РОЗРОБКА МЕТОДОЛОГІЇ АВТОМАТИЗОВАНОГО АНАЛІЗУ ВРАЗЛИВОСТЕЙ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З ВИКОРИСТАННЯМ ТЕХНІК СИМВОЛЬНОГО ВИКОНАННЯ ТА ФАЗИНГУ

Маринкевич О. М.[1]

[1]*Харківський національний університет внутрішніх справ*
alexandr.mr253@gmail.com

Вступ

Актуальність дослідження зумовлена зростаючою потребою в ефективних методах автоматичного виявлення вразливостей програмного забезпечення для підвищення кібербезпеки. Метою даної роботи є дослідження та обґрунтування перспективності поєднання технік символьного виконання та фазингу для більш глибокого та всебічного аналізу програм на наявність потенційних слабких місць. Предметом дослідження є методи автоматизованого аналізу вразливостей програмного забезпечення, зокрема фазинг та символьне виконання, а також їхні комбінації.

Основна частина

Автоматичний пошук вразливостей є важливим напрямом у кібербезпеці, спрямованим на виявлення слабких місць у програмному забезпеченні без прямої участі людини. Одним з ефективних методів такого пошуку є фаз-тестування (фазинг). Ця техніка передбачає автоматизоване генерування великої кількості некоректних, випадкових або несподіваних вхідних даних для програми з метою спровокувати її некоректну роботу, збої або виявити

вразливості безпеки, такі як переповнення буфера чи витоки пам'яті. Аналізуючи реакцію програми на ці аномальні вхідні дані, дослідники можуть ідентифікувати потенційні точки відмови, які можуть бути використані зловмисниками. Фазинг допомагає виявляти не лише відомі, але й невідомі раніше помилки, підвищуючи загальну якість та стійкість програмного забезпечення до кібератак. Інтеграція фаз-тестування в процес розробки дозволяє проактивно виявляти та усувати вразливості на ранніх етапах, знижуючи ризики та витрати на їхнє виправлення після випуску продукту [1].

Символьне виконання та фазинг – це два різні, але важливі підходи до автоматичного пошуку вразливостей у програмах. *Фазинг*, як випливає з джерела, полягає у безперервному «згодовуванні» програмі великої кількості випадкових, некоректних або неочікуваних даних на вхід. Мета фазингу – викликати збій у роботі програми або виявити незвичайну поведінку, що може свідчити про помилку чи вразливість. *Символьне виконання*, на відміну від цього, намагається проаналізувати код програми, розглядаючи вхідні дані як символи, а не конкретні значення. Це дозволяє дослідити всі можливі шляхи виконання програми та виявити потенційні проблеми, навіть ті, які важко знайти за допомогою випадкових тестів. Хоча фазинг ефективний у виявленні багатьох поширених вразливостей завдяки своїй інтенсивності, символьне виконання може глибше аналізувати логіку програми та знаходити складніші дефекти [2].

Існуючі методи автоматичного пошуку вразливостей, зокрема фазинг у його різних формах, мають певні обмеження, які роблять їх не ідеальними. *Фазинг "чорного ящика"*, хоч і швидкий, є менш ефективним у виявленні глибоко прихованих помилок, оскільки не аналізує внутрішню структуру програми. *Фазинг "білого ящика"*, навпаки, хоч і більш результативний у знаходженні складних вразливостей завдяки аналізу покриття коду, є повільним і вимагає доступу до вихідного коду, що не завжди можливо, особливо при аналізі комерційного програмного забезпечення або зловмисного ПЗ. *Фазинг*

“сірого ящика” намагається знайти баланс, але все одно може пропускати вразливості, якщо згенеровані вхідні дані не потрапляють у критичні ділянки коду або не відповідають очікуваному формату вхідних даних програми (як у випадку з граматичним фазингом, який є складним для реалізації) [3].

Незважаючи на ефективність окремих методів, поєднання символьного виконання та фазингу представляє собою перспективний напрямок для вдосконалення автоматизованого аналізу вразливостей. Ідея полягає в тому, щоб використати сильні сторони кожного підходу для компенсації слабких. Наприклад, символьне виконання може генерувати цільові вхідні дані, які направляють фазер до глибоко прихованих ділянок коду, збільшуючи таким чином покриття та ймовірність виявлення складних вразливостей. З іншого боку, фазинг може забезпечити швидке дослідження великої кількості різноманітних вхідних даних, виявляючи неочікувані сценарії, які можуть бути пропущені при статичному символьному аналізі. Інтеграція цих технік може включати використання результатів символьного виконання (наприклад, обмежень на вхідні дані, шляхів виконання) для керування процесом генерації вхідних даних фазером, або ж застосування фазингу для дослідження конкретних станів програми, досягнутих під час символьного виконання. Такий гібридний підхід має потенціал значно підвищити ефективність та глибину автоматизованого аналізу вразливостей порівняно з використанням кожного методу окремо [4].

Висновок

Проведене дослідження підтвердило важливість автоматичного пошуку вразливостей як ключового елементу забезпечення кібербезпеки. Розглянуто основні методи такого пошуку, зокрема фазинг та символьне виконання, виявлено їхні переваги та обмеження. Показано, що кожен з цих методів окремо не є ідеальним рішенням для виявлення всього спектру потенційних загроз. У роботі обґрунтовано перспективність інтеграції символьного

виконання та фазингу, що дозволяє поєднати глибокий аналіз коду з інтенсивним тестуванням різноманітними вхідними даними, підвищуючи таким чином ефективність виявлення складних вразливостей. Запропонований гібридний підхід має потенціал стати більш потужним інструментом у боротьбі з кіберзагрозами.

Список використаних джерел

1. Фаз-тестування: Виявлення вразливостей у програмному забезпеченні. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/fuzz-testing?srsltid=AfmBOorR5Y5oxZteJB14NZ04iuuPBQrmKiv3767PLaFGCu6dwU5togRS (дата звернення 11.04.2025);

2. Фазинг Частина 1: Теорія, URL: https://sayfer.io/uk/ (дата звернення 11.04.2025);

3. Техніка нечіткого тестування та її використання в задачах кібербезпеки, URL: http://www.kibernetika.org/volumes/2022/numbers/01/articles/18/18.pdf (дата звернення 11.04.2025);

4. Fuzzing-тестування – ідеї та приклади, URL: https://devzone.org.ua/post/fuzzing-testuvannia-ideyi-ta-pryklady (дата звернення 11.04.2025).