Análisis del impacto de las vulnerabilidades en las dependencias de proyectos software basado en Satisfiability Modulo Theories (SMT)

Antonio Germán Márquez Trujillo, Ángel Jesús Varela-Vaca,
María Teresa Gómez López, José A. Galindo,
y David Benavides
IDEA Research Group, Universidad de Sevilla
{amtrujillo, ajvarela, maytegomez, jagalindo, benavides}@us.es

***Resumen*—Actualmente el software desarrollado adquiere relaciones dependientes con librerías externas, lo que promueve el aumento del uso de dependencias como una práctica común. Cualquier vulnerabilidad que afecte a estas dependencias puede poner en riesgo todo el proyecto, lo que complica el mantenimiento de la seguridad en los proyectos de desarrollo de software. Actualmente, las herramientas disponibles no abarcan todas las posibles configuraciones de dependencias, lo que amplía el desafío. En este trabajo, se propone un enfoque que permite analizar el espacio de configuración de dependencias de proyectos en términos de las vulnerabilidades. Nuestra propuesta se materializa mediante la construcción de un grafo de dependencias atribuido con vulnerabilidades. Para habilitar el análisis automático, integramos modelos formales basados en modelos de satisfacibilidad. Esto permite el análisis automático para determinar configuraciones libres de vulnerabilidades. Por último, se compara nuestra propuesta en repositorios Python, obteniendo resultados superiores a otras propuestas.**

***Index Terms*—Seguridad, Vulnerabilidad, Análisis Automático, Satisfiability Modulo Theories (SMT), Grafo de Dependencias, Desarrollo del software**

Tipo de contribución: *Investigación ya publicada [1]*

I. Introducción

Hoy en día, los proyectos de desarrollo de software dependen de bibliotecas o herramientas de terceros para diversas tareas. Estas bibliotecas, generan una cadena de dependencias entre los componentes de software, resultando en proyectos con un gran número de dependencias que necesitan gestión [2]. Las vulnerabilidades en las dependencias pueden explotarse para atacar un sistema basado en un defecto de software. El elevado número de configuraciones y dependencias dificulta a los desarrolladores determinar qué vulnerabilidades afectan al software y cómo mitigar los riesgos de seguridad.

Presentamos Depex, una solución que automatiza el análisis sobre el espacio de configuración de dependencias, teniendo en cuenta sus vulnerabilidades. Nuestras contribuciones incluyen un proceso de extracción de dependencias, un proceso de atribución de vulnerabilidades, y un proceso de transformación del grafo de dependencias a un modelo de satisfacibilidad. Esto habilita técnicas de análisis automático mediante operaciones de razonamiento sobre el espacio de configuraciones. Evaluamos nuestra propuesta con una comparación utilizando una lista de repositorios Python, donde nuestra propuesta demuestra que detecta más vulnerabilidades.

II. Trabajos Relacionados

Existen diferentes enfoques para la identificación de vulnerabilidades en las dependencias, incluyendo la coincidencia de nombres, términos y valores con los de las bases de datos de vulnerabilidades [3]. Sólo unas pocas propuestas utilizan grafos de dependencias, y una de ellas se centra en la manipulación de grafos en lugar de en la transformación a modelos formales para el razonamiento automático. Otros enfoques exploran aspectos semánticos [4], análisis de código [5], acoplamiento de dependencias en el código fuente [6], y la aplicación del aprendizaje profundo para mejorar la seguridad del código [7].

La naturaleza del proyecto de software también es significativa, ya que la mayoría de los estudios se centran en proyectos Java que utilizan Maven [8], mientras que otros analizan proyectos JavaScript con NPM [9], o tanto Maven como NPM [10]. Nuestra propuesta destaca como única por su cobertura de proyectos basados en Python y dependencias en repositorios como PyPI.

III. Propuesta

Depex se compone de tres procesos distintos para llevar a cabo el desafío del análisis de vulnerabilidades en las dependencias de proyectos de desarrollo software:

1. Extracción de grafos de dependencias. Los ficheros de requisitos de un proyecto se extraen de un repositorio, y a continuación, se analizan sus dependencias directas. Luego, se examinan las dependencias indirectas empleando información del gestor de paquetes en un proceso recursivo.

2. Atribución de grafos de dependencias. Durante el análisis de dependencias, se asigna a cada dependencia del grafo creado en la base de datos Depex información sobre la vulnerabilidad, la cual se extrae de la base de datos de vulnerabilidades NVD del NIST.

3. Razonamiento automático. La información se genera y codifica a partir de la base de datos de Depex en un modelo formal, utilizando un resolutor basado en Satisfiability Modulo Theories (SMT) [11], lo cual nos permite realizar un análisis exhaustivo de las dependencias y sus vulnerabilidades. Mediante este enfoque, aplicamos una serie de operaciones que facilitan el razonamiento sobre la seguridad de las dependencias.

ISBN:978-84-09-62140-8

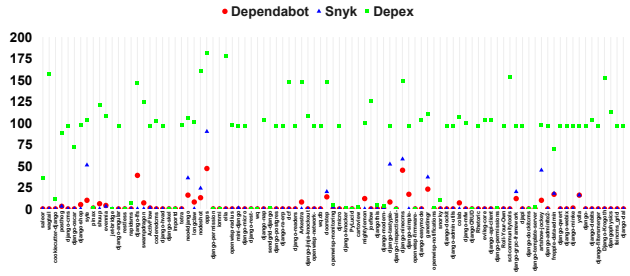

Figura 1. Vulnerabilidades directas detectadas por cada herramienta

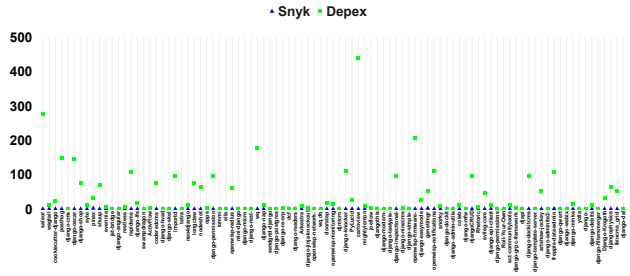

Figura 2. Vulnerabilidades indirectas detectadas por cada herramienta

IV. Experimentación

En la experimentación se compara el rendimiento de Depex con herramientas existentes en el mercado, como Snyk y Dependabot de GitHub, populares y compatibles con proyectos Python. El objetivo del experimento es determinar si Depex puede detectar más vulnerabilidades en las dependencias de un proyecto. Se busca proporcionar información más detallada sobre las vulnerabilidades del proyecto en comparación con otras opciones, lo que beneficiaría a los desarrolladores al ofrecerles una visión más completa de las posibles vulnerabilidades que podrían afectar a sus proyectos. Para minimizar sesgos en la comparación, se distinguen entre vulnerabilidades directas e indirectas, mientras que Dependabot solo analiza las dependencias directas.

En las Figuras 1 y 2, se presentan los resultados del análisis de los repositorios, donde se observa la detección de vulnerabilidades directas e indirectas por parte de diferentes herramientas. Para la mayoría de los proyectos, Snyk identifica más vulnerabilidades directas que Dependabot, pero Depex supera a Snyk en la detección de vulnerabilidades directas en todos los proyectos. Sin embargo, en el caso del proyecto *pinax*, todas las herramientas detectan las mismas vulnerabilidades directas debido a que el fichero de requisitos fija las versiones de las dependencias, eliminando así la variabilidad de versiones y la capacidad de cubrir todo el rango de versiones, lo que otras herramientas no pueden hacer. En cuanto a las vulnerabilidades indirectas, solo se consideraron Depex y Snyk, ya que Dependabot no ofrece un análisis de dependencias indirectas. En este aspecto, Depex detecta más vulnerabilidades indirectas que Snyk en todos los casos.

V. Conclusiones y Trabajos Futuros

Depex aborda el desafío del análisis de vulnerabilidades en proyectos de software mediante la construcción de un grafo de dependencias con información sobre vulnerabilidades y un modelo formal basado en SMT para analizar configuraciones. Sin embargo, presenta limitaciones que requieren atención futura, 1) Dependencia exclusiva de la base de datos NVD, lo que podría generar inconsistencias en las vulnerabilidades. Se sugiere incorporar múltiples bases de datos para mejorar la coherencia; 2) Generación lenta de grafos, especialmente en proyectos asociados a NPM, debido a la gran cantidad de información. Se planea acelerar este proceso y permitir la precarga de grafos para mejorar la velocidad de operación; y, 3) Disponibilidad limitada de métricas, con solo la media y la media ponderada actualmente ofrecidas. Se sugiere implementar diversas métricas para extraer más información sobre la seguridad de las dependencias.

Agradecimientos

Esta publicación es parte de los proyectos COPERNICA (P20_01224) y AETHER-US PID2020-112540RB-C44 y METAMORFOSIS (US-1381375) y ALBA-US TED2021-130355B-C32 y Data-pl (PID2022-138486OB-I00) y ALBA-US TED2021-130355B-C32 y TASOVA PLUS research network (RED2022-134337-T) financiado por MICIU/AEI/10.13039/501100011033 y "European Union NextGenerationEU/PRTR".

Referencias

[1] A. Germán Márquez, Ángel Jesús Varela-Vaca, M. T. Gómez López, J. A. Galindo, and D. Benavides, "Vulnerability impact analysis in software project dependencies based on satisfiability modulo theories (SMT)," *Computers & Security*, vol. 139, p. 103669, 2024.

[2] J. A. Galindo, D. Benavides, and S. Segura, "Debian packages repositories as software product line models. towards automated analysis," in *ACOTA, Belgium, September, 2010*, vol. 688. CEUR-WS.org, 2010, pp. 29–34.

[3] M. Cadariu, E. Bouwers, J. Visser, and A. van Deursen, "Tracking known security vulnerabilities in proprietary software systems," in *2015 IEEE 22nd SANER*, 2015, pp. 516–519.

[4] S. S. Alqahtani, E. E. Eghan, and J. Rilling, "Tracing known security vulnerabilities in software repositories – a semantic web enabled modeling approach," *Science of Computer Programming*, vol. 121, pp. 153–175, 2016.

[5] S. Ponta, H. Plate, and A. Sabetta, "Beyond metadata: Code-centric and usage-based analysis of known vulnerabilities in open-source software," in *2018 ICSME*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2018, pp. 449–460.

[6] N. B. Tàrrega, M. Živković, and A. Oprescu, "Measuring the impact of library dependency on maintenance," vol. 2754, 2020, Conference paper, cited by: 0.

[7] X. Yuan, G. Lin, Y. Tai, and J. Zhang, "Deep neural embedding for software vulnerability discovery: Comparison and optimization," *Security and Communication Networks*, vol. 2022, 2022, cited by: 1; All Open Access, Gold Open Access.

[8] I. Pashchenko, H. Plate, S. E. Ponta, A. Sabetta, and F. Massacci, "Vulnerable open source dependencies: Counting those that matter," in *12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2018, pp. 1–10.

[9] J. Hejderup, "In dependencies we trust: How vulnerable are dependencies in software modules?" 2015.

[10] Q. Li, J. Song, D. Tan, H. Wang, and J. Liu, "Pdgraph: A large-scale empirical study on project dependency of security vulnerabilities," 2021, Conference paper, p. 161 – 173, cited by: 3.

[11] P. Arcaini, A. Gargantini, and P. Vavassori, "Generating tests for detecting faults in feature models," in *2015 IEEE 8th ICST*. IEEE, 2015, pp. 1–10.