

ПРИНЦИПИ ПОБУДОВИ ТА СТРУКТУРНА СХЕМА АВТОМАТИЗОВАНОЇ СИСТЕМИ ДЛЯ АНАЛІЗУ БІНАРНИХ ВРАЗЛИВОСТЕЙ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В. В. Карко¹, М. В. Грайворонський¹

¹Національний технічний університет України «Київський політехнічний інститут»

Анотація

В роботі наводяться вимоги та принцип побудови автоматизованої системи для аналізу бінарних вразливостей програмного забезпечення, тобто системи, яка для заданого бінарного файлу, знаходить множину потенційних вразливостей, перевіряє їх та робить висновок, щодо можливості практичної експлуатації вразливостей.

Ключові слова: бінарні вразливості програмного забезпечення, експлуатація вразливостей, статичний аналіз коду, динамічний аналіз коду

Вступ

З розвитком технологій та засобів для створення програмного забезпечення, а також з його ускладненням, зростає кількість помилок присутніх у ньому. В процесі користування програмним засобом, можуть проявлятися вразливості, які призводять до збою в роботі програми та становлять загрозу безпеці інформації.

Визначення того, чи може вразливість бути експлуатована на практиці — є нетривіальною, складною задачею, вирішення якої вимагає залучення висококваліфікованих спеціалістів з подальшим аналізом бінарної вразливості. Тому, постає питання створення автоматичних та автоматизованих систем для аналізу бінарних вразливостей програмного забезпечення. На відміну від інших підходів для забезпечення безпеки, таких як системи виявлення вторгнень¹, аналіз вразливостей націлений на попереднє виявлення та усунення причин появи загроз, аніж виявлення та блокування атак.

У роботі формулюються завдання, вимоги та принципи побудови універсальної, автоматизованої системи для аналізу бінарних вразливостей програмного забезпечення.

На даний момент існує декілька подібних розробок AEG [?], BitBlaze [?], проте їх застосування на практиці обмежене простими програмами та відсутністю повного циклу етапів для аналізу бінарних вразливостей.

1. Завдання автоматизованої системи

Завданням автоматизованої системи для аналізу бінарних вразливостей програмного забезпечення є — знаходження вразливостей у заданому бінарно-

му файлі, побудова відповідних експлойтів² та їх перевірка, формування висновку, щодо можливості експлуатації вразливостей, а у разі неможливості автоматичної перевірки — представлення детальної інформації про пройдені системою етапи для максимального полегшення аналізу фахівцем.

2. Вимоги до системи

Система має відповідати наступним вимогам:

- Можливість роботи системи без вихідного коду. Більшість програм розповсюджується без вихідного коду. Бінарний код — це саме те, що безпосередньо буде виконуватися, а тому аналіз бінарного коду дасть більше інформації з точки зору захищеності програмного забезпечення, на відміну від аналізу вихідного коду, який може давати хибні результати через оптимізацію компіляторів.
- Робота з певною архітектурою. Бінарний аналіз є складною задачею. Існує багато різних архітектур процесорів, з безліччю різних інструкцій, кожна з яких має складну семантику, може поводитися по різному в залежності від операндів, мати побічні неявні ефекти (вплив на прапорці процесору). Тому, для практичної реалізації система має працювати з конкретно обраною архітектурою, мати набір архітектурно-залежних реалізацій самої себе, або працювати з яким-небудь проміжним рівнем представлення бінарного коду. Наприклад, проект BitBlaze використовує для цього власну мову Vine Intermediate Language [?].
- Робота системи в автоматичному та автоматизованому режимі.

¹Intrusion Detection System (IDS) — програмний або апаратний засіб, призначений для виявлення фактів несанкціонованого доступу в систему або мережу, виявлення та блокування підозрілої активності

²Експлойт (від англ. *exploit* — експлуатувати, використовувати) - комп'ютерна програма спеціального призначення, яка використовує вразливість наявну в програмному забезпеченні для проведення атаки на автоматизовану систему

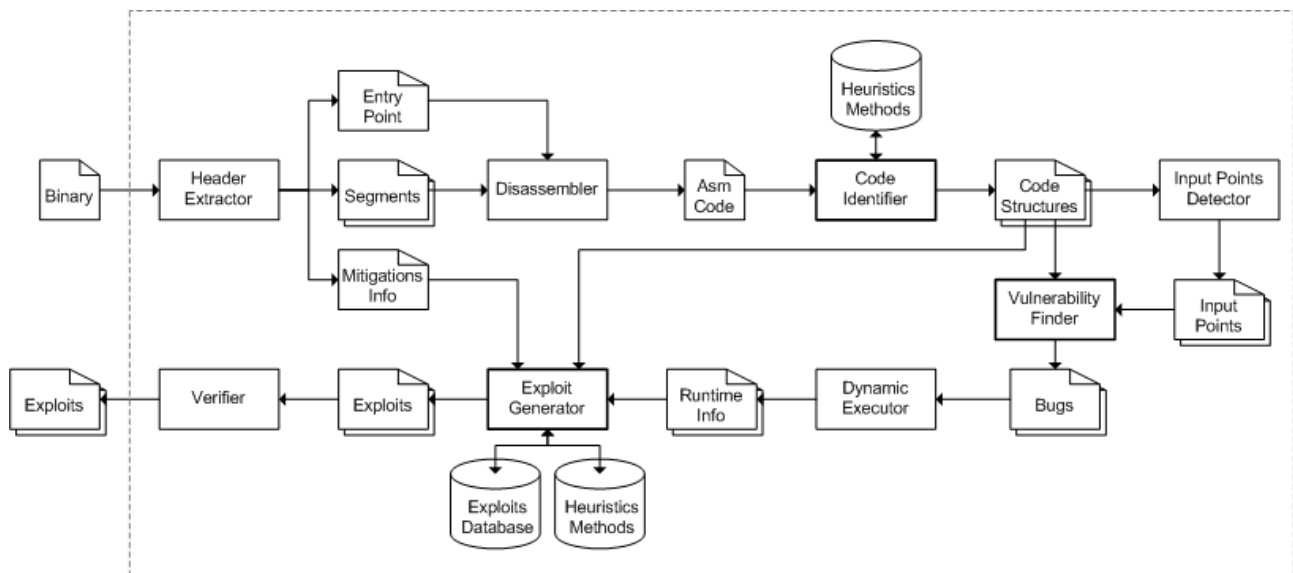


Рис. 1. Структурна схема автоматизованої системи для аналізу бінарних вразливостей програмного забезпечення

Система має працювати в автоматичному режимі (самостійно) та проходити всі етапи, фіксуючи отримані результати. У разі неможливості отримання однозначних результатів, переходити в автоматизований режим (із залученням фахівця), надавати йому всю необхідну інформацію та можливість внести правки і продовжити автоматичну роботу системи.

- Підвищенні вимоги до використання ресурсів системою.
Під час роботи системи створюється величезна кількість об'єктів для аналізу, обстежується практично незліченна кількість гілок програми, тому мають застосовуватися оптимізовані алгоритми.
- Зниження інтерференції системи та програми.
Необхідно забезпечити максимальне зниження впливу роботи системи на програму, яка є предметом дослідження, оскільки вони виконуються в одному адресному просторі.

3. Структурна схема системи

На рис. 1 зображено структурну схему автоматизованої системи для аналізу бінарних вразливостей програмного забезпечення. На вхід системі передається бінарний файл для аналізу, кожен блок системи виконує свою частину аналізу, передаючи отримані дані до інших блоків. На вихід система видає перевірені експлойти, що є підтвердженням можливості експлуатації вразливості.

Розглянемо детально кожний блок.

Блок екстракції заголовку бінарного файлу
Header Extractor — на вхід подається бінарний файл для аналізу, де відбувається зчитування його заголовку (ELF³, PE⁴, тощо), підставляється відповідна

реалізація інших блоків, згідно з визначеною архітектурою, розрядністю та форматом файлу. На основі інформації в заголовку визначається точка входу в програму (*entry point*), сегменти та таблиці програми (*segments*), а також інформація про засоби протидії експлуатації, наприклад: NX/XD-bit⁵, Stack canary⁶, ASLR⁷ (*mitigations info*).

Блок дизасемблювання *Disassembler* — генерує асемблерний код (*asm code*) відповідних сегментів.

Блок ідентифікації коду *Code Identifier* — відновлює структурні одиниці програмного коду, на основі асемблерного коду формує відповідну структуру даних для спрощення подальшого аналізу.

Використовуються методи для ідентифікації функцій, виявлення системних викликів, умовних переходів, циклів [?]. Така операція є надзвичайно складною, через складність методів та відсутність семантичної інформації, наявної на рівні вихідного коду, тому точне встановлення програмних структур неможливе.

На рівні бінарного коду:

- Не існує такої абстракції як *функція*, існують тільки переходи за певними адресами (для процесорів x86 це команди *call*, *jmp*, та інші різновиди).
- Не існує буферів певного типу та розміру, існує лише суцільна пам'ять. Саме ця відмінність є причиною появи помилок переповнення буферу (*buffer overflow*).
- Не існує типів даних та користувацьких типів даних, існують лише регістри процесора та па-

⁵NX/XD-bit - (від англ. *no execute bit* в термінах фірми AMD або *execute disable bit* в термінах фірми Intel) апаратна реалізація механізму запобігання виконання даних

⁶Stack canary - спосіб виявлення переповнення буферу

⁷Address space layout randomization (ASLR) - технологія запобігання експлуатації, шляхом розташування випадковим чином структур процесу в адресному просторі

³ELF - Executable and Linkable Format

⁴PE - Portable Executable

м'ять, а дані можуть бути: цілими значеннями (знаковими або без знаковими), стрічками, тощо.

Для вирішення цього завдання, застосовується набір евристичних методів або інших підходів (*heuristics methods*), які зберігаються в окремому модулі, сховищі, тощо.

Усі виявленні структури фіксуються у журналі, для перегляду і корегування фахівцем.

Блок видає асемблерний код з набором ідентифікованих структур (*code structures*).

Блок виявлення вводу інформації *Input points Detector* — призначений для виявлення точок в програмі де вводяться дані (*input points*). Такими точками можуть бути аргументи передані програмі, змінні середовища, системні виклики зчитування даних з стандартного вводу або файлу, сокету тощо [?].

Блок виявлення вразливостей *Vulnerability Finder* — за поданим на вхід асемблерним кодом з набором ідентифікованих структур та множиною точок вводу даних, знаходиться множина потенційних вразливостей (*bugs*) та відповідні їм вхідні дані.

Застосовуються методи статичного та динамічного аналізу коду [?], [?]: символічний аналіз (*symbolic analysis*) та відстеження потоку вхідних даних (*taint analysis*). Виявленні потенційні вразливості фіксуються у журналі для перегляду і корегування фахівцем.

Дана задача є складною та може бути темою окремого дослідження.

Блок динамічного виконання *Dynamic Executor* — запускає бінарний файл у режимі відладки, вводить

дані для кожної знайденої вразливості та отримує інформацію про збій (*runtime info*).

Блок генерації експлойту *Exploit Generator* — за поданою інформацією про збій, асемблерний код з набором ідентифікованих структур та інформацією про засоби протидії експлуатації генерує експлоїт. Для цього застосовується набір заздалегідь заготовлених шаблонів експлойтів різних типів (*exploits database*) та набір евристичних або інших методів (*heuristics methods*).

Експлоїт може будуватися поетапно, спершу необхідно отримати контроль над вказівником інструкцій (для архітектури сімейства x86 це регістр *eip*), а наступні кроки залежать від типу експлойту та застосованих засобів протидії експлуатації.

Етапи побудови експлойту фіксуються у журналі для перегляду і корегування фахівцем.

Блок перевірки *Verifier* — застосовує отриманий експлоїт на новому екземплярі програми та робить висновок про можливість експлуатації вразливості.

Висновки

В роботі приводяться основні вимоги до автоматизованої системи для аналізу бінарних вразливостей програмного забезпечення, наводиться детальна структурна схема системи з описом кожного блоку та описом проблем, які необхідно вирішити для побудови такої системи. Найбільш складними для реалізації є блоки виявлення вразливостей та генерації експлойту. На поточний момент, не існує єдиного, ефективного рішення цих задач, а тому такі задачі можуть бути темами для окремих детальних досліджень.