УДК 004.56

О.Б. Рогозинський
В.В. Лукічов

ФУНКЦІЇ ТА МЕТОДИ СТАТИЧНОГО ТЕСТУВАННЯ БЕЗПЕКИ

Вінницький національний технічний університет

Анотація

Розглянуто методи та етапи роботи статичного тестування безпеки програмних застосунків (SAST). Визначено основні та допоміжні методи SAST та виконано їх аналіз. Результатами дослідження є інформація про кожен метод тестування, його переваги та недоліки.

Ключові слова: статичне тестування безпеки, SAST, програмне забезпечення, вразливості.

Abstract

Methods and stages of work of static software application security testing (SAST) are considered. The main and auxiliary methods of SAST were determined and their analysis was performed. The research results are information about each testing method, its advantages and disadvantages.

Keywords: static security testing, SAST, software, vulnerabilities.

Вступ

У світі швидко розвивається сфера інформаційних технологій, а разом з тим зростає і загроза кібербезпеці. З урахуванням частоти та складності кібератак, компанії та розробники програмного забезпечення все більше звертають увагу на засоби тестування безпеки застосунків. Статичне тестування є одним із важливих етапів у процесі виявлення та усунення потенційних вразливостей, що робить його темою важливою та актуальною для дослідження. Статичне тестування включає аналіз коду без виконання програми, що дозволяє виявити потенційні вразливості на ранніх стадіях розробки. Цей метод допомагає розробникам знайти і виправити помилки до того, як вони стануть загрозою для кінцевого користувача. [1].

Результати дослідження

Статичний аналіз коду є методом перевірки вихідного коду програми на наявність помилок, потенційних вразливостей та недоліків без його виконання. Цей метод дозволяє розробникам знайти проблеми ще на етапі написання коду, що значно знижує витрати на виправлення помилок на подальших етапах розробки.

Статичний аналіз може виконуватись вручну або за допомогою спеціалізованих інструментів. Ручний аналіз передбачає детальне вивчення коду розробниками або аудиторами безпеки, що може бути досить трудомістким процесом. З іншого боку, автоматичний статичний аналіз виконується за допомогою програмного забезпечення, яке автоматично сканує код на наявність відомих вразливостей і помилок. [2].

Основні завдання статичного аналізу коду включать:

- виявлення синтаксичних помилок;
- перевірка дотримання стандартів кодування;
- виявлення потенційних вразливостей;
- оцінка якості коду і виявлення можливих проблем з продуктивністю.

Статичні сканери вразливостей є спеціалізованими інструментами, які автоматично перевіряють вихідний код на наявність відомих вразливостей. Ці інструменти використовують бази даних відомих вразливостей та перевіряють код на їх відповідність. Статичні сканери можуть виявляти широкий спектр вразливостей, включаючи SQL-ін'єкції, міжсайтові скрипти (XSS), переповнення буфера та

інші. Статичні сканери можуть бути інтегровані в середовище розробки (IDE) або працювати як окремі інструменти.

Основні функції статичних сканерів вразливостей:

- автоматичне сканування коду на наявність вразливостей;
- надання звітів з детальними описами знайдених проблем і рекомендації щодо їх виправлення;
- інтеграція з інструментами управління версіями та системами безперервної інтерграції/безперервної доставки (CI/CD).

Статичний аналіз коду та статичні сканери вразливостей є найбільш розповсюдженими методами статичного тестування. Проте існують інші методи статичного тестування, які також можуть бути корисними для забезпечення безпеки програмного забезпечення. До таких методів належать формальна верифікація, аналіз байт-коду та проміжного коду, а також аналіз залежностей. Кожен з цих методів має свої особливості, переваги та обмеження.

Формальна верифікація - це метод перевірки коректності програмного забезпечення шляхом використання математичних моделей і логічних доведень. Цей метод дозволяє довести, що програма відповідає специфікаціям і не містить певних типів помилок або вразливостей.

Основні етапи формальної верифікації:

- моделювання: створення формальної моделі програми або її частин. Це може включати абстрактні описи алгоритмів, структур даних та взаємодії між компонентами;
- специфікація: визначення формальних специфікацій, які описують бажану поведінку програми. Специфікації можуть включати інваріанти, передумови та постумови для функцій, а також властивості системи, які повинні бути виконані;
- доведення: використання логічних методів і автоматизованих інструментів для доведення того, що модель програми відповідає специфікаціям. Це може включати перевірку логічної еквівалентності, символьне виконання та аналіз моделі.

Формальна верифікація може бути особливо корисною для критично важливих систем, де помилки можуть мати серйозні наслідки, таких як авіаційні системи, медичне обладнання та банківське програмне забезпечення. Проте цей метод має і свої обмеження, зокрема високу складність і вартість впровадження, а також вимоги до високої кваліфікації розробників і аналітиків.

Аналіз байт-коду та проміжного коду - це методи статичного тестування, які полягають у перевірці проміжного представлення програми, такого як байт-код Java або CIL-код .NET. Ці методи можуть бути корисними, оскільки дозволяють виявляти вразливості на рівні, ближчому до виконуваного коду, ніж вихідний код.

Основні переваги аналізу байт-коду та проміжного коду:

- мовна незалежність: аналіз проміжного коду може бути застосований до програм, написаних на різних мовах програмування, якщо вони компілюються до одного і того ж проміжного представлення;
- детальніша перевірка: проміжний код може містити додаткову інформацію, таку як результати оптимізацій компілятора, яка може бути корисною для виявлення вразливостей;
- інтеграція з інструментами безпеки: багато інструментів безпеки, такі як Fortify та Veracode, підтримують аналіз байт-коду та проміжного коду, що дозволяє інтегрувати цей метод у загальний процес забезпечення безпеки.

Проте аналіз байт-коду та проміжного коду має і свої обмеження:

- залежність від компілятора: якість і точність аналізу можуть залежати від характеристик компілятора, який використовується для створення проміжного коду;
- відсутність вихідного коду: аналіз проміжного коду не дозволяє отримати доступ до вихідного коду, що може ускладнити розуміння та виправлення знайдених проблем.

Аналіз залежностей - це метод статичного тестування, який полягає у перевірці зовнішніх бібліотек і фреймворків, які використовуються програмою, на наявність відомих вразливостей. Цей метод є важливим, оскільки багато сучасних програм залежить від великої кількості сторонніх компонентів, які можуть містити вразливості.

Основні етапи аналізу залежностей:

- ідентифікація залежностей: виявлення всіх зовнішніх бібліотек і фреймворків, які використовуються програмою. Це може включати аналіз файлів конфігурації, маніфестів та інших метаданих;

- перевірка на вразливість: перевірка кожної залежності на наявність відомих вразливостей. Це може здійснюватися за допомогою спеціалізованих баз даних;
- оцінка ризиків: оцінка ризиків, пов'язаних з використанням залежностей, з урахуванням їх критичності та можливості експлуатації вразливостей;
- виправлення та оновлення: при виявленні вразливостей необхідно оновити залежності до безпечних версій або знайти альтернативні рішення.

Аналіз залежностей має кілька важливих переваг:

- підвищення безпеки: виявлення і виправлення вразливостей у зовнішніх бібліотеках допомагає підвищити загальний рівень безпеки програмного забезпечення;
- автоматизація: багато інструментів для аналізу залежностей підтримують автоматичну перевірку і оновлення залежностей, що знижує витрати часу і зусиль розробників.

Однак цей метод має і свої обмеження:

- обмежена видимість: аналіз залежностей не завжди дозволяє виявити вразливості в приватних або спеціалізованих бібліотеках, які не входять до загальнодоступних баз даних;
- вразливості нульового дня: метод не дозволяє виявити нові вразливості, які ще не були зареєстровані в базах даних.

Висновки

Розглянуто методи та принципи статичного тестування безпеки програмних застосунків. Інші методи статичного тестування, такі як формальна верифікація, аналіз байт-коду та проміжного коду, а також аналіз залежностей, доповнюють традиційні підходи до забезпечення безпеки програмного забезпечення.

Використання цих методів дозволяє підвищити рівень безпеки і якість коду, проте вимагає додаткових ресурсів і спеціалізованих знань. Вибір конкретного методу або комбінації методів залежить від специфіки проекту, вимог до безпеки та ресурсів, доступних для розробки.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Williams, A., Wichers, D. OWASP Top 10: The Ten Most Critical Web Application Security Risks. 2017.
2. OWASP Foundation. Source Code Analysis Tools. [Електронний ресурс]. – Режим доступу: https://owasp.org/www-community/Source_Code_Analysis_Tools

Рогозинський Олександр Борисович – студент групи 1БС-20б, факультет інформаційних технологій і комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, oleksandr.rohozynsky@gmail.com.

Лукічов Віталій Володимирович – к. т. н., доцент, доцент кафедри захисту інформації, Вінницький національний технічний університет, м. Вінниця, e-mail: lukichov.vitalyi@vntu.edu.ua.

Oleksandr Rohozynskyi – student of group 1BS-20b, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, oleksandr.rohozynskyi@gmail.com.

Vitaly Lukichev – associate professor at the Department of Information Security, Vinnytsia National Technical University, Vinnytsia, email: lukichov.vitalyi@vntu.edu.ua .