| | |
|---|---|
| **Código** | FPI-002 |
| **Objeto** | Protocolo de presentación de proyectos de investigación SIGEVA UNLaM |
| **Usuario** | Director de proyecto de investigación |
| **Autor** | Secretaría de Ciencia y Tecnología de la UNLaM |
| **Versión** | 4 |
| **Vigencia** | 12/11/2021 |

**Integrantes del equipo:**

**Lic. Martin Zeballos**

**Ing. Valeria Silvestri**

**Ing. Andrea Vera**

**Alesio Esteban Sinopoli (alumno)**

**Fecha de inicio: 01/01/2022**

**Fecha de finalización: 31/12/2023**

**1-Cuadro resumen de horas semanales dedicadas al proyecto por parte de director e integrantes del equipo de investigación:**[2]

| Rol del integrante | Nombre y Apellido | Cantidad de horas semanales dedicadas al proyecto |
|---|---|---|
| Director | Jorge Eterovic | 4 |
| Co-director | | |
| Director de Programa | | |
| Docente-investigador UNLaM | Martin Zeballos | 4 |
| | Valeria Silvestri | 4 |
| | Andrea Vera | 4 |
| Investigador externo[3] | | |
| Asesor-Especialista externo[4] | | |
| Graduado de la UNLaM[5] | | |
| Estudiante de carreras de posgrado (UNLaM)[6] | | |
| Alumno de carreras de grado (UNLaM)[7] | Alesio Esteban Sinopoli | 4 |
| Personal de apoyo técnico administrativo | | |

**2-Plan de investigación**

2. Tipo de actividad I+D: APLICADA

2.1. Resumen del Proyecto:

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/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. 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.

[2] Incluir todos los integrantes del equipo de investigación, agregando tantas filas para cada rol de integrante del equipo de investigación como sea necesario.

[3] Deberá adjuntar FPI 28, 29 y 30 debidamente firmados.

[4] Idem nota 2.

[5] Idem nota 2

[6] Adjuntar certificado de materias aprobadas de estudiantes de carrera de posgrado.

[7] Adjuntar certificado de materias aprobadas de estudiantes de carrera de grado.

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 disponibles herramientas SAST gratuitas para los repositorios open-source y pagas para los repositorios privados. Por ejemplo, GitHub es una plataforma de desarrollo colaborativo para alojar proyectos de desarrollo de software utilizando un sistema de control de versiones y tiene la sección Code Scanning ("escaneo de código" en español) con muchas herramientas SAST. 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.

2.2. Palabras clave: HERRAMIENTA SAST; DETECCIÓN DE VULNERABILIDADES, ANÁLISIS ESTÁTICO,

2.3 Resumen del Proyecto (ingles):

Static Application Security Testing (SAST) is a testing method to find security issues in software applications by scanning its code searching for vulnerable patterns.

SAST tools allow you to automate the detection of vulnerabilities and can be integrated into a CI/CD (Continuous Integration and Continuous Delivery) environment to detect such vulnerabilities in an early stage of the SDLC (Software Development Life Cycle). Application Security teams use these and other tools to implement a Secure SDLC (SSDLC).

CI/CD is the combined practices of continuous integration and continuous delivery with the goal of providing faster release cycles by enforcing automation in building, testing and deployment of applications. The integration of new code can cause many problems to both development and operations teams, but CI/CD prevents most of these with automation and constant supervision of the SDLC. In this context, any Application Security team will face the challenge of automating security checks and will find the solution in SAST tools.

SAST tools integrated to a CI/CD process can detect vulnerabilities during the coding stage, as opposed to finding them during the testing stage or directly in production A vulnerability in production implies a constant risk, it costs a lot of effort for security experts to detect it and it is difficult for developers to correct it. However, if found during the coding stage by a SAST tool, this vulnerability never implied a real risk, didn't require any effort at all from the security experts and will be much easier to fix.

We can find many SAST tools out there. They are usually free to use on open-source repositories but require a paid license for private ones. For example, GitHub is a provider of Internet hosting for software development and version control, many open-source repositories are stored on their platform, and they have the Code Scanning section featuring many SAST tools. Some of these are 100% open source, some use a private engine, but their rules are open source, and just a few are very private.

This investigation project aims to analyze these SAST tools and contribute its results to the open-source community, helping to improve their software's security.

2.4 Palabras clave (ingles): SAST, VULNERABILITY DETECTION, STATIC ANALYSIS.

2.5 Disciplina desagregada: 1799 - INFORMATICA-OTRAS

2.6 Campo de aplicación: 12 - OTROS CAMPOS

2.7 Especialidad: SEGURIDAD DE LA INFORMACIÓN

2.8 Estado actual del conocimiento:

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 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, no escrito a medias y lleva tiempo aprender a usarla, pero como maneja mucha información, son evitables muchos falsos positivos y falsos negativos.

2.9. Problemática a investigar:

Desarrollar una regla e integrarla en una herramienta SAST para que permita detectar vulnerabilidades en la etapa de desarrollo del software, en vez de esperar a la etapa de prueba o que se detecte 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.

2.10. Objetivos:[8]

Objetivo General:

- Analizar las herramientas open-source disponibles para realizar las pruebas estáticas de seguridad de las aplicaciones (SAST)

Objetivos específicos:

- 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.
- Clasificar las vulnerabilidades para asistir a los expertos de seguridad sobre cuándo usar herramientas SAST y cuándo no.
- Analizar las vulnerabilidades para determinar si hay patrones detectables en el código o no.
- Desarrollar una regla en una herramienta SAST para un lenguaje específico para detectar una vulnerabilidad determinada.

2.11. Marco teórico:

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

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, 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 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

[8] Detallar objetivo general y objetivos específicos.

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.

2.12. Hipótesis de trabajo o los supuestos implícitos (según corresponda al diseño metodológico):[9]

El desarrollo e incorporación de una regla en una herramienta SAST open-source para un lenguaje de programación determinado puede detectar una vulnerabilidad mejorando la seguridad de los repositorios de proyectos de desarrollo de software.

2.13. Metodología:

Se estudiarán y analizarán las distintas herramientas SAST open-source disponibles en los repositorios. Luego se desarrollarán casos de ejemplo para analizar qué vulnerabilidad encuentra cada herramienta.

Se analizarán las vulnerabilidades para determinar si hay patrones detectables en el código para desarrollar una regla en una herramienta SAST para un lenguaje específico para detectar una vulnerabilidad determinada

[9] En proyectos de desarrollo tecnológico puede ser reemplazada una hipótesis de trabajo por la propuesta de solución al problema de investigación mediante el diseño de un prototipo o elemento equivalente.

2.14. Bibliografía:

[1] de Fran_ca, 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

2.15. Programación de actividades (Gantt):[10]

(1) Jorge Eterovic; (2) Martin Zeballos; (3) Valeria Silvestri; (4) Andrea Vera; (5) Alesio Sinopoli

| Año 2022 | Mes 1 | Mes 2 | Mes 3 | Mes 4 | Mes 5 | Mes 6 | Mes 7 | Mes 8 | Mes 9 | Mes 10 | Mes 11 | Mes 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Análisis Inicial Revisión bibliográfica | (1) (2) (3) (4) | (1) (2) (3) (4) | (1) (2) (3) (4) | (1) (2) (3) (4) | | | | | | | | |
| Desarrollo de ejemplos | | | | (1) (2) (3) | (1) (2) (3) | (1) (2) (3) | | | | | | |

[10] Definir la programación de actividades para cada objetivo específico, y las personas responsables de su ejecución.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | (4)(5) | (4)(5) | (4)(5) | | | | | | |
| **Clasificación de vulnerabilidades** | | | | | | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | | | |
| **Análisis de las vulnerabilidades** | | | | | | | | | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) |
| **Redacción de Informe avance** | | | | | | | | | | | (1)<br>(2)<br>(3)<br>(4) | (1)<br>(2)<br>(3)<br>(4) |

| **Año 2023** | **Mes 1** | **Mes 2** | **Mes 3** | **Mes 4** | **Mes 5** | **Mes 6** | **Mes 7** | **Mes 8** | **Mes 9** | **Mes 10** | **Mes 11** | **Mes 12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Diseño de la regla** | (1)<br>(2)<br>(3)<br>(4) | (1)<br>(2)<br>(3)<br>(4) | | | | | | | | | | |
| **Desarrollo de la regla** | | (1)<br>(2)<br>(3)<br>(4) | (1)<br>(2)<br>(3)<br>(4) | (1)<br>(2)<br>(3)<br>(4) | (1)<br>(2)<br>(3)<br>(4) | (1)<br>(2)<br>(3)<br>(4) | (1)<br>(2)<br>(3)<br>(4) | | | | | |
| **Prueba** | | | | | | | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | | |
| **Implementación** | | | | | | | | | | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) | (1)<br>(2)<br>(3)<br>(4)(5) |
| **Redacción de Informe final** | | | | | | | | | | | (1)<br>(2)<br>(3)<br>(4) | (1)<br>(2)<br>(3)<br>(4) |

2.16. Resultados en cuanto a la producción de conocimiento:

El presente proyecto permitirá a los docentes-investigadores de la UNLaM contar con una serie de conocimientos hasta el momento no abordados por ninguna de sus cátedras debido al carácter actual e innovador de la rama del conocimiento que se tratará.

Como parte del proyecto se organizarán actividades de divulgación y difusión por intermedio del Área de Comunicaciones del DIIT

2.17. Resultados en cuanto a la formación de recursos humanos:

El equipo está integrado por docentes que pertenecen a las cátedras de Criptografía y Seguridad en Redes de la carrera de Ingeniería en Informática de la UNLaM que se están formado como investigadores. Como parte del proyecto se organizarán actividades prácticas de capacitación en la metodología de investigación aplicada.

2.18. Resultados en cuanto a la difusión de resultados:

Se escribirán trabajos para difundir los resultados en diferentes congresos:

- WICC – Workshop de Investigadores en Ciencia de la Computación.
- CoNaIISI – Congreso Nacional de Ingeniería Informática y en Sistemas de Información.
- CACIC – Congreso Argentino de Ciencias Informáticas y Computación.
- Otros Congresos relacionados con la temática.

2.19. Resultados en cuanto a transferencia hacia las actividades de docencia y extensión:

- Los resultados del trabajo de investigación aportarán conocimientos y material para esta nueva temática que aún no es abordada en la carrera de Ingeniería en Informática y las Tecnicaturas Universitarias en Desarrollo Web y de Aplicaciones Móviles.
- Se brindarán charlas y jornadas de capacitación a alumnos, graduados y docentes de Ingeniería Informática, Tecnicaturas Universitarias en Desarrollo Web y para Dispositivos Móviles y carreras afines.

2.20. Resultados en cuanto a la transferencia de resultados a organismos externos a la UNLaM:

Se espera poder brindar soporte y capacitación a otros organismos privados y públicos de todo el país para que utilicen repositorios open-source de proyectos de desarrollo de software.

2.21. Vinculación del proyecto con otros grupos de investigación del país y del exterior:

Se espera vincular el proyecto con grupos de investigación de otras Universidades de Gestión Pública y Privada interesados en esta temática.

2.22. Destinatarios:

| Tipo de destinatario | | Subtipo de destinatario[11] | ¿Cuál? Especificar | Demandante[12] | Adoptante[13] |
|---|---|---|---|---|---|
| Sector Gubernamental | Gobiernos | Del Poder Ejecutivo nacional | | | |
| | | Del Poder Ejecutivo provincial | | | |
| | | Del Poder Ejecutivo municipal | | | |
| | Otras Instituciones gubernamentales | Poder Legislativo en sus distintas jurisdicciones | | | |
| | | Poder Judicial en sus distintas jurisdicciones | | | |
| Sector Salud | | Hospitales, centros comunitarios de salud y otras entidades del sistema de atención | | | |
| Sector Educativo | | Sistema universitario | UNLaM | X | X |
| | | Sistema de educación básica y secundaria | | | |
| | | Sistema de educación terciaria | | | |
| Sector Productivo | | Empresas | Desarrolladoras de Software | | X |
| | | Cooperativas de trabajo y producción | | | |
| | | Asociaciones del Sector | | | |
| Sociedad Civil | | ONG's y otras organizaciones sin fines de lucro | | | |
| | | Comunidades locales y particulares | Comunidad de Desarrolladores | | X |

**3-Recursos Existentes**[14]

| Descripción/ concepto | Cantidad | Observaciones |
|---|---|---|
| No aplica | | |

[11] Marcar con una X

[12] Demandante: entidad administrativa de gobierno nacional, provincial o municipal constituida como demandante externo de las tecnologías desarrolladas, que determina la necesidad del proyecto por su importancia social. Marcar con una X

[13] Adoptante: beneficiario o usuario en capacidad de aplicar los resultados desarrollados (organismos gubernamentales de ciencia y tecnología nacionales o provinciales; universidades e institutos universitarios de gestión pública o privada; empresas públicas o privadas; entidades administrativas de gobierno nacionales, provinciales o municipales; entidades sin fines de lucro; hospitales públicos o privados; instituciones educativas no universitarias; y organismos multilaterales. Marcar con una X

[14] Antes de confeccionar el presupuesto del proyecto, será necesario que el Director incluya en esta tabla si dispone de recursos adquiridos con fondos de proyectos anteriores (equipamiento, bibliografía, bienes de consumo, etc.) a ser utilizados en el proyecto a presentar, y además se recomienda consultar en la Unidad académica la disponibilidad de recursos existentes factibles de ser utilizados en el presente proyecto.

**4-Recursos financieros**[15]

| | Rubro | Año 1 | Año 2 | Total |
|---|---|---|---|---|
| Gastos de capital (equipamiento) | a) Equipamiento (1) | | | |
| | b) Licencias (2) | | | |
| | c) Bibliografía (3) | | | |
| | Total Gastos de Capital | $ 0,00 | $ 0,00 | $ 0,00 |
| Gastos corrientes (funcionamiento) | d) Bienes de consumo | 2.000 | 2.000 | 4.000 |
| | e) Viajes y viáticos (4) | 35.000 | 35.000 | 70.000 |
| | f) Difusión y/o protección de resultados (5) | | | |
| | g) Servicios de terceros (6) | | | |
| | h) Otros gastos (7) | | | |
| | Total Gastos Corrientes | $ 37.000,00 | $ 37.000,00 | $ 74.000,00 |
| | Total Gastos (Capital + Corrientes) | $ 37.000,00 | $ 37.000,00 | $ 74.000,00 |

***Aclaraciones sobre rubros del presupuesto***

d) Bienes de consumo: Elementos de librería: resma de papel, posters para presentar en congresos y tinta para impresora.

e) Viajes y viáticos: Viajes y viáticos en el país: Gastos de viajes para asistir a congresos nacionales a razón de uno por año a presentar los avances del proyecto de investigación. Incluye pasaje de avión al interior del país, traslados, alojamiento y comidas

**4.1-Orígen de los fondos solicitados**

| Institución | % Financiamiento |
|---|---|
| UNLaM | 100 |
| Otros (indicar cuál) | No aplica |

[15] Justificar presupuesto detallado. Para compras de un importe superior a $15000.- se requieren tres presupuestos. (Resolución Rectoral Nº177/2021.)

| | |
|---|---|
| **Código** | FPI-009 |
| **Objeto** | Guía de elaboración de Informe final de proyecto |
| **Usuario** | Director de proyecto de investigación |
| **Autor** | Secretaría de Ciencia y Tecnología de la UNLaM |
| **Versión** | 5 |
| **Vigencia** | 03/9/2019 |

**Departamento:**

## Departamento de Ingeniería e Investigaciones Tecnológicas

**Programa de acreditación:**

CyTMA2

**Título del proyecto:**

# Análisis de las Herramientas SAST

**Director:**

**Mag. Ing. Jorge Eterovic**

**Codirector:**

**Integrantes:**

**Lic. Martin Zeballos**

**Ing. Valeria Silvestri**

**Ing. Andrea Vera**

**Alumno de grado:**

**Alesio Esteban Sinopoli (**No tiene Beca UNLaM/CIN**)**

**Resolución Rectoral de acreditación: N° 618/2022**

**Fecha de inicio: 01/01/2022**

**Fecha de finalización: 31/12/2023**

| | |
|---|---|
| **Código** | FPI-009 |
| **Objeto** | Guía de elaboración de Informe final de proyecto |
| **Usuario** | Director de proyecto de investigación |
| **Autor** | Secretaría de Ciencia y Tecnología de la UNLaM |
| **Versión** | 5 |
| **Vigencia** | 03/9/2019 |

## A. Desarrollo del proyecto (adjuntar el protocolo)

**A.1**. Grado de ejecución de los objetivos inicialmente planteados, modificaciones o ampliaciones u obstáculos encontrados para su realización (desarrolle en no más de dos (2) páginas)

Se estima que el 90 por ciento de los incidentes de seguridad son el resultado de atacantes que explotan errores de software conocidos, por lo que, eliminar esos errores en la fase de desarrollo podría reducir los riesgos de seguridad que enfrentan muchas organizaciones.

Las herramientas de Prueba de Seguridad de Aplicaciones Estáticas (SAST: Static Application Security Testing), analizan el código fuente o las versiones compiladas del código tratando de encontrar vulnerabilidades o fallas de seguridad antes de que se conviertan en una versión final de software.

Las herramientas SAST se pueden agregar al Entorno de Desarrollo Integrado (IDE: Integrated Development Environment). La retroalimentación de la herramienta SAST puede ahorrar tiempo y esfuerzo, especialmente en comparación con la búsqueda de vulnerabilidades más adelante en el ciclo de desarrollo.

Las fortalezas de las herramientas SAST son:

- Escalabilidad: se puede ejecutar en una gran cantidad de lenguajes de software y se puede ejecutar repetidamente (por ejemplo: compilaciones nocturnas o integración continua).
- Identifica ciertas vulnerabilidades bien conocidas, tales como:
  - Desbordamientos de búfer
  - Defectos de inyección de SQL
- La salida ayuda a los desarrolladores, ya que las herramientas SAST resaltan el código con problemas, por nombre de archivo, ubicación, número de línea e incluso el fragmento de código afectado.

También presentan debilidades, tales como:

- Búsquedas difíciles de automatizar para muchos tipos de vulnerabilidades de seguridad, que incluyen:
  - Problemas de autenticación
  - Problemas de control de acceso
  - Uso inseguro de la criptografía
- Las herramientas SAST actuales son limitadas. Pueden identificar automáticamente solo un porcentaje relativamente pequeño de las fallas de seguridad de las aplicaciones.
- Alto número de falsos positivos.
- Con frecuencia no se pueden encontrar problemas de configuración, ya que no están representados en el código.
- Es difícil 'probar' que un problema de seguridad identificado es una vulnerabilidad real.

- Muchas herramientas SAST tienen dificultades para analizar código que no se puede compilar.
- Con frecuencia, los analistas no pueden compilar código a menos que tengan:
  - Bibliotecas correctas
  - Instrucciones de compilación
  - Todo el código requerido

Para seleccionar una herramienta SAST, se pueden seguir los siguientes criterios:

- Que contemple el lenguaje de programación utilizado.
- Capacidad para detectar vulnerabilidades, en base a:
  - El Top Ten de OWASP (Open Worldwide Application Security Project). Representa un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones web.
  - Otros criterios como:
    - OSSTMM (Open Source Security Testing Methodology Manual). Proporciona una metodología para una exhaustiva prueba de seguridad a nivel operacional.
    - Prueba de penetración, también llamada pen test o hacking ético, es una técnica de seguridad cibernética que las organizaciones utilizan para identificar, probar y resaltar vulnerabilidades a su seguridad.
- Precisión:
  - Tasas de falso positivo/falso negativo.
  - Puntaje de referencia del Benchmark OWASP. OWASP Benchmark Project es un conjunto de pruebas en Java diseñado para evaluar la precisión, la cobertura y la velocidad de las herramientas SAST. Sin la capacidad de medir estas herramientas, es difícil comprender sus fortalezas y debilidades y compararlas entre sí.
- Capacidad para comprender las bibliotecas y los marcos que necesita.
- Requisito para el código fuente compilable.
- Capacidad para ejecutarse contra binarios (en lugar de fuente).
- Disponibilidad como complemento en los IDE usado por los desarrolladores.
- Facilidad de configuración y uso.
- Capacidad para ser incluidos en herramientas de integración/implementación continua (CI/CD).
- Costo de la licencia (puede variar según el usuario, la organización, la aplicación o las líneas de código).
- Interoperabilidad de salida:
  - SARIF (Static Analysis Results Interchange Format: Formato de intercambio de resultados de análisis estático). Es un estándar que define un formato de archivo de salida. El estándar SARIF se utiliza para optimizar la manera en el que las herramientas SAST comparten sus resultados.

Luego de hacer una revisión de los principales sitios de usuarios de herramientas SAST, se encontraron que las más utilizadas por la comunidad de desarrolladores son las siguientes:

| ***Nombre*** | ***Licencia*** | ***Características*** |
|---|---|---|
| Bandit | open-source | Para aplicaciones en Python. |
| Brakeman | open-source | Para aplicaciones Ruby on Rails. |
| Checkmarx | comercial | Es compatible con todos los lenguajes. |
| CodeQL | Gratis para repositorios Open Source. Paga para repositorios privados. | Mantenida por Github. Escáner de código fuente para 7 lenguajes. |
| Contrast Scan | comercial (con Edición Comunitaria Gratuita) | Escáner de código fuente para 11 lenguajes. |
| Coverity Scan | comercial | Herramienta SAST de la suite de seguridad de aplicaciones de Synopsys. |
| Fortify Static Code Analyzer | comercial | Escáner de código fuente para 20 lenguajes. |
| HCL AppScan | comercial (AppScan CodeSweep gratis) | Escáner de código fuente para 34 lenguajes. |
| Kiuwan Code Security | comercial | Escáner de código fuente para 28 lenguajes. |
| Klocwork | comercial (con un Free Trial) | Escáner de código fuente para 7 lenguajes. |
| LGTM.COM | comercial (libre para proyectos open-source) | Escáner de código fuente para 8 lenguajes. |
| Reshift | comercial (Gratis para usuario individual) | Escáner de código estático ligero para Node.js |
| Semgrep | comercial (con Edición Comunitaria Gratuita) | Escáner de código fuente para 11 lenguajes. |
| Snyk | comercial (con edición de prueba limitada gratuita) | Escáner de código fuente para 11 lenguajes. |
| SonarQube | comercial (con edición Free Community) | Escáner de código fuente para 29 lenguajes. |
| Veracode Static Analysis | comercial | Proporciona comentarios automatizados en entornos IDE y CI/CD |

Si analizamos el uso de las herramientas SAST en las distintas etapas del ciclo de vida de desarrollo de software, podemos resumirlo de la siguiente manera:

- Requerimientos/planificación: **no**
- Diseño: **no**
- Codificación (cuando el programador está trabajando localmente): **sí**
  - Plugins de IDE: analizan el código en tiempo real o al momento de guardar y muestran los resultados en la misma IDE
  - Hooks de pre-commit: es el script que se ejecuta antes del pasaje a producción y puede interrumpir la ejecución si encuentran algún error.

- Pull request (cuando el programador lanza los cambios y abre un PR (Pull Request), corren los tests automáticos de CI/CD y hay peer-review (punto de revisión): **sí**
  - SAST como check automático de un PR: muestra solo las vulnerabilidades introducidas en este PR y no las que existían de antes. Puede bloquear el merge de un PR.
- Testing: **sí**
  - Full scan de la versión candidata, antes de salir a producción.
  - Full scan de la rama "develop", si se usa git-flow (flujo de trabajo basado en Git que brinda un mayor control y organización en el proceso de integración continua).
- Producción: **sí**
  - Se puede hacer un full scan de la versión productiva. Muestra todas las vulnerabilidades, sin importar en qué versión se introdujeron.
  - Se puede hacer un full scan de master, si se usa git-flow.

SAST se puede usar de forma informativa o bloqueante. Algunos lugares, como un plugin de la IDE, no son capaces de bloquear nada y serán meramente informativos. Pero en otros podemos elegir si bloquear o no:

- Hook de pre-commit: puede interrumpir la ejecución si encuentra algo. Esto evita que la vulnerabilidad llegue a la historia de git (particularmente útil para credenciales hardocdeadas). Se recomienda incluir un flag como reporte de falso positivo que permita ejecutar igual.
- Check en PRs: puede bloquear el merge de un PR. Debería bloquear solo por vulnerabilidades introducidas en este PR y no todas las existentes, porque arreglar una puede ser complejo y sería una buena práctica abrir nuevos PRs para arreglar cada una de las viejas. Se recomienda combinar esto con una gestión de falsos positivos que deje desbloquear casos particulares, y dejar pasar todo en caso de hotfix urgentes a producción.
- Bloqueo sunset: al detectar una vulnerabilidad en un pasaje a producción, se puede dejar pasar pero empezar un contador con tiempo para arreglarla. Si vencido el tiempo se intenta un nuevo pasaje a producción que contiene la misma vulnerabilidad, este se bloquea. Todos los pasajes a producción serán bloqueados hasta que se resuelva la vulnerabilidad. De vuelta, se recomienda combinar con gestión de falsos positivos y dejar pasar en caso de autorización especial.

SAST tiene falsos positivos, eso es por naturaleza. Se pueden tratar de reducir, generalmente a costa de más falsos negativos.

Hay un debate sobre qué es realmente un falso positivo. En el contexto de analizar una herramienta SAST, conviene definir un falso positivo como un caso donde técnicamente cumple el patrón de la regla (y por eso se marca) pero viendo el código completo podemos determinar que no es vulnerable. Es decir, definirlo solamente en base al código y no a otros factores dinámicos. Como contraejemplo, si un caso se ve vulnerable en el código pero después no es

explotable por detalles del entorno externos al código, eso es un verdadero positivo para SAST, no un falso positivo.

Normalmente una herramienta SAST no se usa de forma bloqueante justamente por los falsos positivos. Si queremos bloquear necesitamos:

- Tiempos de ejecución acotados: pensar que todo estará bloqueado hasta que terminemos y podamos confirmar que no hay vulnerabilidades. Cuánto depende del contexto (segundos para precommit, minutos para un PR).
- Reglas precisas: definir una precisión mínima aceptable según el contexto, y las reglas que no cumplan deben sacarse del bloqueo (se pueden seguir mostrando a modo informativo).
- Gestión de falsos positivos: el desarrollador debe tener una forma de avisarnos que este caso no es una vulnerabilidad, es un falso positivo, y que este reporte lo desbloquee. Luego se deben revisar estos reportes, para asegurarnos que no se colaron vulnerabilidades reales.

SARIF es un estándar para expresar resultados de una herramienta SAST. Normalmente estas herramientas queremos integrarlas en nuestro CI/CD, en algunos gestores como Code Scanning de GitHub, y queremos visualizar los resultados en algún front-end, en un plugin de nuestra IDE, etc, y para esto necesitamos una comunicación estándar.

Los resultados se expresan en un gran JSON (JavaScript Object Notation: es un formato de texto sencillo para el intercambio de datos.), que incluye qué herramientas se corrieron (puede haber más de una), qué resultados encontraron, en dónde (apuntan a una porción del código, con fila y columna de inicio y fin), metadata de la herramienta y regla que lo encontró, mensaje de alerta, texto de ayuda, camino de los datos en caso de “data Flow” (donde cada “step” apunta a una porción de código distinta), etc.

El estándar explica, para los creadores de resultados, cómo generar un SARIF (campos obligatorios, formatos, etc.), y para los lectores de resultados cómo leerlo.

Un problema interesante en SAST es el seguimiento de los resultados a lo largo de distintos escaneos. Es importante identificarlos para poder decir si este caso es nuevo, está desde antes y sigue abierto, o está arreglado porque ya no se encuentra más. También será importante en la gestión de falsos positivos.

Ese identificador único que se mantiene de un análisis a otro se llama “fingerprint”, y SARIF lo contempla. SARIF también contempla el objeto “partialFingerprints”, donde podemos pasar distintas propiedades y el sistema lector lo junta con otras cosas (como regla y herramienta) para terminar de definir el fingerprint. Los fingerprints también deben tener versiones y no se deben dejar de mandar las versiones viejas, de forma que el sistema lector “matchee” por la versión más nueva que encuentre en ambos escaneos.

Para generarlo, no se pueden usar datos como nombre de archivo o número de línea, un resultado sigue siendo el mismo por más que se renombre el archivo o se escriba código más arriba variando la línea. Si nuestra herramienta “tokeniza” el código, una estrategia es tomar

los tokens donde se marcó el resultado más algunos tokens extra hacia atrás y adelante de contexto, y "hashear" todo el resultado. Es importante evitar espacios, y en algunos también evitan tomar comentarios, para que cambios en estos no cambien el fingerprint, por ejemplo: un cambio de identación. La indentación es un tipo de notación secundaria utilizado para mejorar la legibilidad del código fuente por parte de los programadores

Algunas vulnerabilidades que se pueden encontrar con las herramientas SAST es mediante el Análisis de contaminación - seguimiento/comprobación de contaminación (Taint Analysis - Taint Tracking/Checking).

El análisis de corrupción es una técnica que se utiliza principalmente para detectar vulnerabilidades de seguridad. Realiza un seguimiento del flujo de información a través de un programa. Las entradas no confiables y los datos confidenciales son a menudo la información que se rastrea en un análisis de contaminación. Una lista de vulnerabilidades de seguridad comunes que pueden detectarse mediante análisis de corrupción es la siguiente:

- SQL Injection (cwe-89)
- OS Command Injection (cwe-78)
- XXS (cwe-79)
- Cross-Site Request Forgery (cwe-352)
- Path Traversal (cwe-22)
- Missing Authentication for Critical Function (cwe-306)
- Use of Hard-coded Credential (cwe-789)
- Missing Encryption of Sensitive Data (cwe-311)
- Open Redirect (cwe-601)

Se ha desarrollado una aplicación de software para automatizar el acceso al uso de las herramientas SAST más relevantes. La misma se ha instalado en el sitio web:

https://herramientas-sast.tfmit.online/

De manera introductoria, el interesado podrá familiarizarse con las herramientas SAST, saber de qué se tratan, para qué se utilizan y los criterios que pueden inclinar el momento de tomar una decisión por una u otra herramienta (Fig. 1).

Además de agregar la posibilidad de descarga del artículo completo desde el repositorio que Ciencia y Tecnología de la Universidad Nacional de La Matanza pone a disposición del público en general, se encontrará una breve reseña de cada una de las herramientas y para que pudieran ser utilizadas las mismas. Cabe destacar que se ha puesto foco en que el interesado pueda saber si las herramientas son gratuitas o pagas de alguna manera.

En la Fig. 2 se encuentra una breve explicación sobre qué son las herramientas SAST y en la Fig. 3 se desarrolla para que sirven.

La Fig. 4 desarrolla los factores a considerar para elegir una herramienta y a partir de las Fig. 5, 6, 7 y 8 se describen brevemente cada una de las herramientas y desde allí se puede acceder a las mismas para hacer el análisis estático de vulnerabilidades.
Las herramientas que se pueden acceder desde la aplicación son: Bandit, Brakeman, Check-marx, Codeql, Contrast Scan, Coverity Scan, Foratify Static Code Analyzer, HCL Appscan, Kiuwan Code Security, Klocwork, LGTM.com, Reshift, Semgrep, SNYK, Sonarqude y Veracoede Static Analysis.

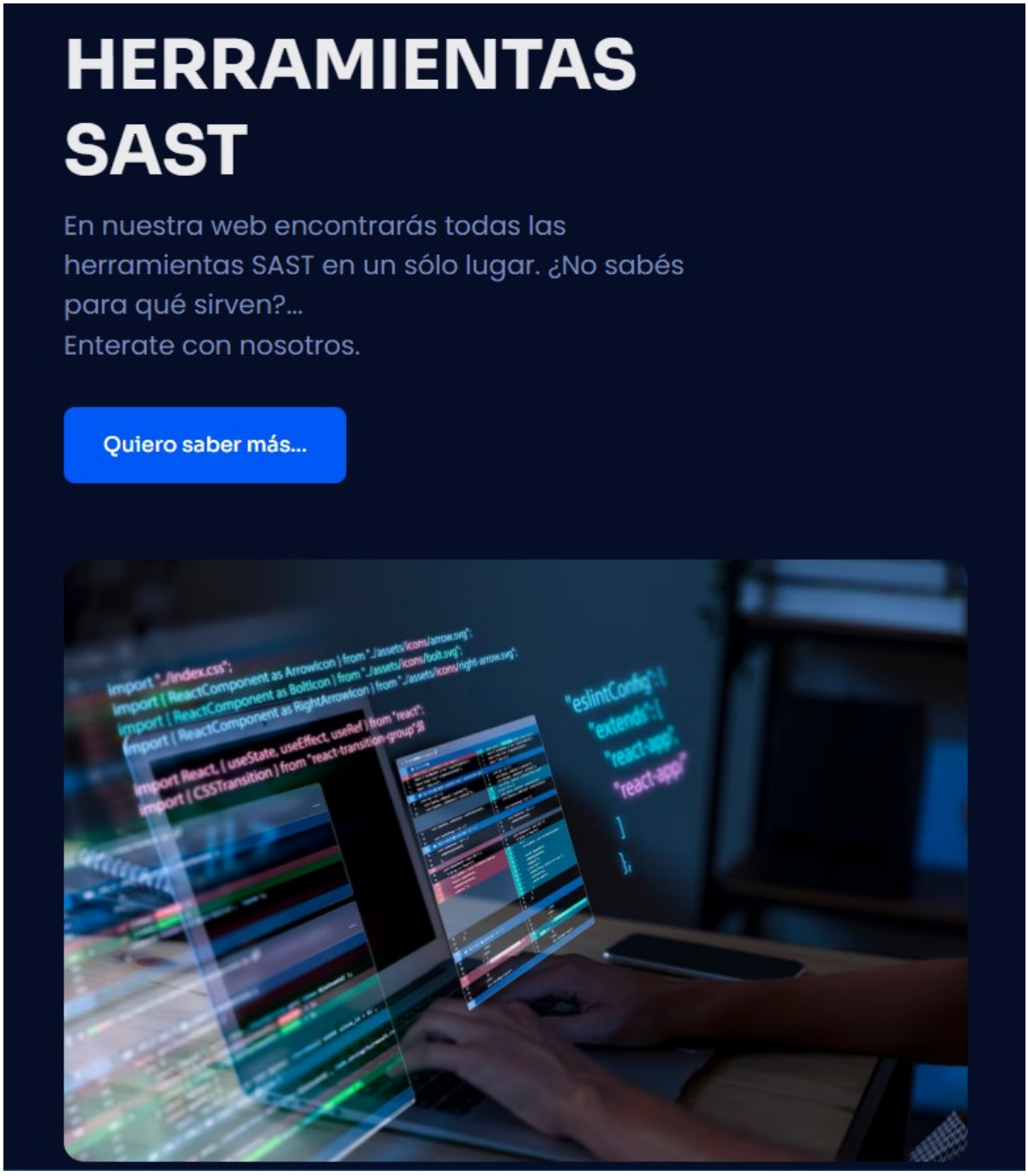

HERRAMIENTAS SAST
En nuestra web encontrarás todas las herramientas SAST en un sólo lugar. ¿No sabés para qué sirven?...
Enterate con nosotros.
Quiero saber más...

Fig. 1


Herramientas SAST
¿Qué son las SAST?
Las pruebas estáticas de seguridad de la aplicación (SAST: Static Application Security Testing) analizan el código fuente de la aplicación sin ejecutarlo, tratando así de encontrar vulnerabilidades o bugs. 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, 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.

Fig. 2


¿Para qué sirven?
Las herramientas de Pruebas Estáticas de Seguridad de las Aplicaciones, permiten automatizar la detección de vulnerabilidades y se pueden integrar al sistema de distribución de las aplicaciones (CI/CD: Integración Continua/Distribución Continua) para que detecten las vulnerabilidades en etapas tempranas del ciclo de vida, lo que llevaría a implementar un ciclo de vida seguro de desarrollo de software.

Fig. 3


Factores a considerar
Los factores a tener en cuenta para elegir una herramienta son la velocidad a la que se quiere realizar el análisis y la profundidad del mismo, ya que a más profundidad, más se tardará en realizar y viceversa. Además, se ha de considerar el porcentaje de falsos positivos que puede dar la herramienta y si contempla el lenguaje de programación de la aplicación.

Fig. 4


ALTERNATIVAS
En esta sección encontraremos diferentes alternativas con una breve descripción de cada una de ellas. Hace clic sobre el nombre y conocé más.
BANDIT
Open-Source
Es una herramienta diseñada para encontrar errores de seguridad comunes en código escrito en lenguaje Python.
BRAKEMAN
Open-Source
Se trata de un escáner de análisis de código gratuito y vulnerabilidad de seguridad para aplicaciones Ruby on Rails..
CHECKMARX
Comercial
Es una plataforma unificada que busca garantizar la seguridad de aplicaciones empresariales.
CODEQL
Open-Source repo público - Comercial repo privado
Es un motor de análisis de código semántico. Busca vulnerabilidades y permite compartirlas.

Fig. 5

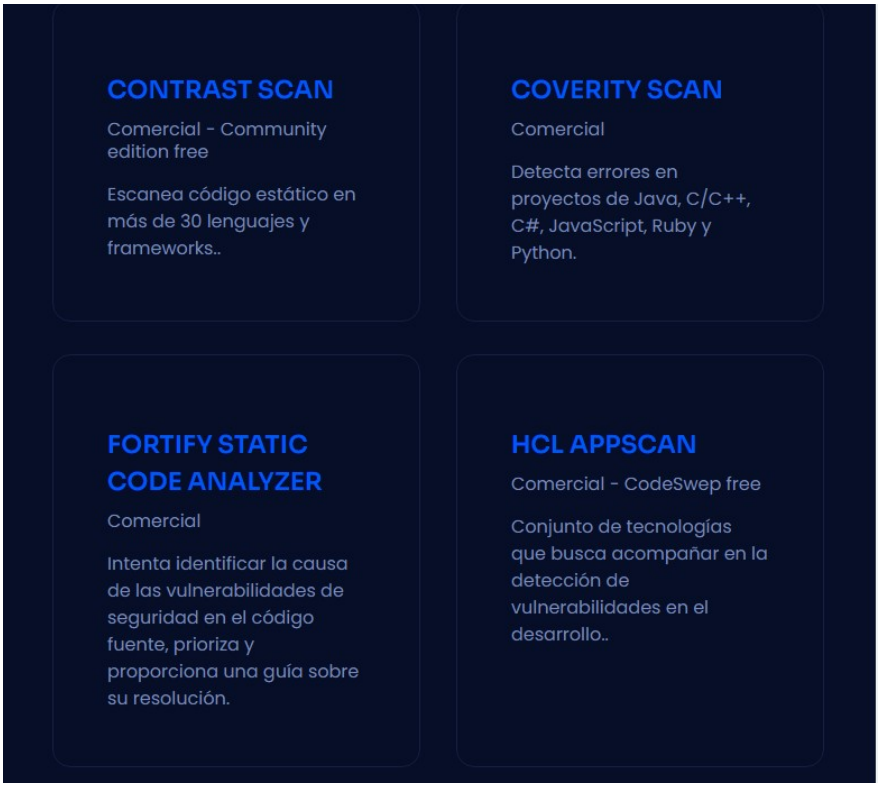

CONTRAST SCAN
Comercial - Community edition free
Escanea código estático en más de 30 lenguajes y frameworks..
COVERITY SCAN
Comercial
Detecta errores en proyectos de Java, C/C++, C#, JavaScript, Ruby y Python.
FORTIFY STATIC CODE ANALYZER
Comercial
Intenta identificar la causa de las vulnerabilidades de seguridad en el código fuente, prioriza y proporciona una guía sobre su resolución.
HCL APPSCAN
Comercial - CodeSwep free
Conjunto de tecnologías que busca acompañar en la detección de vulnerabilidades en el desarrollo..

Fig. 6


KIUWAN CODE SECURITY
Comercial
Se define cómo una plataforma integral de seguridad de aplicaciones, mezclando varios tipos de herramientas.
KLOCWORK
Comercial - Free Trial
Destinado a encontrar vulnerabilidades en C, C++, C#, Java, JS, Python y Kotlin.
LGTM.COM
Comercial - Free open source projects
Integrado a Github, se trata de una herramienta similar a CodeQl.
RESHIFT
Comercial - Free para usarlo individual
Provee gran velocidad, automatización y análisis de soluciones basándose en vulnerabilidades de componentes.

Fig. 7


Fig. 8

Finalmente se muestra en la Fig. 9 hay un enlace a un artículo de investigación titulado "*Características de las Herramientas de Pruebas Estáticas de Seguridad de las Aplicaciones*" que desarrolla en detalle este tema publicado en la ReDDI – Revista Digital del Departamento de Ingeniería del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza.


Fig. 9

## B. Principales resultados de la investigación

B.1. Publicaciones en revistas (informar cada producción por separado)

| Artículo 1: | |
|---|---|
| Autores | *Jorge Eterovic*<br>*Valeria Silvestri*<br>*Martín Zeballos*<br>*Andrea Vera*<br>*Alesio Sinopoli* |
| Título del artículo | *Características de las Herramientas de Pruebas Estáticas de Seguridad de las Aplicaciones* |
| N° de fascículo | *2* |
| N° de Volumen | *7* |
| Revista | *ReDDI – Revista Digital del Departamento de Ingeniería* |
| Año | *2022* |
| Institución editora de la revista | *Universidad Nacional de La Matanza* |
| País de procedencia de institución editora | *Argentina* |
| Arbitraje | SI |
| ISSN: | *2525-1333* |
| URL de descarga del artículo | *En prensa* |
| N° DOI | |

B.2. Libros

| Libro 1 | |
|---|---|
| Autores | |
| Título del Libro | |
| Año | |
| Editorial | |
| Lugar de impresión | |
| Arbitraje | Elija un elemento. |
| ISBN: | |
| URL de descarga del libro | |
| N° DOI | |

B.3. Capítulos de libros

| | |
|---|---|
| Autores | |

| | |
|---|---|
| Título del Capitulo | |
| Título del Libro | |
| Año | |
| Editores del libro/Compiladores | |
| Lugar de impresión | |
| Arbitraje | Elija un elemento. |
| ISBN: | |
| URL de descarga del capítulo | |
| N° DOI | |

B.4. Trabajos presentados a congresos y/o seminarios:

| | |
|---|---|
| Autores | *Jorge Eterovic; Valeria Silvestri; Andrea Vera; Martin Zeballos; Alesio Sinópoli* |
| Título | *Análisis de las Herramientas para Realizar Pruebas Estáticas de Seguridad de las Aplicaciones* |
| Año | *2023* |
| Evento | *XXV Workshop de Investigadores en Ciencias de la Computación* |
| Lugar de realización | *Junín, Pcia. de Bs. As.* |
| Fecha de presentación de la ponencia | *Abril 2023* |
| Entidad que organiza | *UNNOBA y RedUNCI* |
| URL de descarga del trabajo (especificar solo si es la descarga del trabajo; formatos pdf, e-pub, etc.) | *https://wicc2023.unnoba.edu.ar/libro-de-actas/* |

B.5. Otras publicaciones

| | |
|---|---|
| Autores | |
| Año | |
| Título | |
| Medio de Publicación | |

**C. Otros resultados. Indicar aquellos resultados pasibles de ser protegidos a través de instrumentos de propiedad intelectual, como patentes, derechos de autor, derechos de obtentor, etc. y desarrollos que no pueden ser protegidos por instrumentos de propiedad intelectual, como las tecnologías organizacionales y otros. Complete un cuadro por cada uno de estos dos tipos de productos.**

C.1. Títulos de propiedad intelectual. Indicar: Tipo (marcas, patentes, modelos y diseños, la transferencia tecnológica) de desarrollo o producto, Titular, Fecha de solicitud, Fecha de otorgamiento

| Tipo | Titular | Fecha de Solicitud | Fecha de Emisión |
|---|---|---|---|
| | | | |

C.2. Otros desarrollos no pasibles de ser protegidos por títulos de propiedad intelectual. Indicar: Producto y Descripción.

| Producto | Descripción |
|---|---|
| Aplicación de Software | Aplicación que permite el acceso directo a distintas herramientas SAST para realizar el análisis estático de vulnerabilidades de software. |

**D. Formación de recursos humanos. Trabajos finales de graduación, tesis de grado y posgrado. Completar un cuadro por cada uno de los trabajos generados en el marco del proyecto.**

D.1. Tesis de grado

| Director (apellido y nombre) | Autor (apellido y nombre) | Institución | Calificación | Fecha /En curso | Título de la tesis |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

D.2 Trabajo Final de Especialización

| Director (apellido y nombre) | Autor (apellido y nombre) | Institución | Calificación | Fecha /En curso | Título del Trabajo Final |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

D.2. Tesis de posgrado: Maestría

| Director (apellido y nombre) | Tesista (apellido y nombre) | Institución | Calificación | Fecha /En curso | Título de la tesis |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |

D.3. Tesis de posgrado: Doctorado

| Director (apellido y | Tesista (apellido y nombre) | Institución | Calificación | Fecha /En curso | Título de la tesis |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| nombre) | | | | | |
| | | | | | |
| | | | | | |

D.4. Trabajos de Posdoctorado

| Director (apellido y nombre) | Posdoctorado (apellido y nombre) | Institución | Calificación | Fecha /En curso | Título del trabajo | Publicación |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |

**E. Otros recursos humanos en formación: estudiantes/ investigadores (grado/posgrado/ posdoctorado)**

| Apellido y nombre del Recurso Humano | Tipo | Institución | Período (desde/ hasta) | Actividad asignada[1] |
|---|---|---|---|---|
| Alesio Esteban Sinopoli | Estudiante de grado | DIIT-UNLaM | 01/01/2022 al 31/12/2023 | Búsqueda y análisis de información |
| | | | | |

**F. Vinculación[2]:** Indicar conformación de redes, intercambio científico, etc. con otros grupos de investigación; con el ámbito productivo o con entidades públicas. Desarrolle en no más de dos (2) páginas.

**G. Otra información. Incluir toda otra información que se considere pertinente.**

**NO SE HAN ENVIADO TRABAJOS A CONGRESOS DURANTE EL AÑO 2022 PORQUE SE ACREDITARON LOS FONDOS DE LA 1RA. CUOTA A MEDIADOS DE OCTUBRE DE ESE AÑO, CUANDO YA HABÍAN CERRADO TODAS LOS LLAMADOS A PRESENTACIÓN DE TRABAJOS.**

**H. Cuerpo de anexos:**

- Anexo I: Copia de cada uno de los trabajos mencionados en los puntos B, C y D, y certificaciones cuando corresponda.[3]
- Anexo II:
    - FPI-013: Evaluación de alumnos integrantes. (si corresponde)

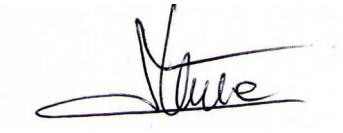
[1] Descripción de la/s actividad/es a cargo (máximo 30 palabras)

[2] Entendemos por acciones de "vinculación" aquellas que tienen por objetivo dar respuesta a problemas, generando la creación de productos o servicios innovadores y confeccionados "a medida" de sus contrapartes.

[3] En caso de libros, podrá presentarse una fotocopia de la primera hoja significativa o su equivalente y el índice.

- FPI-014: Comprobante de liquidación y rendición de viáticos. (si corresponde)
- FPI-015: Rendición de gastos del proyecto de investigación acompañado de las hojas foliadas con los comprobantes de gastos.
- FPI-035: Formulario de reasignación de fondos en Presupuesto.

- Nota justificando baja de integrantes del equipo de investigación.



Mag. Jorge Eterovic

San Justo, 28 de febrero de 2023

- Cargar este formulario junto con los documentos correspondientes **exclusivamente** al Anexo I en SIGEVA UNLaM. Realizar la presentación impresa de los mismos junto con los restantes Anexos en la Secretaría de Investigación de la unidad académica correspondiente**. Límite de entrega: 28 de febrero de 2020.**

# Anexo I

Revista Digital del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza

ISSN: 2525-1333
Vol.: 7 - Nro. 2 (Diciembre-2022)


*artículo original*

# CARACTERÍSTICAS DE LAS HERRAMIENTAS DE PRUEBAS ESTÁTICAS DE SEGURIDAD DE LAS APLICACIONES

# CHARACTERISTICS OF STATIC APPLICATION SECURITY TESTING TOOLS


*Jorge ETEROVIC[1][2], Valeria SILVESTARI[1], Martin ZEBALLOS[1], Andrea VERA[1], Alesio SINOPOLI[1],*

[1] Universidad Nacional de La Matanza
Departamento de Ingeniería e Investigaciones Tecnológicas
San Justo, Pcia. de Buenos Aires, Argentina
[2] eterovic@unlam.edu.ar



**Resumen:**

Las herramientas de Pruebas Estáticas de Seguridad de las Aplicaciones (SAST: Static Application Security Testing), permiten automatizar la detección de vulnerabilidades y se pueden integrar al sistema de distribución de las aplicaciones (CI/CD: Integración Continua/Distribución Continua) para que detecten las vulnerabilidades en etapas tempranas del ciclo de vida, lo que llevaría a implementar un ciclo de vida seguro de desarrollo de software.

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. En este

ISSN: 2525-1333 – Vol. 7 - Nro. 2 (Diciembre -2022)



trabajo se analizan las fortalezas, las debilidades, las características a considerar para la selección de una herramienta y se muestra un listado de las mas usadas, ya sean open-source o pagas.

**Abstract:**

Static security test tools for applications (SAST: Static Application Security Testing), allow automating vulnerabilities detection and can be integrated into the applications distribution system (CI/CD: Continuous integration/continuous distribution) to detect Vulnerabilities in early stages of the life cycle, which would lead to implementing a safe software development life cycle.

Integrating a SAST tool into the CI/CD process allows you to detect vulnerabilities in the development stage, instead of waiting for the test stage or to be detected directly in production. In this work the strengths, weaknesses, characteristics to consider for the selection of a tool are analyzed and a list of the most used ones is shown, whether they are open-source or paid.



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

**Key Words:** SAST tool; Vulnerability Detection; Static Analysis; Application Security.


## I. INTRODUCCIÓN

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.

Revista Digital del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza
ISSN: 2525-1333 – Vol. 7 - Nro. 2 (Diciembre -2022)


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].

Esto tiene grandes ventajas, pues permite tener feedback temprano del cliente para mejorar el producto desde las primeras fases de 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].

DevSecOps surge para incluir la seguridad en DevOps, alineando los equipos de desarrollo, de operaciones y de seguridad durante todo el ciclo de vida de desarrollo. Esto se consigue haciendo participar al equipo de seguridad 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].

Al igual que en DevOps, las herramientas juegan un papel muy importante. Existen una gran cantidad de herramientas para llevar a cabo DevSecOps. Una referencia relevante es la guía OWASP DevSecOps Guideline [11] para establecer los aspectos más importantes relativos a la seguridad: la detección de vulnerabilidades, 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.

## II. 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, 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 para elegir una herramienta son la velocidad a la que se quiere realizar el análisis y la profundidad del mismo, ya que a más profundidad, más se tardará en realizar y viceversa. Además, se ha de considerar el porcentaje de falsos positivos que puede dar la herramienta y si contempla el lenguaje de programación de la aplicación.

Revista Digital del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza
ISSN: 2525-1333 – Vol. 7 - Nro. 2 (Diciembre -2022)


## III. CARACTERÍTICS DE LAS HERRAMIENTAS SAST

Se estima que el 90 por ciento de los incidentes de seguridad son el resultado de ataques que explotan errores de software conocidos, por lo que, eliminar esos errores en la fase de desarrollo podría reducir los riesgos de seguridad.

Las herramientas de Pruebas Estáticas de Seguridad de Aplicaciones analizan el código fuente o las versiones compiladas del código tratando de encontrar vulnerabilidades o fallas de seguridad antes de que se conviertan en una versión final de software.

Las herramientas SAST se pueden agregar al Entorno de Desarrollo Integrado (IDE: Integrated Development Environment). La retroalimentación de la herramienta SAST puede ahorrar tiempo y esfuerzo, especialmente en comparación con la búsqueda de vulnerabilidades en fases avanzadas del ciclo de vida del desarrollo de software.

Las fortalezas de las herramientas SAST son:

- Escalabilidad: se puede ejecutar en una gran cantidad de lenguajes de software y se puede ejecutar repetidamente (por ejemplo: compilaciones nocturnas o integración continua).
- Identifica ciertas vulnerabilidades bien conocidas, tales como:
  - Desbordamientos de búfer
  - Defectos de inyección de SQL
- La salida ayuda a los desarrolladores, ya que las herramientas SAST resaltan el código con problemas, por nombre de archivo, ubicación, número de línea e incluso el fragmento de código comprometido.

También presentan debilidades, tales como:

- Búsquedas difíciles de automatizar para muchos tipos de vulnerabilidades de seguridad, que incluyen:
  - Problemas de autenticación
  - Problemas de control de acceso
  - Uso inseguro de la criptografía
- Las herramientas SAST actuales son limitadas. Pueden identificar automáticamente solo un porcentaje relativamente pequeño de las fallas de seguridad de las aplicaciones.
- Existe un alto número de falsos positivos.
- Con frecuencia no se pueden encontrar problemas de configuración, ya que no están representados en el código.
- Es difícil 'probar' que un problema de seguridad identificado es una vulnerabilidad real.
- Muchas herramientas SAST tienen dificultades para analizar código que no se puede compilar.
- Con frecuencia, los analistas no pueden compilar código a menos que tengan:
  - Bibliotecas correctas
  - Instrucciones de compilación
  - Todo el código requerido

Algunas de las herramientas SAST mas usadas son las siguientes:

| *Nombre* | *Tipo de Licencia* |
|---|---|
| Bandit | open-source |
| Brakeman | open-source |
| Checkmarx | comercial |

Revista Digital del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza
ISSN: 2525-1333 – Vol. 7 - Nro. 2 (Diciembre -2022)


| | |
|---|---|
| CodeQL | Gratis para repositorios Open Source. Paga para repositorios privados. |
| Contrast Scan | comercial (con Edición Comunitaria Gratuita) |
| Coverity Scan | comercial |
| Fortify Static Code Analyzer | comercial |
| HCL AppScan | comercial (AppScan CodeSweep gratis) |
| Kiuwan Code Security | comercial |
| Klocwork | comercial (con un Free Trial) |
| LGTM.COM | comercial (libre para proyectos open-source) |
| Reshift | comercial (Gratis para usuario individual) |
| Semgrep | comercial (con Edición Comunitaria Gratuita) |
| Snyk | comercial (con edición de prueba limitada gratuita) |
| SonarQube | comercial (con edición Free Community) |
| Veracode Static Analysis | comercial |

*Fuente: recopilación propia*

## IV. CONCLUSIONES

Para seleccionar una herramienta SAST, se pueden seguir los siguientes criterios:

- Que contemple el lenguaje de programación utilizado.
- Que tenga capacidad para detectar vulnerabilidades, en base a:
  - El Top Ten de OWASP (Open Worldwide Application Security Project). Representa un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones Web.
  - Otros criterios como:
    - OSSTMM (Open-Source Security Testing Methodology Manual). Proporciona una metodología para una exhaustiva prueba de seguridad a nivel operacional.
    - Prueba de penetración, también llamada pen test o hacking ético, es una técnica de seguridad cibernética que las organizaciones utilizan para identificar, probar y resaltar vulnerabilidades a su seguridad.
- Precisión:
  - Tasas de falso positivo/falso negativo.
  - Puntaje de referencia del Benchmark OWASP. OWASP Benchmark Project es un conjunto de pruebas en Java diseñado para evaluar la precisión, la cobertura y la velocidad de las herramientas SAST. Sin la capacidad de medir estas herramientas, es difícil comprender sus fortalezas y debilidades y compararlas entre sí.
- Capacidad para comprender las bibliotecas y los marcos que necesita.
- Requisito para el código fuente compilable.
- Capacidad para ejecutarse contra binarios (en lugar de fuente).
- Disponibilidad como complemento en los IDE usado por los desarrolladores.

Revista Digital del Departamento de Ingeniería e Investigaciones Tecnológicas de la Universidad Nacional de La Matanza
ISSN: 2525-1333 – Vol. 7 - Nro. 2 (Diciembre -2022)


- Facilidad de configuración y uso.
- Capacidad para ser incluidos en herramientas de integración/implementación continua (CI/CD).
- Costo de la licencia (puede variar según el usuario, la organización, la aplicación o las líneas de código).
- Interoperabilidad de salida:
  - SARIF (Static Analysis Results Interchange Format: Formato de Intercambio de Resultados de Análisis Estático). Es un estándar que define un formato de archivo de salida. El estándar SARIF se utiliza para optimizar la manera en el que las herramientas SAST comparten sus resultados.

## V. REFERENCIAS


[1] de Franca, B. B. N., Jerónimo, 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 Scientific 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/

# 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

UNIVERSIDAD NACIONAL NOROESTE BUENOS AIRES
ESCUELA DE TECNOLOGÍA

Se certifica que Eterovic Jorge E., ha participado en calidad de Asistente en el XXV Workshop de Investigadores en Ciencias de la Computación (WICC 2023), realizado los días 13 y 14 de Abril de 2023 en la Ciudad de Junín, Buenos Aires, Argentina.

Lic. Patricia Pesado
Coordinadora Titular
Red Unci

Lic. Monica Sarobe
Directora de la Escuela de Tecnología
UNNOBA

# Anexo II

UNLaM - SECyT

FPI-013

FORMULARIO DE EVALUACIÓN DE ALUMNO S INTEGRANTE S DE EQUIPO S DE INVE STIGACIÓN

Unidad Académica: Departamento de Ingeniería e Investigaciones Tecnológicas
Código: C2-ING-094
Título del Proyecto: Análisis de Herramientas SAST
Director del Proyecto: Mag. Ing. Jorge Eterovic
Programa de acreditación: CyTMA2
Fecha de inicio: 1/1/2022.
Fecha de finalización: 31/12/2023.

**1. Datos del alumno**
Apellido y Nombre: Sinopoli Alesio, Esteban Abel
DNI: 40.808.298
Unidad Académica: Departamento de Ingeniería e Investigaciones Tecnológicas
Carrera que cursa: Ing. informática
Período evaluado: 2022 - 2023

**2. Dictamen de evaluación de desempeño del alumno:**
*Colocar una cruz donde corresponda*

2.1 Satisfactorio: X
2.1 No satisfactorio:

Fundamentos del dictamen:
El alumno participó en la búsqueda bibliográfica de artículos y publicaciones sobre el tema de investigación e hizo un análisis de la misma que sirvió como fuente de información para escribir un artículo en la revista ReDDI. También colaboró en el diseño de la Aplicación de Software para el acceso automatizado a las herramientas SAST

**3. Propuesta de continuidad en el proyecto (si corresponde según duración estimada)**
*Colocar una cruz donde corresponda*

3.1 Continuar en el presente proyecto:
3.2 No continuar en el presente proyecto: X

Fundamentos del dictamen:
Su participación en el proyecto ha concluido satisfactoriamente, alcanzando los resultados esperados.
No estaba prevista otra actividad en el plan de trabajo del proyecto.

| | | |
|---|---|---|
| San Justo, 28/02/2024 |  | Mag. Jorge Eterovic |
| Lugar y fecha | Firma del Director | Aclaración de firma |