Método de identificación y minimización de superficie de ataque utilizando antipatrón de vulnerabilidades *

Miguel Solinas[1 [0000−0002−7550−1067]], Gina Commisso[1 [0009−0000−2848−2927]], and Leandro Antonelli[2 [0000−0003−1388−0337]]

[1] LARYC, Facultad de Ciencias Exactas Físicas y Naturales, Universidad Nacional de Córdoba, AR {miguel.solinas, ginacommisso}@unc.edu.ar
[2] Lifia - Fac de Informatica, UNLP, La Plata, AR; CAETI - Facultad de Tecnología Informática - Universidad Abierta Interamericana, Buenos Aires, AR {leandro.antonelli@lifia.info.unlp.edu.ar}

Abstract. Sea un software mínimamente vulnerable aquel que no reitera vulnerabilidades conocidas. Evitar reiterar vulnerabilidades conocidas en el desarrollo de software es un excelente punto de partida para disponer de un ciberespacio más seguro. El número de vulnerabilidades descubiertas y solucionadas expresa la superficie de ataque digital de un sistema. Pero el comportamiento dinámico temporal del registro de vulnerabilidades que muestran ciertos Content Management Systems (CMS) no generan la confianza asociada a la solución de las vulnerabilidades descubiertas. En este trabajo proponemos un método para identificar y minimizar la reiteración de vulnerabilidades conocidas en un proyecto de software, utilizando el antipatrón de vulnerabilidades y partiendo de los componentes de la arquitectura del CMS utilizado.

Keywords: Software Security · Vulnerability · Security Pattern.

1 Introducción

1.1 Motivación

Las vulnerabilidades cuantifican la superficie de ataque digital de un sistema. En particular el stack de componentes expuestos a una red pública o dentro de una red privada compleja incluyen en su versión mínima: firmware, sistema operativo, frameworks varios y múltiples aplicaciones con sus propias arquitecturas. Todos ellos tienen vulnerabilidades conocidas y registradas en bases de datos como CVE [1]. Dichas vulnerabilidades constituyen una información útil en la construcción de nuevas aplicaciones de software en la medida en que no se reiteren. Sentido común que no tiene su correlato en la realidad y las vulnerabilidades conocidas se reiteran en nuevos sistemas todo el tiempo.

¿Cómo es posible que esto ocurra? Si bien los modelos de proceso de construcción de software tienen una madurez que aportan técnicas para minimizar el

* Supported by Secretaría de Ciencia y Tecnología de la UNC.

problema, por ejemplo la construcción de modelos de amenazas [7]; es frecuente que las herramientas utilizadas y urgencias de entregar funcionalidad conduzcan el desarrollo en la dirección de prácticas que repiten vulnerabilidades conocidas. Mas la falta de formación formal en aspectos de ciberseguridad conducen a una escasa conciencia de las consecuencias de repetir fallas conocidas. Cuando se construye software utilizando Content Management Systems (CMS) los 10 principales frameworks representan más del 86% del market share, con un 65,1% de WordPress [2]. Cuando se utiliza este framework, existe una exposición a la historia de sus vulnerabilidades. En la Figura 1 se muestra la evolución de la superficie de ataque de los principales CMS del mercado, con año en abcisas y vulnerabilidades descubiertas en ordenadas. Una curva llama la atención, la de WordPress. Sumado a su popularidad no es de extrañar que conduce a usuarios expuestos.

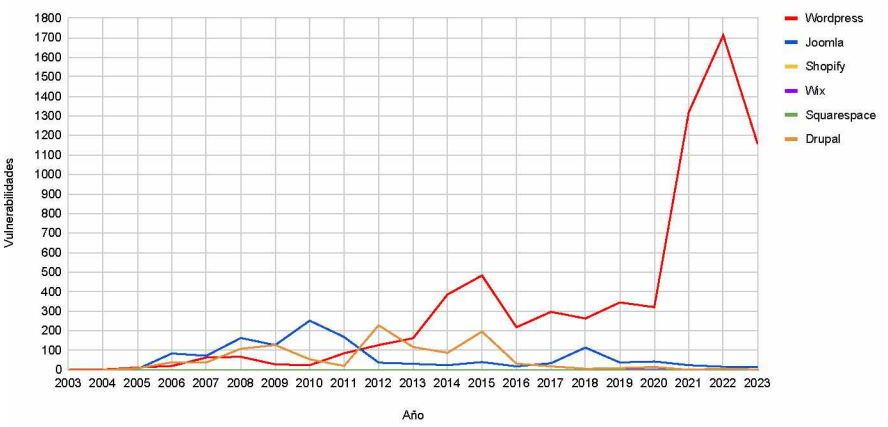

Fig. 1. Cantidad de vulnerabilidades de los principales CMS.

1.2 Antipatrón de vulnerabilidades

Las vulnerabilidades conocidas se registran desde finales del siglo XX en CVE. Y sea una vulnerabilidad de día cero o conocida, se trata de un resultado indeseable que genera decididamente consecuencias negativas para los usuarios. Esto es un antipatrón [9] [10]. Documentar vulnerabilidades como vulnerability antipattern (VAP) permite traducir conocimiento del dominio de la ciberseguridad [7] al de la ingeniería de software [8] y dependiendo de los autores, la documentación contiene desde una plantilla estándar hasta recomendaciones en términos de acciones para las etapas de requerimientos, diseño e implementación [11] que guían al desarrollador en la dirección de un software seguro.

2 Contribución

En el contexto de una propuesta de Doctorado en Informática de la Facultad de Informática de la UNLP investigamos estos aspectos para contribuir con un método para identificar y minimizar la reiteración de vulnerabilidades conocidas de aplicación directa a proyectos de software que utilicen WordPress. En la Figura 2 se esquematiza el método y a continuación se lo describe.

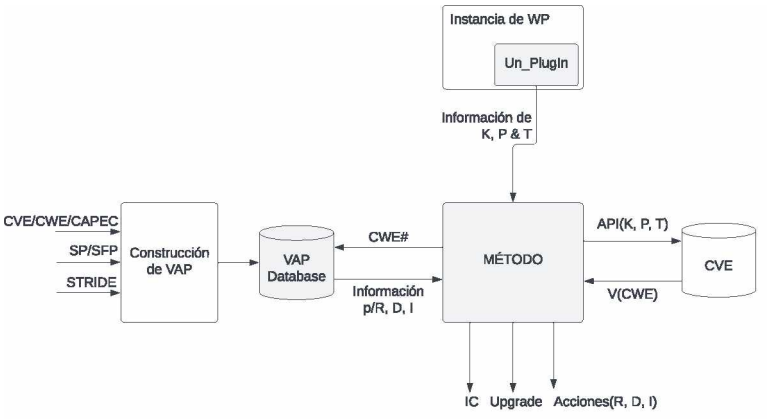

Fig. 2. Método propuesto.

El primer objetivo del trabajo es documentar los antipatrones de vulnerabilidades de WordPress. Atacamos el problema desde los componentes de su arquitectura: kernel, plug-in y themes y con información de fuentes como CWE [3], CAPEC [4], SP [13], SFP [12] y STRIDE [7].

A partir de los componentes de la arquitectura del sistema el método propone identificar vulnerabilidades conocidas desde CVE y recuperar los antipatrones de vulnerabilidad asociados, a partir de CWE, para proponer acciones sobre las etapas de requerimientos (R), diseño (D) e implementación (I) que minimicen la posibilidad de reiterarlas. Esto incluye el upgrade del componente y un indicador de copromiso (IC). A continuación se definen los requerimientos del método:

1. Diponibilidad de antipatrones de vulnerabilidades de la arquitectura a evaluar.
2. La información de entrada son los componentes de una arquitectura. Caso WordPress: kernel (K), plugin (P) y themes (T).

3. Debe identificar vulnerabilidades conocidas y recomendar acciones destinadas al programador.
4. Debe aportar trazabilidad de las acciones que recomienda sobre el SDLC.
5. Debe ser un proceso automatizado que para WordPress es Un_Plugin.
6. Deseable generar índice de compromiso (IC), similar a propuesto en CVSS[5].

3 Conclusiones

El VAP como fuente de información es una representación del conocimiento del dominio de la ciberseguridad [7] contenido en la documentación de vulnerabilidades conocidas referidas a una arquitectura particular. Esta información tiene el potencial para evitar fallas que han sido documentadas. En la arquitectura de un sistema como WordPress, encontramos los componentes que han presentado las fallas y deben ser actualizados o al menos mejorar su utilización. Un método automatizado, como parte de la arquitectura, puede asistir al ingeniero de software [8] sobre acciones para minimizar la reiteración de vulnerabilidades conocidas extensamente documentadas, actuando sobre cada una de las etapas del SDLC. Esto garantiza trazabilidad y un tratamiento de la seguridad en términos de requerimiento funcional. Por último, contar con una síntesis del nivel de exposición de un sistema, en términos de un indicador de compromiso, facilita la comparación entre proyectos similares y priorización de decisiones a fin de minimizar riesgos de exposición.

References

1. CVE Homepage, https://www.cve.org/. Last accessed 4 Abr 2024
2. IONOS CMS en comparativa, https://www.ionos.es/digitalguide/hosting/cms/cms-en-comparativa-los-gestores-de-contenido-mas-usados/. Last accessed 4 Abr 2024
3. CWE Homepage, https://cwe.mitre.org/. Last accessed 4 Abr 2024
4. CAPEC Homepage, https://capec.mitre.org/. Last accessed 4 Abr 2024
5. CVSS Homepage, https://www.first.org/cvss/. Last accessed 4 Abr 2024
6. Microsoft Threat Modeling Tool, https://learn.microsoft.com/eses/azure/security/develop/threat-modeling-tool. Last accessed 4 Abr 2024
7. CYBOK, https://www.cybok.org/. Last accessed 4 Abr 2024
8. SWEBoK, https://www.computer.org/education/bodies-of-knowledge/software-engineering. Last accessed 4 Abr 2024
9. Brown, W. J.: AntiPatterns: Refactoring software, architectures, and projects in crisis. Wiley, New York (1998).
10. Corry, A.: Retrospectives Antipatterns; Addison-Wesley (2021).
11. Nafees, T: Addressing the Knowledge Transfer Problem in Secure Software Development Through Anti-Patterns. PhD Thesis. Abertay University. Dundee, Escocia, UK (2019).
12. B. A. Calloni, D. Campara, and N. Mansourov. White Box Definitions of Software Fault Patterns. Final Report. Lockheed Martin Corporation and KDM Analytics, Inc. (2011).
13. Fernandez E.: Security Patterns in Practice, Designing Secure Arquitecture Using Software Patterns, Wiley, United Kingdom (2013)