trabajo.

1. INTRODUCCIÓN

Las herramientas SAST permiten automatizar la detección de vulnerabilidades y se pueden integrar al sistema de CI/CD para que detecten vulnerabilidades en etapas tempranas del ciclo de vida del desarrollo del software. Esto ayuda al equipo de Seguridad de las Aplicaciones a implementar un ciclo de vida seguro.

CI/CD es un método para distribuir las aplicaciones a los clientes mediante el uso de la automatización en las etapas del desarrollo de las aplicaciones. Los principales conceptos que se le atribuyen son la integración, la distribución y la implementación continuas. Se trata de una solución para los problemas que la integración de código nuevo puede generar a los equipos de desarrollo y de operaciones.

El proceso de integración y distribución continuas incorpora la automatización y la supervisión permanentes en todo el ciclo de vida de las aplicaciones, desde las etapas de integración y prueba hasta las de distribución e implementación. En este contexto, cualquier equipo de Seguridad de las Aplicaciones se enfrentará al desafío de automatizar los chequeos de seguridad y encontrará la solución en herramientas SAST.

Integrar una herramienta SAST al proceso de CI/CD permite detectar vulnerabilidades en la etapa de desarrollo, en vez de esperar a la etapa de prueba o que se detecten directamente en producción. Una vulnerabilidad en producción implica un riesgo constante, cuesta mucho esfuerzo de los expertos en seguridad detectarla y para los desarrolladores es difícil de corregir. En cambio, si se detecta durante la etapa de desarrollo, nunca generó un riesgo real, no requirió esfuerzo de personas de seguridad para detectarla y es mucho más fácil de corregir.

Hay muchas herramientas SAST y el análisis estático está muy vinculado al lenguaje de programación. Una herramienta puede analizar varios lenguajes, pero en realidad agregar un lenguaje nuevo a la herramienta, es desarrollar un producto nuevo. Es decir, una misma herramienta va a tener distintos grados de madurez, distintas características y distintas limitaciones según el lenguaje a analizar.

Un criterio de comparación entre distintas herramientas SAST es el grado de complejidad. Una herramienta simple se ejecuta rápidamente, soporta código que no compila y es fácil de aprender a usar, pero no es precisa, da muchos falsos positivos y falsos negativos incorregibles porque no maneja la información necesaria para refinar los resultados.

Por otro lado, una herramienta compleja se ejecuta más lentamente, tiene requisitos extras como por ejemplo que el código sea compilable y completo y lleva tiempo aprender a usarla, pero a su vez como maneja mucha más información y esta se puede usar para evitar falsos positivos y falsos negativos.

Hay bastante variedad entre las herramientas SAST. Algunas son open-source, otras usan un motor privado pero las reglas son open-source y algunas pocas son totalmente privadas. Algunas incluso distinguen su sistema de precios según el código a analizar, siendo gratuitas para los repositorios open-source y pagas para los repositorios privados. Y por último hay sistemas que simplemente se encargan de integrar y ofrecer varias herramientas. Por ejemplo, GitHub tiene una sección de escaneo de código denominada “Code Scanning” con muchas herramientas SAST.

2. LÍNEAS DE INVESTIGACION Y DESARROLLO

Se desarrollan a continuación los aspectos teóricos de este proyecto de investigación:

- DevOps
- DevSecOps
- Pruebas Estáticas de la Seguridad de la Aplicación (SAST)
- Integración Continua y Despliegue Continuo (CI/CD)

DevOps

Existen muchas definiciones diferentes de DevOps disponibles en libros, en artículos de revistas o en Internet. A raíz de esta disparidad en las definiciones, surgen diversos estudios que tratan de darle una descripción académica [1] [2]. Según estos estudios, podemos definir DevOps como una cultura que trata de aunar a los equipos de desarrollo y operaciones, basándose en una serie de principios y prácticas [2] que pretenden acelerar las entregas del producto mejorando el feedback de los clientes y la capacidad de reacción ante los cambios [3].

El término DevOps surge a finales de los 2000, en un contexto en el que las metodologías de desarrollo ágiles cada vez tomaban mayor relevancia en la industria del desarrollo de software. La velocidad a la que se desarrollaban nuevas características o se corregían bugs distaba mucho de la velocidad a la que se realizaban los despliegues de estos cambios, con lo que el ciclo

de desarrollo del software se veía ralentizado. Esta ralentización era resultado de la falta de comunicación existente entre el equipo de desarrollo y el de operaciones.

Fue Patrick Debois quien, en 2007, tras una experiencia frustrante trabajando en la migración de un gran centro de datos, se percató de cómo esta falta de comunicación entre desarrolladores y administradores de sistemas afectaba al flujo de trabajo [4]. En 2009, John Allspaw y Paul Hammond, ingenieros de Flickr, presentaron su charla "10 Deploys a Day: Dev and Ops Cooperation at Flickr" [5] donde propusieron integrar desarrollo y operaciones en un flujo automatizado. Patrick Debois, tras esta conferencia, decidió organizar una similar en Bélgica, a la que llamó DevOpsDays, de donde surge el término DevOps [6].

Un aspecto importante para seguir exitosamente la cultura DevOps es la automatización de procesos, al ser lo que permite mantener la agilidad durante el desarrollo del software. La importancia de la automatización de procesos ha hecho surgir una gran cantidad de herramientas que contemplan la construcción del software, la integración y el despliegue continuos, la gestión de logs y la monitorización [7].

DevSecOps

El crecimiento de las metodologías ágiles de desarrollo del software y la acogida de la cultura DevOps por parte de las organizaciones ha incrementado la velocidad a la que las aplicaciones reciben actualizaciones. Esto tiene grandes ventajas, pues permite tener feedback temprano del cliente para mejorar el producto desde las primeras fases del desarrollo, mejorando la adaptabilidad del producto con el entorno y permitiendo marcar la diferencia con la competencia gracias a la implementación de nuevas funcionalidades y la mejora del funcionamiento de las ya existentes [8]. Sin embargo, a veces esta agilidad se consigue a costa de sacrificar otros aspectos del producto final, como puede ser la seguridad [9]. Los estudios muestran que menos de un 20% de las compañías que siguen la cultura DevOps tienen en cuenta la seguridad como parte del ciclo de desarrollo del software [5].

DevSecOps surge para incluir la seguridad en DevOps, alineando los equipos de desarrollo, de operaciones y de seguridad durante todo el ciclo de desarrollo. Esto se consigue desplazando la seguridad a la izquierda, es decir, considerándola desde las primeras etapas del desarrollo [9]. De esta manera, al tenerla en cuenta desde el diseño de la aplicación, es posible realizar los controles de seguridad necesarios a lo largo del ciclo de desarrollo del software y automatizarlos para que sean rápidos, escalables y efectivos [10]. De esta manera se mantiene la agilidad en el desarrollo del software y se detectan desde fases tempranas los fallos de seguridad que, de llegar al cliente, conllevarían grandes pérdidas de tiempo y dinero.

Al igual que en DevOps, las herramientas juegan un papel de gran importancia. Existen una gran cantidad de herramientas para llevar a cabo DevSecOps. Se ha tomado como referencia la guía OWASP DevSecOps Guideline [11] para establecer los aspectos más importantes relativos a la seguridad: la detección de secretos, las Pruebas Estáticas de la Seguridad de la Aplicación (SAST), las Pruebas Dinámicas de la Seguridad de la Aplicación (DAST), el escaneo de la infraestructura y la comprobación del cumplimiento normativo.

Pruebas Estáticas de la Seguridad de la Aplicación (SAST)

Las pruebas estáticas de seguridad analizan el código fuente de la aplicación sin ejecutarlo, tratando así de encontrar vulnerabilidades o bugs [12]. Los análisis estáticos se pueden realizar de diferentes formas, desde las más sencillas y rápidas que contemplan sólo un análisis del código fuente en base al árbol sintáctico, hasta las más complejas que combinan diversas representaciones del código como grafos de control y flujo de datos para realizar un análisis semántico en busca de patrones vulnerables [12].

Los factores a tener en cuenta a la hora de decantarse por una herramienta u otra son la velocidad a la que se quiere realizar el análisis y la profundidad del mismo, pues a más profundidad, más se tardará en realizar y viceversa. Además, se ha de considerar el porcentaje de falsos positivos que puede señalar la herramienta y el lenguaje de programación de la aplicación. Las herramientas SAST ayudan a detectar vulnerabilidades como inyecciones SQL, cross-site scripting (XSS) y problemas de gestión de memoria, entre otras.

Algunos ejemplos de estas herramientas son: Semgrep, CodeSonar o CodeQL[13] [14].

Integración Continua y Despliegue Continuo (CI/CD)

La integración continua (CI) surge como una de las prácticas de la metodología ágil Extreme Programming (XP), en la cual se propone que los desarrolladores publiquen sus cambios varias veces al día en el repositorio de código. De esta forma pueden encontrarse problemas de compatibilidad entre estos cambios en etapas más tempranas del desarrollo, y se evitan complejos y largos procesos de integración en los días anteriores de la fecha de entrega del proyecto o del hito [15]. Gracias a estas ventajas, la integración continua se utiliza como práctica independiente de XP, respaldada por referentes en el mundo del desarrollo del software como Martin Fowler [16].

El despliegue continuo (CD) amplía las bases propuestas por la integración continua, proponiendo no solo la integración automatizada del código en el repositorio, sino también el despliegue automatizado del mismo en el entorno de producción [17]. La automatización del despliegue es especialmente beneficiosa cuando existen varios entornos en los que se ha de desplegar el software cuando se genera una nueva versión, y cuando este proceso de despliegue ocupa mucho tiempo [18]. Cabe clarificar las diferencias entre la entrega continua y el despliegue continuo, conceptos que en ocasiones se confunden. Mientras la entrega continua tiene como objetivo mantener siempre el software en un estado que permite su despliegue inmediato [19], el despliegue continuo implica el despliegue de las nuevas versiones del software de forma automática.

Los procesos automatizados de integración y de despliegue pueden tardar varios minutos en ejecutarse, ralentizando el flujo de desarrollo. Además, requieren de una gran coordinación por parte de los desarrolladores para evitar conflictos en los cambios y en el orden en que se realizan las integraciones. Por estos motivos, desde la metodología XP se propone el uso de un servidor dedicado a realizar las integraciones. Este servidor, denominado servidor de integración continua, asegura la realización de los procedimientos necesarios para integrar los nuevos cambios de manera ordenada y evitando conflictos [20] y su elección es clave para llevar a cabo con éxito estos procesos.

Existen varias alternativas en el mercado, siendo GitLab CI, Jenkins y GitHub Actions los servidores de integración continua más destacables. Tanto GitLab CI como GitHub Actions forman parte del ecosistema de los gestores de repositorios GitLab y GitHub respectivamente, por lo que son los que se tendrán en cuenta en este proyecto, evitando así la dependencia de una herramienta adicional para este propósito.

3. RESULTADOS OBTENIDOS/ESPERADOS

El objetivo general es analizar las herramientas open-source disponibles para realizar las pruebas estáticas de seguridad de las aplicaciones (SAST).

Los objetivos específicos son:

- Comparar las herramientas SAST open-source para determinar sus fortalezas y debilidades.
- Desarrollar casos de ejemplo para analizar qué vulnerabilidad encuentra cada herramienta SAST y establecer sus limitaciones. Esta tarea se realiza para cada lenguaje de programación.
- Analizar las vulnerabilidades para determinar si hay patrones detectables en el código o no.
- Clasificar las vulnerabilidades para asistir a los expertos de seguridad sobre cuándo usar herramientas SAST y cuándo no.
- Desarrollar una regla en una herramienta SAST para un lenguaje específico para detectar una vulnerabilidad determinada.

4. FORMACIÓN DE RECURSOS HUMANOS

El equipo de trabajo de este proyecto está formado por dos ingenieras y un licenciado en informática, un especialista en seguridad teleinformática y un alumno avanzado de la carrera. Este trabajo se desarrolló en el marco del proyecto de investigación: “Análisis de las Herramientas SAST”.

Dada la complejidad del desarrollo del proyecto de investigación, fue necesaria la colaboración de

varios expertos con amplia experiencia en la industria y la investigación académica.

5. BIBLIOGRAFIA

[1] De Franca, B. B. N., Jeronimo, H., & Travassos, G. H. (2016). Characterizing DevOps by Hearing Multiple Voices. Proceedings of the 30th Brazilian Symposium on Software Engineering - SBES '16. Published. https://doi.org/10.1145/2973839.2973845

[2] Jabbari, R., bin Ali, N., Petersen, K., & Tanveer, B. (2016). What is DevOps? Proceedings of the Scienti_c Workshop Proceedings of XP2016. Published. https://doi.org/10.1145/2962695.2962707

[3] Virani, M. (2015). Understanding DevOps & bridging the gap from continuous integration to continuous delivery. Fifth International Conference on the Innovative Computing Technology (INTECH 2015). Published. https://doi.org/10.1109/intech.2015.7173368

[4] Watts, S. (2019, 29 March). A Brief History of DevOps. BMC Blogs. https://www.bmc.com/blogs/ devops-history/

[5] Allspaw, J., & Hammond, P. (2009, Jun 22). 10+ Deployes per Day: Dev and Ops Cooperation at Flickr [Talk]. O'Reilly Velocity Conference, San Jose, California. https://www.youtube.com/ watch?v=LdOe18KhtT4

[6] Debois, P. (s. f.). About devopsdays. DevOpsDays. Recuperado 18 de enero de 2022, de https://devopsdays.org/about/

[7] Akshaya, H. L., Nisarga Jagadish, S., Bidya, J., & Veena, K. (2015). A Basic Introduction to DevOps Tools. International Journal of Computer Science and Information Technologies, 6(3). http://ijcsit.com/docs/Volume%206/vol6issue03/ijcsit2015060382.pdf

[8] Beck, K., Fowler, M., Martin, R. C., Beedle, M., Cockburn, A., Cunningham, W., Thomas, D., Mellor, S., Schwaber, K., Sutherland, J., Bennekum, A. V., Grenning, J., Highsmith, J., Hunt, A., Je_ries, R., Kern, J., & Marick, B. (2001, 13 February). Principios del Manifiesto Ágil. Agile Manifesto. http://agilemanifesto.org/iso/es/principles.html

[9] Shackleford, D. (2016, 8 March). A DevSecOps Playbook. SANS. https://www.sans.org/webcasts/devsecops-playbook-101472

[10] Amazon Web Services. (2016, 9 November). Introduction to DevSecOps on AWS. https://www.slideshare.net/AmazonWebServices/introduction-todevsecops-on-aws-68522874

[11] The OWASP Foundation. (s. f.). OWASP DevSecOps Guideline. OWASP. Recuperado 24 de enero de 2022, de https://owasp.org/www-projectdevsecops-guideline/

[12] Adkins, H., Beyer, B., Blankinship, P., Lewandowski, P., Oprea, A., & Stubble _eld, A. (2020). Building Secure and Reliable Systems: Best Practices for Designing, Implementing, and Maintaining Systems (Illustrated ed.). O'Reilly Media. https://sre.google/books/building-secure-reliable-systems/

[13] Peterson, J. (2020, 19 November). Software Composition Analysis Explained. WhiteSource. https://www.whitesourcesoftware.com/resources/blog/softwarecomposition-analysis/

[14] Weerasinghe, M. (2019, 24 December). NodeJS Security Tools – Manjula Weerasinghe. Medium. https://medium.com/@manjula.aw/nodejs-securitytools-de0d0c937ec0

[15] Wells, D. (1999). Continuous Integration. Extreme Programming. http://www.extremeprogramming.org/rules/integrateoften.html

[16] Fowler, M. (2000, 10 September). Continuous Integration (original version). martinfowler.com. https://www.martinfowler.com/articles/originalContinuousIntegration.html

[17] Rahman, A. A. U., Helms, E., Williams, L., & Parnin, C. (2015). Synthesizing Continuous Deployment Practices Used in Software Development. 2015 Agile Conference. Published. https://doi.org/10.1109/agile.2015.12

[18] Humble, J., Read, C., & North, D. (2006). The Deployment Production Line. AGILE 2006 (AGILE'06). Published. https://doi.org/10.1109/agile.2006.53

[19] Fowler, M. (2013, 30 May). Continuous Delivery. martinfowler.com. https://martinfowler.com/bliki/ ContinuousDelivery.html

[20] Wells, D. (1999). Dedicated Release Computer. Extreme Programming. http://www.extremeprogramming.org/rules/dedicated.html