УДК 528.481
DOI: 10.25140/2411-5363-2021-1(23)-109-117

Ігор Карпачев, Володимир Казимир

ВИЯВЛЕННЯ ШКІДЛИВИХ ДОДАТКІВ ОС АНДРОІД ПО СИГНАТУРІ ФУНКЦІОНАЛЬНОГО ЛАНЦЮЖКА

На сьогодні операційна система Андроід представляє собою розповсюджену мобільну платформу номер один, яка дає функціональну свободу користувачеві. Саме широке розповсюдження ОС робить її привабливим середовищем для зростаючої кількості шкідливого програмного забезпечення. У цій статті запропоновано використовувати методи із біоінформатики для прискорення пошуку збігу послідовностей шкідливих ланцюжків API функцій. Описаний комбінований метод динамічного порівняння послідовностей API викликів, в якому використовуються алгоритми локального та глобального вирівнювання. Обґрунтовані переваги запропонованого методу перед існуючими підходами.

***Ключові слова**: СФЛД; ОС Андроїд; API функція; вирівнювання послідовностей.*
Табл.: 1. Рис.: 6. Бібл.: 12.

Актуальність теми дослідження. Популярність та впровадження смартфонів сильно стимулювали розповсюдження шкідливого програмного забезпечення для мобільних пристроїв, особливо на таких популярних платформах, як Android. У світлі цього виникає нагальна потреба у розробці ефективних рішень по захисту користувача від всіх видів зовнішніх загроз. Однак захисна здатність значною мірою ускладнена обмеженим розумінням нових мобільних шкідливих програм та відсутністю своєчасного доступу до відповідних додатків. Динаміка розповсюдження шкідливих мобільних додатків операційної системи Android [1] свідчить про неймовірний рівень погіршення стану як інформаційної, так і функціональної безпеки користувачів мобільних пристроїв.

Постановка проблеми. Для ідентифікації та пошуку шкідливих додатків традиційно використовуються два основних способи моніторингу – статичний та динамічний. Після виходу шостої версії операційної системи Android, яка значно удосконалила систему запитів дозволів, значно ускладнився статичний аналіз додатку на етапі інсталяції. Це сприяло посиленню подальшого розвитку динамічних систем аналізу мобільних додатків. На сьогоднішній день існує багато спроб систематично охарактеризувати шкідливі додатки з різних аспектів [2; 3], включаючи способи їх встановлення, механізми активації, а також характеристики та здатність обійти виявлення з боку існуючого мобільного антивірусного програмного забезпечення. Саме остання властивість вимагає необхідності кращої розробки мобільних анти-зловмисних програмних комплексів наступного покоління. Кожен мобільних додаток взаємодіє з ОС за допомогою виклику API функцій SDK. Це дає можливість відслідкувати його дію за допомогою системи динамічного функціонального трасування. Отже, автоматична побудови сигнатури функціонального ланцюжка додатку (СФЛД) викликів API може допомогти заздалегідь визначити й повідомити користувача або розробника про потенційно-небезпечну поведінку мобільних додатків.

Аналіз останніх досліджень і публікацій. Проведений аналіз показав, що проблеми безпеки ОС Android пов'язані насамперед із недосконалою системою привілеїв, яка має зручний та ефективний механізм відображення дозволів при встановленні користувачами нових додатків [4], але все ж таки залишає зловмисникам доступ до функціональних вузлів та чутливих даних. Значна кількість досліджень була присвячена вивченню способів виявлення шкідливих програм перед їх встановленням. Наприклад, TaintDroid, DroidRanger, DroidScope [5] можуть контролювати поведінку програм під час виконання. Водночас такі системи, як Kirin [6], ідентифікують шкідливе програмне забезпечення за допомогою статичного аналізу. Обидва методи мають переваги і недоліки. Наприклад, динамічна ідентифікація шкідливого додатку створює навантаження для ресурсів мобільного пристрою, і не завжди може бути застосована, якщо продуктивність є критичною. Так, програмне забезпечення для статичного аналізу не спричиняє таких додаткових витрат під час вико-

нання, але воно в основному побудовано на створених шаблонних алгоритмах. Іншим рішенням є система виявлення DroidRanger, яка базується на двоступеневому аналізі з метою виявлення шкідливого програмного забезпечення, як нульового дня, так і добре відомого. Існує також сучасний підхід, який використовує метод нейронної проекції для категоризації характеристик сімейств шкідливих програм [9]. Здебільшого цей метод застосовується не до однієї програми, а до сімейства додатків.

Виділення недосліджених частин загальної проблеми. Традиційно найновіші дослідження у сфері захисту від шкідливого програмного забезпечення зосереджені на прикладному рівні, таких як віруси, перехресні атаки тощо. Майже всі загрози такого роду можуть бути виправлені, за допомогою оновлення ОС або патчів для конкретного програмного додатку. Але такій підхід залишає час для зловмисників між виявленням типу атаки та фактичним завантаження оновлень на фізичний пристрій. За допомогою аналізу побудованої сигнатури ланцюжка функціональних API викликів можливо заздалегідь попередити користувача а також, у разі потреби, блокувати виконання шкідливого додатку. З погляду на це дуже перспективним є підхід до порівняння двох послідовностей API викликів за допомогою методів, які використовуються в галузі обчислювальної біології – біоінформатиці. Одна зі сфер цієї галузі лежить у площині застосування алгоритмів для аналізу великих структур білків та вирівнювання послідовностей з метою виявлення ступеню ідентичності та схожості.

Метою статті є дослідження розробленого підходу до покращення модулю системи функціональної безпеки ОС Android шляхом використання досягнень біоінформатики для порівняння послідовностей API викликів для виявлення схожості та ідентичності шкідливих програм та подальшої нотифікації користувача про можливі загрози або зупинки їх виконання таких додатків.

Виклад основного матеріалу.

1. Побудова псевдосимволу СФЛД

СФЛД являє собою набір функцій, які послідовно використовує програмний додаток під час комунікації з SDK Android. Строкове порівняння двох послідовностей СФЛД є доволі ресурсномістке завдання в реляційній базі даних, яке можна значно спростити якщо замінити API виклик на псевдосимвол для пошуку співпадіння.

Функція API виклику являє собою строкову константу, переважно від 30 до 100 символів, яка складається з повної назви пакета, назви класу та самої функції SDK Android (рис. 1).

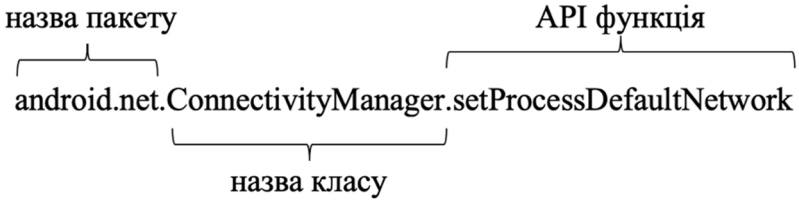

Рис. 1. Приклад API функції встановлення мережі за замовченням

Для того щоб програмний додаток міг використовувати будь-яку функцію ресурсів операційної системи, йому необхідні дозволи (android permissions) які користувач надає під час інсталяції або виконання додатку. У свою чергу, кожен дозвіл належить до своєї групи дозволів операційної системи Android. Отже, кожну функцію можна закодувати у вигляді унікальної трьох сегментної послідовності яка складається з: номера групи, номера дозволу, який належить цій групі, та номера функції всередині цього дозволу.

Таким чином, повну вхідну СФЛД можна представити в вигляді набору псевдосимволів, які значно швидше порівнювати при використанні будь-якого алгоритму вирівнювання послідовностей, а також можна легко трансформувати в первинну форму (рис. 2).

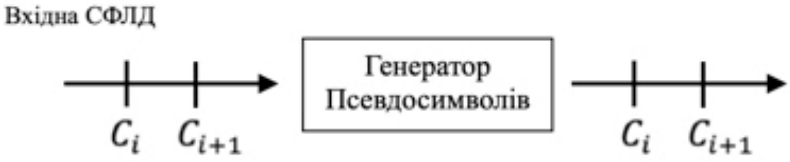

Рис. 2. Схема генерації псевдосимволу СФЛД

Детально розглянути трансформацію вхідної функції можна на прикладі функції android.net.ConnectivityManager.setProcessDefaultNetwork. Побудова псевдосимволу починається в зворотному порядку, тобто спочатку знаходиться індекс функції в таблиці функцій SDK. Далі знаходиться позиція дозволу зі списку android.permissions в який входить дана функція, та група, до якої входить знайдений дозвіл. У результаті вхідна функція буде мати вигляд G5P9C12 (рис. 3).

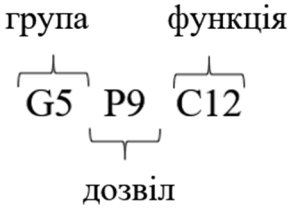

Рис. 3. Детальний опис псевдосимволу API функції SDK Android

Таким чином, будь яка вхідна послідовність буде спочатку трансформуватися в псевдосимвол для подальшої обробки з метою пошуку збігу. В наведеному прикладі початкова строкова репрезентація має 56 символів а трансформована послідовність містить лише 6 символів. Такий підхід не тільки покращить продуктивність алгоритму порівняння СФЛД а також буде використаний для нотифікації користувача, тому що такий запис одразу дає можливість зрозуміти який це тип загрози.

2. Побудова СФЛД на базі проекту “Malgenome”

Для побудови бази шаблонних СФЛД у межах цієї роботи було використано набір ланцюжків на базі дослідницького проєкту Malgenome [7; 8]. Проєкт являє собою набір метаінформації представленої у файлі формату .csv (malgenome_dataset.csv). Кожен стовпчик в файлі являє собою довгу комбінацію API викликів та запитів на дозволи від операційної системи Android. Рядки описують наявність виклику в бінарному форматі «0» або «1». Приклад фрагмента шкідливого додатку показано на рис. 4.

	READ_PHONE_STATE	getBinder	ClassLoader	Landroid.content.Context.registerReceiver	Ljava.lang.Class.getField	Landroid.content.Context.unregisterReceiver
2	1	0	0	1	0	1

Рис. 4. Фрагмент сигнатури функціонального ланцюжку з проекту Malgenome

Перший етап в побудові бази СЛФД полягає в обробці всіх ланцюжків із проєкту Malgenome шляхом зчитування та подальшої фільтрації виключно функціональних викликів Android SDK. Таким чином, після зчитування, отримаємо ланцюжок API викликів, який разом з метаданими можна зберегти у базі даних за структурою, аналогічній, що наведена в таблиці.

Таблиця. API виклики та призначені їм псевдосимволи

Номер послідовності	API виклик	Псевдо символ
1	FileInputStream.getChannel	A
2	FileChannel.map	B
3	File.delete	C
4	FileInputStream.close	D
5	Ljava.lang.Class.getField	E
6	File.exists	F
7	AlertDialog.Builder	G
8	alertDialog.setCancelable	H
9	android.os.IBinder.bindService	I

Слід зазначити, що фізична доступність apk файлу певної програми додає додаткового потенціалу для використання різних способів реверсивного інжинірингу при вивченні особливостей саме цього представника сімейства шкідливих додатків. Такий підхід дозволяє дещо розширити спектр евристичного аналізу для подальшого удосконалення методу виявлення потенційно шкідливих програм.

3. Порівняння СФЛД

Сама СФЛД окремо не має в собі корисної інформації без порівняння її з іншими елементами, про які наявна якась початкова інформація. СФЛД з проєкту Malgenome вже класифікувались як шкідливі програмні додатки користувачами або дослідниками в галузі інформаційної та функціональної безпеки мобільних пристроїв. Таким чином, задачу порівняння двох СФЛД можна умовно звести до послідовного порівняння двох строкових послідовностей з метою пошуку повного збігу або ступеня схожості. На перший погляд задача виглядає тривіальною, але при детальнішому розгляді стає зрозуміло, що простий перебір дуже швидко перестає бути практично доцільним.

Розглянемо простий приклад пошуку, якщо база СФЛД складається з m послідовностей розміром n та цільовим розміром порівнюваного фрагмента $d(d \geq n)$. Згідно із заданими параметрами, може бути розроблений алгоритм „грубої сили”, який буде:

1) перераховувати кожну підпослідовність розміру d з кожної послідовності m;
2) нумерувати та зберігати кожну підпослідовність розміру d від кожної послідовності m;
3) порівнювати кожен залишок ланцюжка першої підпослідовності, починаючи з позиції збігу з кожним API викликом у тому самому положенні другої підпослідовності.

Результатом роботи даного алгоритму є набір підпослідовностей, які відповідають лише підпослідовностям з однієї шаблонної вибірки.

Можна оцінити складність даного алгоритму. Крок номер 1 буде виконаний $\mathrm{m} \times (\mathrm{n} - \mathrm{d})$ разів, або $\mathrm{O}(\mathrm{m} \times \mathrm{n})$, крок 2 буде виконаний $(\mathrm{m} - 1) \times (\mathrm{n} - \mathrm{d})$ разів, або $\mathrm{O}(\mathrm{m} \times \mathrm{n})$ разів, крок 3 буде включати в гіршому випадку d порівнянь, тож $\mathrm{O}(\mathrm{d})$. Таким чином, обчислювальна складність алгоритму «грубої сили» становить:

$$O(m^2 \times n^2 \times d).$$

Навіть для досить невеликого набору СФЛД, де $n \approx 10^2, m \approx 10^1$ і $d \approx 10^1$ (тобто в простому випадку 10 послідовних API викликів), алгоритм вже включає 10^7 операцій, що створює дуже серйозні проблеми навіть для сучасних апаратних конфігурацій. Треба також зауважити, що під час виконання алгоритму «грубої сили», припускаючи гіпотетично оптимальний сценарій пошуку, кожен API виклик кожної послідовності у шаблонному наборі доведеться проходити принаймні один раз, тобто складність оптимального алгоритму становить принаймні $O(m \times n)$ операцій. Очевидно, що алгоритми, які ґрунтується на простому послідовному переборі, погано масштабуються та будуть чутливими до збільшення масиву оброблюваних даних [9].

Біоінформатика займається вирішенням подібних задач, а саме порівнянням послідовностей структур білків на предмет схожості або ідентичності. Оцінка подібності між біологічними послідовностями є одним із основних засобів, за допомогою якого біоінформатика сприяє розумінню біології.

Вирівнювання послідовностей – це процес порівняння різних послідовностей шляхом пошуку серії окремих символів або шаблонів символів, що мають однакове розташування в обох послідовностях. Взагалі існує три основних типи вирівнювання послідовностей: попарне вирівнювання послідовностей, багаторівневе вирівнювання послідовностей та структурне вирівнювання послідовностей. Вирівнювання парних послідовностей одночасно може використовуватися лише між двома послідовностями. Вирівнювання декількох послідовностей – це продовження попарного вирівнювання, що включає більше двох послідовностей одночасно. Структурне вирівнювання послідовностей аналізує всю структуру білкової ланцюга, на відміну від попарного та багаторівневого вирівнювання послідовностей і здебільшого візуалізується тривимірно.

Попарне вирівнювання послідовностей, у свою чергу, може бути локальним або глобальним. У той час як локальні вирівнювання знаходять найкращий збіг між двома послідовностями, глобальні вирівнювання знаходять найкращий збіг за загальною довжиною різних послідовностей, що беруть участь у вирівнюванні. Серед методів, що забезпечують попарне вимірювання найбільшу популярність отримали метод глобального вирівнювання, який базується на алгоритмі Нідлмана-Вунша, та метод локального вимірювання на основі алгоритму Сміта-Вотермана [10]. Останній, у свою чергу, є нічим іншим, як модифікацією алгоритма Нідлмена-Вунша. Незважаючи на той факт, що методи, представлені вище, були результатом еволюції різних наук, таких як інформатики, біоінформатики та інших, їх застосування не обмежується лише цими галузями. Нині вони знаходять своє застосування в ідентифікації, оцінці, розумінні та прогнозуванні змін білка, системній біології, обчислювальній еволюційній біології та інших. Найпоширенішими біоінформативними інструментами для досягнення цієї мети є базовий інструмент локального пошуку вирівнювання BLAST, та швидке вирівнювання FASTA [11], які проводять порівняння між парами послідовностей на основі регіонів локальної подібності.

4. Комбінування методів вирівнювання СФЛД

Після побудови бази шаблонних шкідливих додатків та їх метаданих в першу чергу виникає проблема продуктивного пошуку по достатньо великому масиву даних. Проблему продуктивності системи потрібно розглядати комплексно використовуючи як провідні алгоритми пошуку в реляційних базах даних так і ефективні алгоритми пошуку фрагментованого співпадіння послідовностей. На сьогодні існує багато способів підвищити продуктивність пошуку строкового рядку в базі даних як наприклад Full Text Search (FTS) [12]. У межах цієї роботи було використано індексування стовпчика з функцією API виклику а також з псевдосимволом відповідного виклику, що погіршило продуктивність при побудові шаблонної бази даних, але дасть значне покращення по строковому пошуку вже після вставки (якщо проаналізувати частоту запису та зчитування інформації з шаблонної бази даних то очевидно що кількість зчитування значно перевищить кількість запису, тому ми можемо знехтувати втратами продуктивності при додаванні нової інформації).

Метод перебору разом з алгоритмом глобального вирівнювання послідовностей Нідлмана-Вунша працює достатньо швидко коли кількість порівнюваних послідовностей не велика, що не відповідає дійсності в нашому випадку. База даних складається з початкової бази шаблонних послідовностей:

$$СФЛД_{initial} = \{C_1, C_2, C_3 \dots C_n\},$$

де C_i – послідовність API викликів; n – кількість ланцюжків у базі даних.

Зрозуміло, що розмір бази даних з часом буде тільки збільшуватись, у такому випадку продуктивність виконання буде погіршуватись лінійно зі збільшенням розміру бази СФЛД.

Метод глобального вирівнювання послідовностей Нідлмана–Вунша вирівнює послідовності по всій довжині ланцюжка та дає повну картину співпадіння, але використання цього методу під час отримання першої послідовності, яка буде значно менша шаблонної СФЛД, потребує, по-перше, значних обчислювальних потужностей, а по-друге, є вірогідність пропустити схожість фрагментованної послідовності на якійсь ділянці шаблонної СФЛД, що у свою чергу призведе до помилки першого роду. Тому на першому етапі (ініціалізаційному) необхідне використання більш чутливого алгоритму пошуку співпадіння послідовностей, а саме використовувати алгоритм локального вирівнювання Сміта-Ватермана, для того щоб побудувати підмножину всіх СФЛД, яка містить спільні ділянки з фрагметованою послідовністю. У процесі ініціалізаційного пошуку також необхідно зберегти набір метаданих, які будуть використані в подальшій агрегації ланцюжків:

- фрагмент клієнтської СФЛД, який використовувався в локальному пошуку;
- початкову та кінцеву позицію співпадіння в шаблонному СФЛД;
- строковий токен, який унікально ідентифікує клієнтській мобільний пристрій.

Після збереження клієнтської послідовності та всіх метаданих на сервері кожен наступний запит з таким самим токеном буде оброблений з урахуванням попередньо збережених метаданих. Оскільки клієнтський програмний додаток знімає сигнатури викликів на протязі якогось часового інтервалу Q, то послідовність функцій API викликів майже завжди буде фрагментована (рис. 5), а отже, кожен клієнтський запит, у контексті сервера, повинен мати стан, який допоміг би ідентифікувати та аналізувати декілька послідовних запитів, що містять фрагментовані СФЛД від того ж самого клієнта.

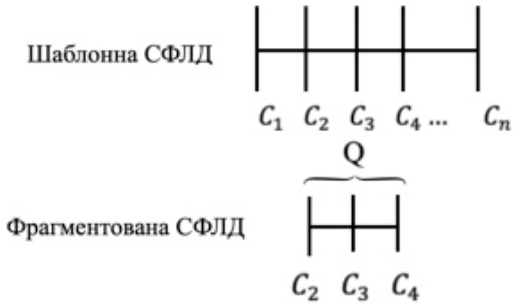

Рис. 5. Вхідна фрагментована СФЛД в порівнянні з шаблонною СФЛД

Блок-схема алгоритму обробки вхідної послідовності СФЛД представлена на рис. 6.

У контексті СФЛД викликів ключову роль відіграє саме послідовність API викликів, тобто при повторному зверненні клієнтського програмного додатку з таким самим токеном потрібно зробити конкатенацію фрагментованих послідовностей:

$$C_t = C_1 + C_2 + \cdots + C_n$$

де C_t – агрегація фрагментованих послідовностей клієнтського мобільного пристрою з токеном t, n – кількість запитів.

Ідеальний показник порівняння СФЛД (показник ідентичності) повинен бути добре обґрунтованими та зрозумілими, мати одне «підсумкове» число у фіксованому діапазоні - наприклад, від 0% (повністю не збігаються) до 100 % (повністю збігаються), так і вторинний показник подібності, який буде показувати можливе відхилення фрагментованого ланцюжку від шаблонної послідовності. Можливі мінімальні зміни в структурах СФЛД не повинні призводити до значних стрибків в розрахункових значеннях міри подібності. Показник повинен фіксувати подібність або відмінності між структурами СФЛД на будь-якому заданому рівні точності або подібності.

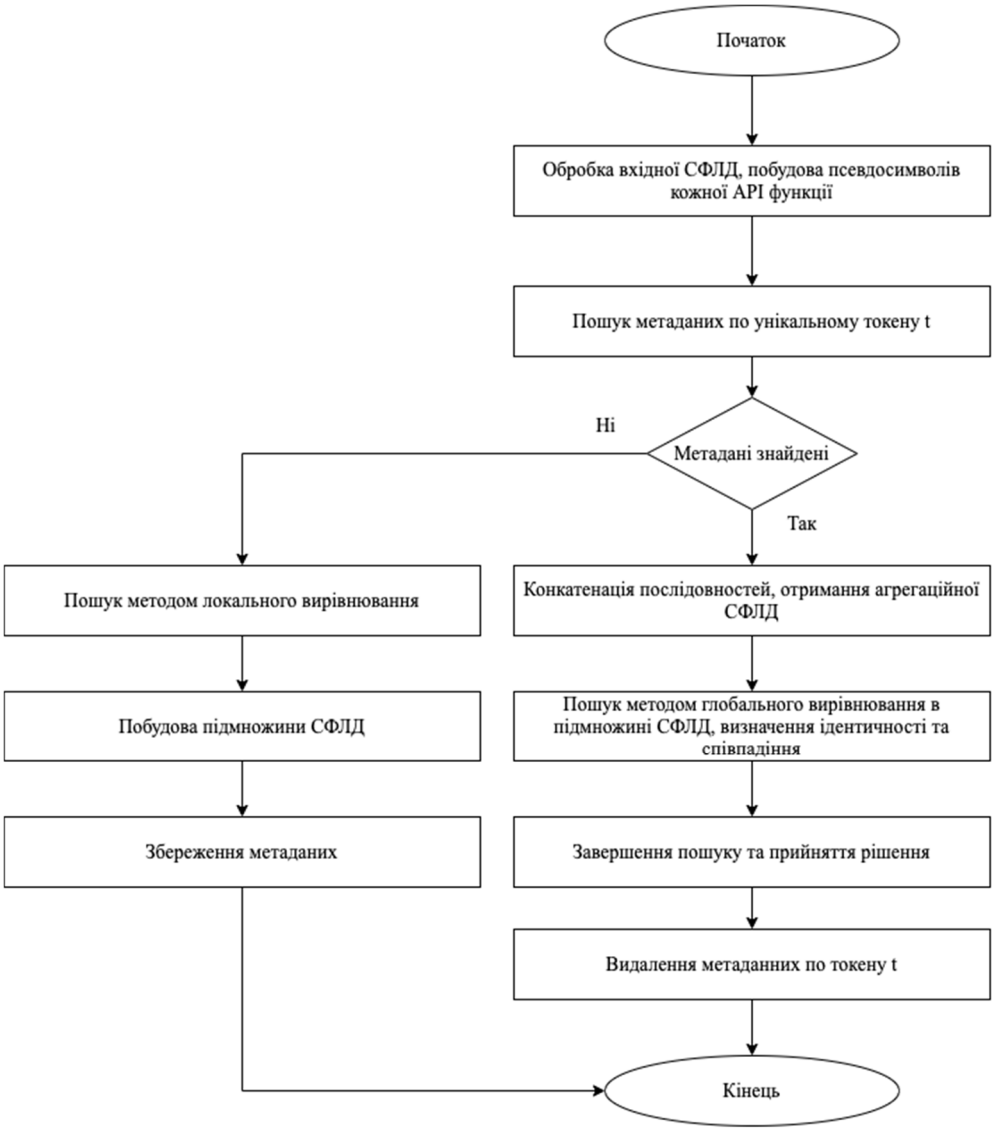

Рис. 6. Блок-схема ітеративного алгоритму пошуку співпадіння шкідливих СФЛД

Ідентичність послідовності - це кількість символів, які точно співпадають між двома різними послідовностями. Пропуски не враховуються та вимірювання стосується коротшої з двох послідовностей. Це призводить до того, що ідентичність послідовності не є транзитивною, тобто якщо послідовність A = B і B = C, то A не обов'язково дорівнює C в тому випадку, якщо в усіх трьох порівняннях використовувалося одна і та сама формула обрахунку якості порівняння. Для того щоб вирахувати ідентичність двох послідовностей, використовується формула:

$$\frac{N}{\min\left(length(C1), length(C2)\right)},$$

де N – кількість співпадінь яка ділиться на мінімальну довжину ланцюжків C_1 та C_2.

У такому випадку, якщо ідентичність пари (C_1, C_2) = 100%, ідентичність (C_2, C_3) = 100%, то можлива ситуація коли ідентичність (C_1, C_3) = 85%. Отже, 100% ідентичність не означає, що дві послідовності однакові. Знаходження подібності послідовностей надає можливість ефективного узгодження двох або більше послідовностей та забезпечує мінімальну кількість операцій редагування (вставки, видалення та заміни) шляхом перетворення однієї послідовності в точну копію іншої послідовності, що вирівнюється. Наприклад, для трьох порівнюваних послідовностей:

C1: AAGGCTT
C2: AAGGC
C3: AAGGCAT

може використовувалося одна і та сама формула обрахунку якості порівняння. Використовуючи такий підхід, відсоткова подібність послідовностей C_1, C_2, C_3 для наведеного прикладу становить: подібність (C_1, C_2) = 60 %, подібність (C_2, C_3) = 60 %, подібність (C_1, C_3) = 86 %.

Висновки. ОС Android вимагає постійних удосконалень та оновлень з погляду безпеки насамперед завдяки високому охопленню користувачів та зростаючій популярності. Системи динамічного аналізу демонструють високу ефективність та переваги у захисті операційної системи від сучасного шкідливого програмного забезпечення шляхом постійного аналізу взаємодії програми із середовищем ОС. Комбінування існуючих підходів біоінформатики до аналізу та порівняння послідовностей є ефективним засобом вирішення задачі пошуку шкідливих додатків.

Список використаних джерел

1. Buthaina Mohammed AL-Zadjali. A critical evaluation of vulnerabilities in Android OS (Forensic Approach). : In International Journal of Computer Applications (0975 – 8887). Volume 130, No.5, 2015. P. 38-42.
2. Lasheras F., Comminello D., Krzemien A. Advances in complex systems and their applications to cybersecurity. Complexity, Hindawi, 2019. P. 1-2.
3. Allen G. Android Security and Permissions.Beginning Android, 2015. P.343–354.
4. Elenkov N. Android Security Internals: An In-Depth Guide to Android's Security Architecture. No Starch Press; 1st edition, 2014. 401 p.
5. Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang. PScout: Analyzing the Android Permission Specification. In: CCS '12: Proceedings of the 2012 ACM conference on Computer and communications security, 2012. P. 217–228.
6. Arp D., Spreitzenbarth M., Hubner M., Gascon H., Rieck K. Drebin: Effective and explainable detection of android malware in your pocket. In: Proc. of Annual Symposium on Network and Distributed System Security (NDSS). The Internet Society, 2014. P. 23–26.
7. Yajin Zhou, Xuxian Jiang. Android Malware Genome Project. URL: - http://www.malgenomeproject.org/.
8. Suleiman Yerima. Android malware dataset for machine learning 1. URL: - https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_1/5854590/1.
9. Zahariev M, Dahl V., Chen W. Efficient algorithms for the discovery of DNA oligonucleotide barcodes from sequence databases. Molecular Ecology Resources, vol. 9, issues 1, Special Issue: Special Issue on Barcoding Life, 2009. P. 58-64.
10. Jones, N., Pevzner, P.: An introduction to bioinformatics algorithms. Massachusetts Institute of Technology. A Bradford book, 2004. – 435 p.
11. Eric S. Donkor, Nicholas T. K. D. Dayie, Theophilus K Adiku. Bioinformatics with basic local alignment search tool (BLAST) and fast alignment (FASTA). Journal of Bioinformatics and Sequence Analysis, vol. 6(1), 2014. P. 1-6.
12. Hilary Cotter. Search me: using SQL Server full-text search, 2007. URL: https://cdn.ttgtmedia.com/searchSQLServer/downloads/insider_fulltextsearch_0121.pdf.

References

1. Buthaina Mohammed AL-Zadjali. A critical evaluation of vulnerabilities in Android OS (Forensic Approach). : In International Journal of Computer Applications (0975 – 8887). Volume 130, No.5, 2015. P. 38-42.
2. Lasheras F., Comminello D., Krzemien A. Advances in complex systems and their applications to cybersecurity. Complexity, Hindawi, 2019. P. 1-2.
3. Allen G. Android Security and Permissions.Beginning Android, 2015. P.343–354.
4. Elenkov N. Android Security Internals: An In-Depth Guide to Android's Security Architecture. No Starch Press; 1st edition, 2014. 401 p.

5. Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang. PScout: Analyzing the Android Permission Specification. In: CCS '12: Proceedings of the 2012 ACM conference on Computer and communications security, 2012. P. 217–228.

6. Arp D., Spreitzenbarth M., Hubner M., Gascon H., Rieck K. Drebin: Effective and explainable detection of android malware in your pocket. In: Proc. of Annual Symposium on Network and Distributed System Security (NDSS). The Internet Society, 2014. P. 23–26.

7. Yajin Zhou, Xuxian Jiang. Android Malware Genome Project. URL: - http://www.malgenomeproject.org/.

8. Suleiman Yerima. Android malware dataset for machine learning 1. URL: - https://figshare.com/articles/dataset/Android_malware_dataset_for_machine_learning_1/5854590/1.

9. Zahariev M, Dahl V., Chen W. Efficient algorithms for the discovery of DNA oligonucleotide barcodes from sequence databases. Molecular Ecology Resources, vol. 9, issues 1, Special Issue: Special Issue on Barcoding Life, 2009. P. 58-64.

10. Jones, N., Pevzner, P.: An introduction to bioinformatics algorithms. Massachusetts Institute of Technology. A Bradford book, 2004. – 435 p.

11. Eric S. Donkor, Nicholas T. K. D. Dayie, Theophilus K Adiku. Bioinformatics with basic local alignment search tool (BLAST) and fast alignment (FASTA). Journal of Bioinformatics and Sequence Analysis, vol. 6(1), 2014. P. 1-6.

12. Hilary Cotter. Search me: using SQL Server full-text search, 2007. URL: - https://cdn.ttgtmedia.com/searchSQLServer/downloads/insider_fulltextsearch_0121.pdf.

UDC 528.481

Ihor Karpachev, Volodymyr Kazymyr

ANDROID MALWARE DETECTION BY FUNCTIONAL CHAIN SIGNATURE

Mobile devices are at the epicenter of modern growing demand of people staying interconnected and being able to solve their everyday tasks online. That is why, mobile devices captured more than half of the market of computing and communication systems, which led to extreme urgency of general protection of mobile applications, user data and functional security.

Most of existing security measures provided by OS Android are based on preventive actions and system restrictions in order to ensure platform security in general. Considering significant delay in resolving security issues and potential risk to lose private data or even disruption in functional safety of the system there is a necessity of an extra safety unit on top of existing general security system, which will help to notify user of potentially malicious software at the executing stage in runtime.

Development of a method for ensuring the functional safety of Android mobile device is a complex task. Every android application generates digital trace when using Android SDK in order to communicate with Android OS resources. Application execution signature chain (AESC) is a sequence of functions called from android application to Android OS via SDK. The aim of AESC is to model the chain of API calls and match it with patterns during malware detection.

The Niedlmann – Wunsch global alignment method aligns sequences along the entire length of the chain and gives a complete picture of the match, but using this method to obtain the first sequence, which will be much smaller than the template AESC, requires significant computing power first, and the second is the probability skip the similarity of the fragmented sequence in any part of the template AESC, which in turn will lead to an error of the first kind. Therefore, at the first stage (initialization) it is necessary to use a more sensitive sequence matching algorithm, namely to use the Smith-Waterman local alignment algorithm to build a subset of all AESC, which contains common areas with a fragmented sequence. Combining existing approaches to bioinformatics analysis is an effective means of solving the problem of finding malicious applications.

***Keywords**: AESC; Android OS; API function; sequence alignment.*

Fig.: 6. References.: 12.

Карпачев Ігор Ігорович – аспірант кафедри інформаційних та комп'ютерних систем, Національний університет «Чернігівська політехніка» (вул. Шевченка, 95, м. Чернігів, 14035, Україна).
Karpachev Igor – PhD student, Department of Informational and Computer Systems, Chernihiv Polytechnic National University (95 Shevchenka Str., 14035 Chernihiv, Ukraine).
E-mail: benchakalaka@gmail.com
ORCID: https://orcid.org/0000-0003-1910-3264
ResearcherID: R-3626-2016

Казимир Володимир Вікторович – доктор технічних наук, професор, Національний університет «Чернігівська політехніка» (вул. Шевченка, 95, м. Чернігів, 14035, Україна).
Kazymyr Volodymyr – Doctor of Technical Sciences, Professor, Chernihiv Polytechnic National University (95 Shevchenka Str., 14035 Chernihiv, Ukraine).
E-mail: vvkazymyr@gmail.com
ORCID ID: orcid.org/0000-0001-8163-1119
Scopus Author ID: 56644727300

Карпачев І., Казимир В. Виявлення шкідливих додатків ОС Андроїд по сигнатурі функціонального ланцюжка. *Технічні науки та технології.* 2021. № 1(23). С. 109-117.