*Рекута Владислав Віталійович, студент*

*Університет Короля Данила*

**РОЗРОБКА ТА ОЦІНКА МЕТОДІВ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ БЕЗПЕКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Програмне забезпечення, як невіднята частина цифрового світу, вимагає високого рівня надійності та безпеки. Життєвий цикл розробки [1, c.1-10] підкреслює важливість ефективного тестування, а безпека стає ключовим компонентом для стійкості програмного забезпечення [2, c.7-13].

Автоматизоване тестування безпеки використовує різноманітні методи та інструменти , і вибір конкретного методу залежить від контексту проєкту. Розгляд різних методів, таких як SAST, DAST, IAST, оцінка вразливостей та тестування на проникнення, є важливим для забезпечення повного захисту.

Швидкий розвиток технологій породжує нові загрози для кібербезпеки, і компанії повинні бути готові до цих викликів, використовуючи відповідні заходи для захисту інформації та систем [3, c.194; 4, c.72-75; 5, с. 8]. Забезпечення високого рівня кібербезпеки стає невіднятою частиною успішної розробки та експлуатації програмного забезпечення в умовах постійного росту та зміни технологічного ландшафту.

Останні тенденції у сфері кібербезпеки свідчать про зростання складності та вдосконалення методів кібератак [6, 72-75; 7, c.8]. Різноманітність мотивацій, таких як фінансові, політичні та особисті, підкреслюють необхідність постійного вдосконалення заходів безпеки та підвищення рівня обізнаності стосовно різновидів кіберзагроз.

Зловмисне програмне забезпечення та програмне забезпечення вимагачі є серйозною загрозою для інформаційних систем, і спостережене зростання їхньої кількості вказує на активність кіберзлочинців. Забезпечення найвищого рівня безпеки інформаційних систем вимагає комплексного та постійного підходу до кібербезпеки, включаючи удосконалення технологій захисту, навчання персоналу та активний моніторинг нових загроз.

Історія кібербезпеки служить прикладами визначних інцидентів, які наголошують на необхідності постійної уваги до цього питання. Захист від різних методів кібератак вимагає комплексного підходу, включаючи освіту користувачів, вдосконалення технічних засобів безпеки та систематичне оновлення програмного забезпечення.

Загалом, забезпечення безпеки програмного забезпечення є вирішальним завданням для забезпечення захищеності інформаційних систем у сучасному цифровому середовищі.

Автоматизована генерація тестів використовує різні методи, включаючи випадкове тестування чорного ящика та символьне виконання білого ящика. Обидва підходи є важливими для забезпечення ефективності та повноти тестування програмного забезпечення [8, c.72-75]. Випадкове тестування та фаззінг є ефективними методами, які дозволяють систематично та ефективно виявляти помилки та вразливості безпеки. Генерація тестів із символьним виконанням вирізняється як найточніший підхід, дозволяючи детально аналізувати структуру обчислень програми. Статична та динамічна генерація тестів із символьним виконанням мають свої переваги та обмеження.

Однак, автоматизована генерація тестів також має свої виклики та складнощі. По-перше, вона потребує високої кваліфікації та досвіду тестувальників, які повинні знати особливості та можливості використовуваних методів та інструментів. По-друге, вона потребує великої кількості часу та ресурсів для підготовки, налаштування, запуску та аналізу тестів, особливо для великих та складних програм. По-третє, вона потребує постійної підтримки та оновлення тестів, щоб враховувати зміни в програмному забезпеченні, яке тестується, а також в технологіях та стандартах тестування.

Тому, для ефективного та якісного використання автоматизованої генерації тестів, слід дотримуватися наступних рекомендацій:

* вибирати найбільш відповідні методи та інструменти для тестування залежно від типу, мети, рівня та об’єкта тестування;
* планувати та проектувати тести заздалегідь, визначаючи очікувані результати, критерії успіху, ризики та пріоритети;
* використовувати модульне та інкрементальне тестування, щоб перевіряти окремі частини програми та їх взаємодію;
* використовувати зворотний зв’язок та інструментування, щоб отримувати детальну інформацію про хід та результати тестування, а також виявляти та усувати проблеми.

Оцінювати та аналізувати ефективність та якість тестування, використовуючи метрики та показники, такі як покриття, відповідність, надійність, продуктивність тощо.

Динамічне символьне виконання виявляє програмні шляхи, надаючи компроміси у використанні ресурсів. Підходи, такі як конколічне виконання та розгалуження стану, розв’язують проблему ефективності та аналізують структуру програм. Вибір стратегії – конкретизації чи абстрагування – залежить від умов використання та цілей аналізу.

Конколічне виконання поєднує конкретне та символьне виконання, використовуючи конкретні вхідні дані для запуску програми та символьні змінні для представлення невідомих вхідних даних. Це дозволяє зменшити простір пошуку та уникнути паразитних шляхів, які не можуть бути досягнуті жодними вхідними даними. Однак, конколічне виконання може пропустити деякі шляхи, які залежать від взаємодії між конкретними та символьними вхідними даними, а також від властивостей символьного розв’язувача.

Розгалуження стану використовує символьне виконання для дослідження всіх можливих шляхів виконання програми, зберігаючи та відновлюючи стани програми на кожному розгалуженні. Це дозволяє знайти всі вхідні дані, які викликають певну поведінку програми, таку як помилки, вразливості, або відхилення від специфікації. Однак, розгалуження стану стикається з проблемою експоненціального зростання кількості станів, які потрібно зберігати та відновлювати, а також з проблемою вирішення складних символьних обмежень, які можуть бути нерозв’язними або невиразними.

Зведення та злиття станів ускладнюють вирішення обмежень. Компроміси у формі повного розділення чи злиття станів визначаються алгоритмами оцінювання, балансуючи зменшенням шляхів зі збільшенням часу вирішення. Ефективне вирішення обмежень – ключ у генерації тестів. Інструменти, як DART чи KLEE, використовують різні методи та розв’язувачі для вирішення обмежень залежно від властивостей програми.

Розпаралелювання та концепція “тестування як послуги” обіцяють прискорення і автоматизацію тестування, дозволяючи ефективно вирішувати обмеження та забезпечуючи доступність тестових сервісів. Проєкти, такі як SAGE, активно втілюють ці стратегії для полегшення розробки та порівняння програмного забезпечення через автоматизоване тестування.

Розпаралелювання дозволяє використовувати багатопроцесорні або розподілені системи для одночасного виконання багатьох шляхів або станів, зменшуючи час та пам’ять, необхідні для тестування. Розпаралелювання може бути реалізоване на рівні шляхів, станів, обмежень, або вхідних даних, в залежно від характеру програми та методу символьного виконання. Однак, розпаралелювання також має свої виклики, такі як синхронізація, комунікація, балансування навантаження, та відмово стійкість.

Тестування як послуги (TaaS) передбачає надання тестування програмного забезпечення як конкурентоспроможної та легкодоступної онлайн послуги, а також ідею повністю автоматизованого тестування в хмарі. Мета полягає в тому, щоб використовувати величезні та еластичні хмарні ресурси, щоб автоматизоване тестування стало практичною реальністю для реального програмного забезпечення. Служба тестування програмного забезпечення дозволяє користувачам і розробникам завантажувати своє програмне забезпечення, вказувати необхідний тип тестування, ініціювати процес і отримувати повний звіт із результатами. У професійних умовах TaaS може легко інтегруватися в процес розробки, проводячи постійне тестування під час написання коду. Крім того, TaaS може служити загальнодоступним сервісом сертифікації, що полегшує порівняння надійності та безпеки різних програмних продуктів.

Фаззинг Whitebox із SAGE є потужним інструментом для автоматичного тестування, здатним виявляти вразливості в програмах, зокрема в області аналізу файлів та пакетів. Використання динамічного символьного виконання та техніки “пошук поколінь” роблять його ефективним для великих програм, а висока масштабованість робить його перспективним інструментом для виявлення безпекових проблем. Успішні результати виявлення вразливостей у програмах Microsoft свідчать про його потенціал та ефективність у практичних сценаріях тестування безпеки. SAGE, використовуючи символьне виконання, успішно виявляє вразливості в Microsoft Windows, здійснивши значний вплив у промисловості. Його інтеграція в Project Springfield підтверджує високу довіру Microsoft до цього інструменту в контексті безпеки ПЗ. S2E, заснований на символьному виконанні “in vivo”, ефективний для тестування системного коду, а його різноманітність в застосуванні свідчить про його гнучкість та успішність в індустрії та науковому середовищі. Різні моделі узгодженості та оптимізації, впроваджені в S2E, роблять його універсальним для вирішення викликів у тестуванні та забезпеченні безпеки програм та систем. Всі ці підходи до автоматизованої генерації тестів відображають різноманітні техніки та стратегії для забезпечення якості та безпеки програмного забезпечення [9, с. 216–233].

Однією з особливостей фаззингу Whitebox із SAGE є його здатність генерувати тести, які покривають всі можливі шляхи виконання програми, використовуючи символьні обмеження, які збираються під час виконання. Це дозволяє виявляти вразливості, які можуть бути приховані в глибоких або рідкісних шляхах, які не можуть бути досягнуті звичайним фаззингом. Крім того, SAGE використовує техніку “пошук поколінь”, яка дозволяє ефективно використовувати вхідні дані, які вже викликали помилки, для генерації нових тестів, які можуть викликати інші помилки. Це дозволяє збільшити шанси виявлення вразливостей, які можуть бути пов’язані або залежні одна від одної.

Ще однією перевагою фаззингу Whitebox із SAGE є його масштабованість та інтеграція з хмарними сервісами. SAGE може використовувати багатопроцесорні або розподілені системи для паралельного виконання багатьох тестів, зменшуючи час та пам’ять, необхідні для тестування. Крім того, SAGE є частиною Project Springfield, який надає тестування як хмарну послугу, дозволяючи користувачам та розробникам легко та швидко тестувати своє програмне забезпечення на вразливості безпеки. Це забезпечує доступність, економічність та надійність тестування, а також можливість порівняння та сертифікації якості та безпеки програмних продуктів.

В ході дослідження був проаналізований стандарт CCOF, який визначає цілі та вимоги для управління ризиками безпеки інформації. Проаналізовані основні принципи, контролі та переваги CCOF, а також його відношення до різних стандартів та регуляторних вимог. Спостерігається, що CCOF дозволяє ефективно впроваджувати контролі безпеки в організаціях, що використовують інформаційні технології, та спрощує аудит та перевірку ефективності цих контролів.

Висновки щодо CCOF вказують на його значущість у забезпеченні єдиної мови та критеріїв для управління ризиками безпеки інформації в організаціях. Використання CCOF для порівняння різних методів автоматизованого тестування безпеки програмного забезпечення визначено як обґрунтований підхід, що може сприяти якості та надійності програмного забезпечення.

**Література:**

1. Чумаченко І. В., Чумаченко О. В., Литвиненко О. В., Шевченко О. В. Технології розроблення програмного забезпечення. Київ: КНУБА, 2019. С. 1-10.
2. ДСТУ ISO/IEC 12207:2016 Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення. Київ: Мінекономрозвитку України, 2016. С. 7-13.
3. Шевченко О. В., Чумаченко І. В., Чумаченко О. В., Литвиненко О. В. Якість програмного забезпечення та тестування: базовий курс. Київ: КНУБА, 2020. С. 194.
4. Шевченко О. В., Чумаченко І. В., Чумаченко О. В., Литвиненко О. В. Тестування програмного забезпечення. Київ: КНУБА, 2021. С. 72-75.
5. Литвиненко О. В. Застосунок для аналізу результатів тестування програмного забезпечення: автореф. дис. на здобуття наук. ступеня канд. техн. наук: спец. 05.13.06 «Інформаційні технології». Київ: КНУБА, 2022. С. 8.
6. Шевченко О. В., Чумаченко І. В., Чумаченко О. В., Литвиненко О. В. Тестування програмного забезпечення. Київ: КНУБА, 2021. С. 72-75.
7. Литвиненко О. В. Застосунок для аналізу результатів тестування програмного забезпечення: автореф. дис. на здобуття наук. ступеня канд. техн. наук: спец. 05.13.06 «Інформаційні технології». Київ: КНУБА, 2022. С. 8.
8. Авраменко А.С., Авраменко В.С., Косенюк Г.В. Тестування програмного забезпечення. Навчальний посібник. – Черкаси: ЧНУ імені Богдана Хмельницького, 2017. С. 72-75.
9. Авгерінос, Т., Ча, С.К., Хао, Б.Л.Т., Брамлі, Д.: "AEG: автоматичне генерування експлойтів." У: Симпозіум з безпеки мережі та розподілених систем (2011), с. 216–233.