КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (23), 2024
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2024.23.213224
УДК 004.55

Сосновий Владислав Олексійович
Асистент кафедри комп'ютерної інженерії
Державний університет інформаційно-комунікаційних технологій, Київ, Україна
ORCID0000-0002-3217-4537
vladyslavsosnovyy@gmail.com

Лащевська Наталія Олександрівана
к.т.н., доцент, завідувач кафедри Комп'ютерної інженерії
Державний університет інформаційно-телекомунікаційних технологій, Київ Україна
ORCID 0000-0003-2148-115X
lastchevska@gmail.com

ВИЯВЛЕННЯ ШКІДЛИВОЇ ДІЯЛЬНОСТІ З ВИКОРИСТАННЯМ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ БЕЗПЕРЕРВНОЇ РОБОТИ

Анотація. У цій статті описана проблема з виявленням шкідливих програм в запущених системах користувачів мобільних додатків. Оскільки користувачі можуть завантажити будь-який додаток на свій телефон, який з часом може підтягнути додаткові налаштування, в яких можуть зберігатися шкідливі підпрограми моніторингу як за особистим життям, так і за їх особистими даними такого типу як логіни, паролі, банківські дані. Виявлення таких підпрограм базується на динамічному аналізі та формулюється як слабко контрольована проблема. Стаття містить аналіз інформації про розробки дослідників, які займались моделями та методами виявлення такими як: статистичні та динамічні методи виявлення вторгнень, модель виявлення аномалій, методи класифікації налаштувань, методи з застосування машинного та глибинного навчання. Машинне навчання і особливо глибинне навчання стали надзвичайно корисною та цікавою темою в кібербезпеці за останні кілька років. У цьому контексті виявлення зловмисного програмного забезпечення приділяло значну увагу. В статті розглядається проблема виявлення активності зловмисного програмного забезпечення мобільних операційних систем в часовій області шляхом аналізу поведінкових послідовностей великої кількості промислових даних. Коли зловмисне програмне забезпечення виконується в системі, його поведінка складається з ряду різних дій, розміщених уздовж осі часу, та існує лише підпослідовність дій, які призводять до зловмисної діяльності. Дуже часто зловмисне програмне забезпечення не виказує себе відразу, і в певний момент виконання формується шкідлива активність. Отже, основна задача і складність полягає в тому, щоб ідентифікувати таку підпослідовність у всій послідовності подій. Завдяки цьому пропонується розробити модель поведінки, яка б аналізувала динамічну поведінку програми в системі під час виконання. Для цього використовується послідовність викликів API/функцій, згенерованих програмою під час виконання, як вхідні дані та запропоновано архітектуру Рекурентної Нейронної Мережі (РНМ), яка дозволяє виявляти зловмисну активність. У статті описується метод навчання запропонованої моделі та наводиться перевірка її роботи на великій вибірці промислових даних, що складаються з великої кількості зразків, згенерованих на фермі емулятора. Багато постачальників мобільних телефонів прагнуть до апаратного прискорення на пристрої, щоб забезпечити кращу підтримку. Тому можна вважати, що розгортання моделі на основі РНМ безпосередньо на пристрої як одного з рівнів безпеки, може стати життєздатним рішенням. Дані тестування моделі, описані в статті, показують достатньо високі позитивні результати під час виявлення зловмисних активностей.

Ключові слова: рекурентна нейронна мережа; LSTM; машинне навчання; глибинне навчання, нейромережі; виявлення шкідливих програм.

ВСТУП

Збільшення використання смартфонів протягом останнього десятиліття супроводжується зростанням поширення шкідливого програмного забезпечення для мобільних пристроїв. Наприклад, згідно зі звітом G DATA [1] майже кожні десять секунд з'являється новий екземпляр шкідливого програмного забезпечення мобільних операційних систем (Android, iOS та ін.).

Автори зловмисного програмного забезпечення використовують багато методів, щоб уникнути виявлення, наприклад, шифрування, включаючи дозволи, які не потрібні програмі, запит небажаного обладнання, завантаження або оновлення, під час яких доброякісна програма оновлюється сама, деякі з них можуть обходити офлайн перевірки безпеки, наприклад, покладаючись на так звані дроппери, які завантажують зловмисне корисне навантаження після активації. Усі такі методи часто ускладнюють ідентифікацію зловмисного програмного забезпечення за допомогою статичного аналізу (наприклад, використання дозволів, фільтри намірів, використання API, статичні графіки викликів).

Тому доцільно додати ще один рівень виявлення на основі постійного моніторингу працюючої системи та використання модемних інструментів машинного навчання для виявлення зловмисної активності безпосередньо на пристрої. В роботі запропоновано вирішення даної проблеми шляхом розробки поведінкової моделі, яка піддається надійному навчанню на великій кількості промислових даних. Вивчається поведінка програми під час виконання, де її можна розглядати як послідовність дій. Певна підпослідовність дій у певному порядку може бути відповідальною за зловмисну діяльність, наприклад, доступ до SMS-коду та подальший вхід у ваш банківський рахунок, тоді як та сама підпослідовність у іншому порядку може бути безпечною.

Постановка проблеми. Існує кілька наукових проблем, які необхідно вирішити: описати і зрозуміти особливості поведінки та представити часові параметри запущеного процесу програми, проектування послідовної моделі, яка може обробляти ці функції, навчання обраної моделі, і, нарешті, отримати достатню кількість експериментальних даних для навчання моделі.

Аналіз останніх досліджень і публікацій. Методи виявлення зловмисного програмного забезпечення зазвичай поділяються на дві категорії: статичні та динамічні. У разі статичного виявлення інформація витягується з двійкового файлу без запуску програми. Згадавши лише деякі з них, що стосуються мобільних ОС, можна почати зі створення унікальних підписів [2], проаналізувати запитувані дозволи, як у [3], виконати більш детальний аналіз використаних викликів API, або навіть змоделювати послідовність викликів API, як у MaMaDroid [4].

Модель виявлення аномалій [5] постійно відстежує різні характеристики стану пристрою, такі як рівень заряду батареї, використання ЦП, мережевий трафік тощо. Вимірювання виконуються під час роботи, а потім передаються алгоритму, який класифікує їх відповідно. CrowDroid і AntiMalDroid — це два різні інструменти на основі аномалій, які використовуються для виявлення зловмисного програмного забезпечення на мобільних пристроях. Перший залежить від аналізу журналів системних викликів, тоді як другий аналізує поведінку програми, а потім генерує сигнатури поведінки зловмисного програмного забезпечення.

У цій роботі було проведено динамічний аналіз, коли додаток перевіряється за його поведінкою під час виконання. Робота з динамічним аналізом завжди означає вирішення проблеми того, що не всі події в одному записі можна віднести до зловмисної поведінки. Автори [6] перетворюють усі послідовності на вектори ознак за допомогою обчислення

відносних частот n-довгих підпослідовностей, зменшуючи розмірність за допомогою спеціального вибору ознак і класифікуючи за допомогою Support Vector Machines (SVM), тоді як [7] спочатку кластеризують підпослідовності на основі додаткової інформації про використання ЦП і пам'яті, а потім навчити класифікатор випадкового лісу на кожному кластері (використовуючи ту саму процедуру вибору функцій на основі n-грам).

Аналіз настроїв або аналіз думок — це обчислювальне дослідження думок, настроїв, емоцій, оцінок та інших поглядів користувачів. Техніка виявлення шкідливих програм, представлена в цій статті, має аналогію з аналізом настроїв тексту. Структура лежить у сфері класифікації настроїв на рівні документів, де документи є послідовністю подій мобільних ОС, за припущенням, що документи є думкою, тобто вони позначені експертом як позитивні чи негативні. Підходи до класифікації суб'єктивності та рівня речення не підходять, оскільки основною інформаційною одиницею є повна послідовність подій виконання однієї програми без локального маркування підпослідовностей.

Існуючі дослідження створили численні методи для різних завдань аналізу настроїв, які включають як контрольовані, так і неконтрольовані методи. У контрольованих умовах використовувалися SVM, Максимальна ентропія, Наївний Байєс тощо [8] з різними видами представлень функцій, як мішок уніграм і комбінацій ознак. До неконтрольованих методів належать різноманітні методи, які використовують лексику настроїв, граматичний аналіз і синтаксичні шаблони, тобто фіксовані синтаксичні фрази [9]. Глибинне навчання з'явилося як потужна техніка машинного навчання [10] і створило продуктивні результати в багатьох сферах застосування, починаючи від комп'ютерного зору [11], [12] розпізнавання[13] до Нейро-лінгвістичного Програмування (НЛП) [14]. Застосування глибинного навчання для аналізу настроїв також стало дуже популярним [15] серед дослідників.

Під час навчання надається послідовність подій, зібрана на емуляторі, і мітка цієї послідовності (зловмисне програмне забезпечення чи безпечне програмне забезпечення), передані з бази даних програм для мобільних ОС. У контексті аналізу настроїв кожну подію можна розглядати як слово, отже, вся послідовність утворює «текст». Мітка аналогічна настрою певного тексту, негативному чи позитивному. Кожну подію (або слово) можна перетворити на одноразово закодований розріджений вектор, а одноразовий розріджений вектор вкладається в низьковимірний щільний векторний простір, представлений матрицею вбудовування. Це вбудовування представляє значущий простір, який можна навчити, де дві семантично пов'язані події зберігаються близько одна до одної.

Рекурентна Нейронна Мережа (РНМ) [16] — це клас нейронних мереж, зв'язки між нейронами яких утворюють спрямований цикл. На відміну від нейронних мереж прямого зв'язку, РНМ може використовувати свою внутрішню «пам'ять» для обробки послідовності вхідних даних, що робить її важливою для обробки послідовної інформації. РНМ виконує те саме завдання для кожного елемента послідовності, причому кожен результат залежить від усіх попередніх обчислень, що схоже на «запам'ятовування» інформації про те, що було оброблено до цього моменту. Теоретично РНМ може використовувати інформацію в довільно довгих послідовностях, але на практиці стандартна РНМ обмежена через проблему зникаючого градієнта або розривного градієнта [17]. Дослідники розробили більш складні типи РНМ, щоб усунути недоліки стандартної моделі РНМ, наприклад, двонаправлену РНМ [18], глибинну двонаправлену РНМ або довгострокову короткочасну пам'ять (LSTM) [19].

LSTM — це особливий тип РНМ, який здатний вивчати довготривалі залежності. Звичайний блок LSTM складається з комірки, вхідного вентиля, вихідного вентиля та пропускного вентиля. Комірка відповідає за «запам'ятовування» значень протягом довільних інтервалів часу. Gated Recurrent Unit (GRU) можна розглядати як спрощену версію блоку LSTM. Він був введений у [20], де автори досягли семантично та синтаксично значущого представлення мовних фраз. Було показано, що GRU демонструють кращу продуктивність на менших наборах даних [21]. Варто зазначити, що в контексті зловмисного програмного забезпечення Windows [22] використовується архітектура РНМ, але лише для вбудовування, а не для остаточної класифікації.

Мета статті. Впровадити механізм спостерігача в ядрі мобільних ОС, який створює послідовність подій із запущеного процесу програми. Розробити Рекурентну Нейронну Мережеву архітектуру (РНМ), здатну обробляти великий потік подій. Розробити модель яка може спостерігати за діяльністю запущених програм, та зрештою, може заблокувати виконання зловмисних дії, перш ніж завдати шкоди користувачеві.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Коли програмний процес позначається як зловмисний, тоді призначаємо йому одну зловмисну мітку. Але не все, що робить процес, є шкідливим. Процес можна представити у вигляді послідовності або потоку подій $\vec{e} = (e_t)_{t=0}^{\infty}$. Зловмисна діяльність прихована в послідовності багатьох доброякісних подій. Мета полягає в тому, щоб навчити функцію передбачення $f(\cdot)$ і знайти порогове значення θ так, щоб

$$f(\overrightarrow{e_\tau}) \begin{cases} > \theta \\ \leq \theta \end{cases}, \text{ якщо } e_\tau \text{належить шкідливій підпослідовності ,та інші} \quad (1)$$

де $\overrightarrow{e_\tau} = (e_0, e_1, \dots, e_\tau)$ — послідовність подій від початку процесу до кроку часу τ. Функція висновку (1) представлена моделлю Рекурентної Нейронної Мережі (РНМ), яка живиться потоком подій із послідовності $\vec{e}$ одна за одною, і на кожному кроці часу функція (1) забезпечує висновок про поточну подію e_τ, що належить до шкідливої діяльності. Якщо це можна зробити точно, цікавим наслідком є те, що зловмисне програмне забезпечення може бути виявлено на початку зловмисної діяльності.

Оскільки для кожного потоку, створеного процесом, маємо лише двійкову мітку (зловмисне програмне забезпечення чи безпечне) для всього потоку, маємо справу зі слабко контрольованою проблемою, оскільки не знаємо, яка частина потоку належить до зловмисної діяльності.

Поставлене завдання можна розглядати як аналіз настроїв тексту (наприклад, відгуків користувачів) із домену NLP. Однією з моделей, що використовуються в аналізі настроїв, є багато-до-одного РНМ, де текст слово за словом подається в модель. Однак цей підхід не дуже підходить для поставленої проблеми, тому на наступному кроці модифікуємо цільову функцію таким чином, щоб зворотне поширення не було прив'язане до останнього вихідного вузла, що дозволяє моделі зосередити свою увагу на шкідливих підпослідовностях.

Далі, необхідно знайти поріг θ для досягнення оптимальної продуктивності моделі. Поріг θ буде встановлюватися динамічно під час процедури навчання відповідно до цільової (бажаної) продуктивності частоти помилкових позитивних результатів. Процедура встановлення порогу θ буде описана далі.

Щоб отримати функції від запущеного процесу, в ядрі мобільної ОС реалізовано серверний механізм спостерігання. Від запущеної програми механізм отримує особливості

поведінки (події), які використовуються моделлю для розрізнення шкідливих і безпечних програм.

Механізм спостереження змінив реалізацію C++ системи керування процесами мобільних ОС, щоб перехоплювати транзакції у двійковому форматі перед кожною транзакцією, що проходить через драйвер зв'язування ядра мобільної ОС. Шляхом реконструкції полів бінарного корисного навантаження та вилучення унікального цілочисельного ідентифікатора, що посилається на API (або виклик функції), шаблон і/або частота кожного виклику міжпроцесного зв'язку (IPC), надісланого в системі, підраховується та зберігається в пам'яті структури для обробки та аналізу. Хоча можливе майже 50000 унікальних викликів зв'язування, для оптимальної продуктивності механізм спостерігача відстежував лише підмножину з 1383 ідентифікаторів, найбільш релевантних запущеній програмі, вибраній на основі експертних знань.

Отже, загалом збирається 1383 окремих API/функціональних викликів, а потік цілих чисел від 0 до 1382 утворює послідовність подій e, зафіксованих протягом перших 60 секунд виконання програми. Детальний опис механізму спостерігача можна знайти в [34].

АРХІТЕКТУРА НЕЙРОННОЇ МЕРЕЖІ

A. Підхід до аналізу настроїв

Проблему аналізу настрою можна вирішити за допомогою багато-до-одного архітектури РНМ. Блок РНМ має свій внутрішній стан, відповідальний за «запам'ятовування» інформації, наданої попередніми вхідними даними. Оскільки мережа отримує один вхід за раз, на кожному кроці часу внутрішній стан оновлюється та може бути змінений. Потім прогноз робиться поверх виходу РНМ після отримання останнього введення. Ситуація схематично показана на рис. 1. Права частина ілюструє розгорнуте представлення графіка, зображеного ліворуч, від кроку часу T назад до кроку часу 0. На кожному кроці часу повторюваний блок A отримує поточну вхідну подію e_t і оновлює свій прихований стан h_t, виводить $\hat{y}_t$ і передає прихований стан h_t на наступний часовий крок. Рекурентний блок A містить блок РНМ (GRU або LSTM) і має спільні параметри. Детальніше про повторюваний блок A буде надано далі.

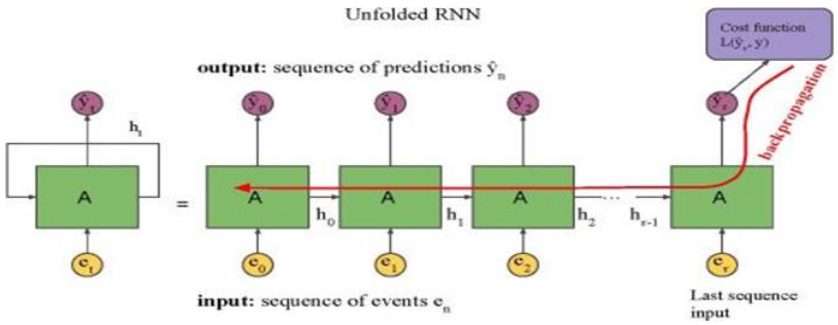

Рис. 1 Ілюстрація розгорнутої архітектури «багато до одного» та шляху зворотного поширення зі стандартною цільовою функцією, як визначено (3)

Спосіб розвитку стану разом із входом залежить від параметрів блоку РНМ і початкового стану блоку. Більш формально це можна записати у формі рекурентної функції:

$$h_t = f_\theta(h_{t-1}, \mathrm{e_t}), \tag{2}$$

де θ — параметри, які ми хочемо вивчити. Коли ми змінюємо параметри, рекурентний блок виконуватиме іншу поведінку. Параметри 0 навчаються за допомогою

ISSN 2663 - 4023

зворотного поширення через алгоритм часу. Для аналізу настроїв зворотне поширення виконується після отримання останнього вхідного сигналу e_T, отже, цільову функцію можна записати так:

$$L(\hat{y}, y) = \frac{1}{2}||\hat{y}(e_\tau) - y||L2, \quad (3)$$

де $\hat{y}$ — прогноз мережі, а y — ціль. Як показано на рис.1, зворотне поширення завжди починається в останньому вихідному вузлі після отримання останнього вхідного e_T.

B. Фаза навчання з максимальними втратами

Щойно описаний підхід можна застосувати для виявлення шкідливих дій. Однак проблема полягає в тому, що цільова функція (3) погано підходить для визначення зловмисної активності, як тільки вона відбувається. Підхід скоріше розроблений для випадку, коли крок висновку виконується після отримання останнього виходу. Ціллю статті є архітектура «один-до-одного», де для кожного входу мережа видає ймовірність того, що поточна подія e_t належить до зловмисної активності. Для навчання такої мережі необхідно мати цільову мітку для кожної вхідної події e_t. Така мітка відповідає, чи є поточний вхід частиною зловмисної підпослідовності. Однак на практиці збирати такі данні неможливо. Для кожної послідовності можемо мати лише слабку мітку, яка повідомляє, чи є зловмисна підпослідовність (чи більше підпослідовностей) у всій послідовності чи ні.

Важливо відзначити, що на етапі оцінки в системі для кожного входу по черзі модель виводить прогноз, і якщо прогноз перевищує поріг G, уся послідовність позначається як шкідливе програмне забезпечення незалежно від майбутніх входів. Цю слабко контрольовану проблему можна вирішити шляхом послаблення функції втрат (3) наступним чином.

$$L(\hat{y}, y) = \frac{1}{2}||\max_t(e_t) - y||L2, \quad (4)$$

де доданок $\max_t(e_t)$ представляє максимальний вихід рекурентної одиниці для всієї послідовності під час навчання.

Наслідком оптимізації (4) є те, що під час навчання зворотне поширення починається з вузла, де відбувся максимальний вихід рекурентної одиниці. Ситуація зображена на рис. 2. Для певної послідовності параметри мережі θ змінюються під час навчання. В кожному періоді максимум з'являється на іншому вузлі. Інтуїтивним наслідком формули (4) є те, що мережа змушена звертати увагу на зловмисні підпослідовності, як тільки вони з'являються.

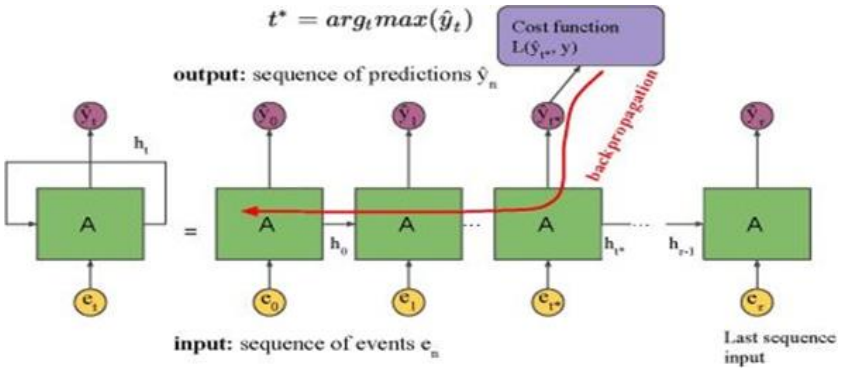

Рис. 2. Ілюстрація шляху зворотного поширення із запропонованою цільовою функцією, як визначено (4). Зворотне поширення починається у вихідному вузлі з міткою часу t^*

ISSN 2663 - 4023

Останньою проблемою, яку потрібно вирішити, є те, як встановити порогове значення G. Для механізму виявлення зловмисного програмного забезпечення в промислових додатках критично важливо досягти дуже низького коефіцієнта Хибнопозитивного Виявлення (ХПВ) при досягненні досить високого коефіцієнта Справжнього Позитивного Виявлення (СПВ). Компроміс між ХПВ і СПВ при різних порогових значеннях G може бути виражений кривою робочої характеристики приймача (ROC). Для ілюстрації на рис. 3 показана крива ROC за даними.

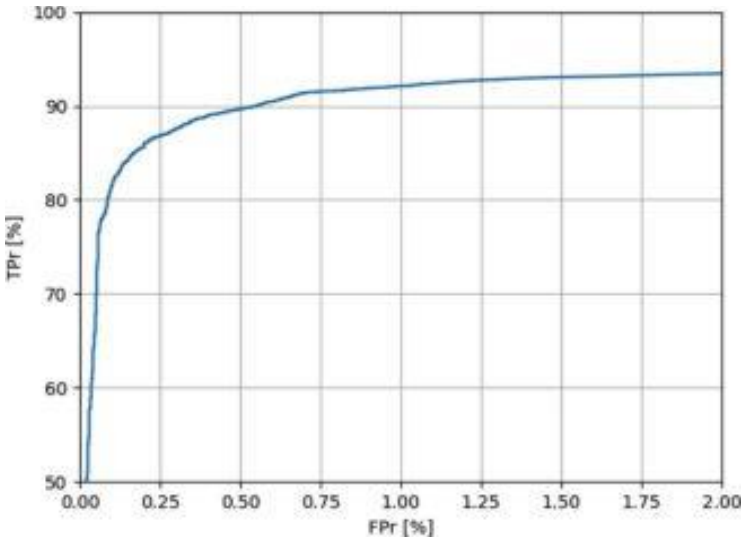

Рис. 3. Крива ROC з використанням моделі на основі даних AV-тесту

Проблема полягає в тому, як вибрати поріг G таким чином, щоб для невидимих даних у дикій природі ХПВ був близький до цільового ХПВ. Визначаємо цільовий ХПВ як бажану продуктивність ХПВ на невидимих даних. Однак розподіл невидимих даних відрізняється від даних, які бачить модель під час навчання. Тому його крива ROC відрізняється від кривої на даних навчання або даних перевірки. Якби була відома крива ROC для невидимих даних, можна було б вибрати порогове значення θ, що відповідає цільовій продуктивності ХПВ. На жаль, на практиці це не так. Щоб вирішити цю проблему, пропонується встановити порогове значення θ під час навчання на основі тестових даних для кожного періоду навчання.

Розглянемо однин період під час навчання. Після оновлення моделі для кожної партії тестових даних будуємо ROC і обчислюємо порогове значення θ таким чином, щоб ХПВ при використанні цього порогу відповідав цільовому ХПВ.

Припускається, що масив θs (запис усіх значень θ) створено нормальним розподілом.

Після оцінки всіх пакетів тестових даних обчислюємо середнє значення та стандартне відхилення записаних значень у масиві θs, а оптимальне порогове значення для поточної епохи обчислюється як:

$$\Theta^* = \mu_\Theta + k * \sqrt{\sigma_\Theta}, \quad (5)$$

де μ_Θ та $\sqrt{\sigma_\Theta}$ позначають середнє та стандартне відхилення значень у списку θs, а k — є гіперпараметром.

У сценарії встановлюємо k = 2, щоб охопити 93% порогової сукупності. Також відстежуємо середнє значення та дисперсію відповідного СПВ, щоб покращити політику ранньої зупинки.

Під час оцінювання навчена рекурентна модель разом із відповідним порогом 0, про який було сказано вище, розгортається на пристрої або серверній частині. Коли запускається невідома програма, модель живиться подіями, створеними механізмом спостереження, представленим вище. Після отримання вхідної події модель виводить висновок, і якщо значення більше 0, процес позначається як зловмисне програмне забезпечення, незалежно від наступні події.

РЕЗУЛЬТАТИ ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

A. Опис набору даних

Було взято 361 265 додатків мобільних ОС (Android та iOS) і для кожного згенерували послідовність поведінки на фермі емуляторів. Послідовності були призначені міткою та часовою міткою програми з промислової бази даних.

Таким чином, кожна програма в наборі даних представлена (i) цілочисельною послідовністю e, що представляє події, отримані механізмом спостереження, (ii) міткою, переданою з бази даних, і (iii) міткою часу Unix, що представляє, коли програма з'явилася вперше. Загалом набір даних містить 100 595 шкідливих програм і 260 670 доброякісних зразків, а в усіх послідовностях існує 1383 окремі події.

B. Архітектура та деталі навчання

Експерименти проводились з різними установками. Було використано різні розміри матриці вбудовування, одинарні або подвійні шари блоків LSTM або GRU, за якими слідував один-три повністю зв'язаних (FC) шари, а також спробували різні функції регулярізації (L2, LI, пакетна норма, випадання) і оптимізатори (ADAM, RMSprop, Імпульс Несторова).

Гіперпараметри: кожна вхідна подія e_t кодується в один гарячий вектор, який передається в матрицю вбудовування розміром 1384×16, оскільки в наборі даних існує 1383 відмінні функції, а один додатковий маркер використовується для доповнення. Кожна подія вбудовується в 16-вимірний щільний вектор, який передається в шар GRU розміром 256, за яким слідує шар FC розміром 128, після чого відбувається вилучення та об'єднується в один нейрон. Вихідні дані передаються сигмоїдній функції для отримання прогнозу.

Під час навчання використовується ймовірність збереження 0,7 для рівня випадання та нелінійності ReLU. Було помічено, що складання одиниць GRU або додавання додаткових шарів FC не покращує результати. Розмір партії 512 використовувався протягом усього експерименту, і найкращі результати були досягнуті з оптимізатором RMSProp зі швидкістю навчання 2,3-10-3. Швидкість навчання зменшується на коефіцієнт 0,6, якщо немає покращення втрати навчання протягом двох періодів.

Політика ранньої зупинки встановлюється таким чином, що навчання припиняється, якщо оцінений СПВ у наборі перевірки не покращується протягом 8 періодів. Зближення до найкращої моделі зазвичай досягалося за 30 періодів. Константа k у формулі (5) була встановлена рівною 2,0 для оцінки порогу моделі.

Розподіл даних: стандартна практика полягає в тому, щоб розділити набір даних на набори навчальних тестів шляхом випадкового вибору зразків. У випадку зловмисного програмного забезпечення досвід довів, що це не дуже добра стратегія. Цілком імовірно, що випадково вибраний навчальний набір може містити модифіковані копії того самого зловмисного програмного забезпечення, переупакованого як у тестовому наборі. Це призвело б до штучно хороша продуктивність, але класифікатор не зможе узагальнити нові зразки зловмисного програмного забезпечення.

Щоб запобігти цій проблемі, було розділено дані за часом, щоб найновіші зразки були в наборі для тестування, але не в наборі для навчання. Набір даних було розділено на дані навчання, перевірки та тестування наступним чином: усі послідовності подій були ранжовані відповідно до часових позначок і розділені у співвідношенні 90%/10%. Останній використовується як тестовий набір. Більший фрагмент далі випадковим чином перемішується, і 10% його використовується для перевірки, а решта для навчання.

Підводячи підсумок, 36.3fc найновіших зразків належить до тестового набору, 293.7k зразків використовуються для навчання моделі, а 32.6fc зразків використовуються для перевірки. Для вибраного розміру партії та обмежень пам'яті GPU кожна послідовність доповнюється або обрізається до довжини 1024, якщо необхідно.

Було проведено порівняння запропонованого методу з кількома сучасними як динамічними, так і статичними методами. Архітектура maxNet перевершує більшість результатів. Було значно перевершено інші методи за показником ХПВ (1,6%), що є критичним для промислового застосування. Крім того, продуктивність СПВ набагато краща порівняно з іншими показниками. Деякі експерименти вже показували подібні результати , але дослідження проводились з меншими наборами даних і повідомили про СПВ 97,3%, але отримали ХПВ 31,0%. А отже обраний метод в даній статті досяг найвищого показника Fl.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Зробивши ряд досліджень можна сформулювати виявлення зловмисного програмного забезпечення як слабко контрольовану проблему та розробляємо послідовну модель РНМ, здатну виявляти зловмисну активність у процесі, щойно вона відбувається. Було продемонстровано, навчання моделі шляхом релаксації цільової функції з багато-до-одного архітектури РНМ. У рамках цієї статті запропоновано набір даних, що складається з 361 зразка. Експерименти з цим набором даних демонструють ефективність 96,2% істинно позитивних результатів при 1,6% помилкових позитивних результатах, що перевершує сучасні результати.

В подальших роботах плануються нові дослідження з розробки та удосконалення моделей та методів для протидії зловмисним несанкціонованим проникненням в особисті гаджети користувачів та програмам моніторингу та реагування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lueg, C. (2017). *8,400 new Android malware samples every day.* https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android malware-samples-every-day
2. Feng, Y., et al. (2014). Apposcopy: Semantics-based detection of android malware through static analysis. *22nd ACM SIGSOFT International Symposium on Foundations of Software EngineeringNovember*, 576–587. https://doi.org/10.1145/2635868.2635869
3. Felt, A., et al. (2012). Android permissions demystified. *18th ACM conference on Computer and communications security*, 627–638. https://doi.org/10.1145/2046707.2046779
4. Mariconti, E., et al. (2016). MaMaDroid: Detecting android malware by building markov chains of behavioral models. https://doi.org/10.48550/arXiv.1711.07477
5. Shabtai, A., et al. (2012). "Andromaly": A behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems, 38*, 161–190. https://doi.org/10.1007/s10844-010-0148-x
6. Canfora, G., et al. (2015). Detecting android malware using sequences of system calls, *3rd International Workshop on Software Development Lifecycle for Mobile*, 13–20. https://doi.org/10.1145/2804345.2804349

7. Ferrante, A., et al. (2016). Spotting the malicious moment: Char- acterizing malware behavior using dynamic features. *11th International Conference on Availability, Reliability and Security*. https://doi.org/10.1109/ARES.2016.70
8. Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: Sentiment classification using machine learning tech- niques, *ACL-02 Conference on Empirical Methods in Natural Language Processing, 10*, 79–86.
9. Turney, P. (2002) Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews. *40th Annual Meeting on Association for Computational Linguistics, ACL '02,* 417–424.
10. Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Fourteenth International Conference on Artificial Intelligence and Statistics.*
11. Zhang, R., et al. (2017). Real-time user-guided image colorization with learned deep priors. *TOG*.
12. Arandjelovic, R., et al. (2017). NetVLAD: CNN architecture for weakly su- pervised place recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(6).* https://doi.org/10.1109/TPAMI.2017.2711011
13. Upadhyay, S., et al. (2018). (Almost) Zero-shot cross-lingual spoken lan- guage understanding. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing*. https://doi.org/10.1109/ICASSP.2018.8461905
14. Li, J., et al. (2017). Adversarial learning for neural dialogue generation. *2017 Conference on Empirical Methods in Natural Language Processing*, 2157–2169. https://doi.org/10.18653/v1/D17-1230
15. Dong, L., et al. (2014). Adaptive recursive neural network for target-dependent twitter sentiment classification, *52nd Annual Meeting of the Association for Computational Linguistics, 2,* 49–54. https://doi.org/10.3115/v1/P14-2009
16. Elman, J. (1990). Finding structure in time. *Cognitive Science, 14(2),* 179–211. https://doi.org/10.1016/0364-0213(90)90002-E
17. Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long- term dependencies with gradient descent is difficult, *IEEE Transactions on Neural Networks, 5(2).* https://doi.org/10.1109/72.279181
18. Schuster, M., & Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing, 45(11).* https://doi.org/10.1109/78.650093
19. Hochreither, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*
20. Cho, K., et al. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP),* 1724–1734. https://doi.org/10.3115/v1/D14-1179
21. Arp, D., et al. (2014). DREBIN: Effective and explainable detection of android malware in your pocket. *NDSS*
22. Tobiyama, S., et al. (2016). Malware detection with deep neural network using process behavior," *IEEE 40th Annual Computer Software and Applications Conference (COMPSAC).* https://doi.org/ 10.1109/COMPSAC.2016.151

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 3 (23), 2024
ISSN 2663 - 4023

Vladyslav O. Sosnovyy
Assistant of the Department of Computer Engineering
State University of Information and Communication Technologies, Kyiv, Ukraine
ORCID 0000-0002-3217-4537
vladyslavsosnovyy@gmail.com

N.O.Lashchevska
Ph.D., associate professor, head of the Department of Computer Engineering
State University of Information and Communication Technologies, Kyiv, Ukraine
ORCID 0000-0003-2148-115X
lastchevska@gmail.com

DETECTION OF MALICIOUS ACTIVITY USING A NEURAL NETWORK FOR CONTINUOUS OPERATION

Abstract. This article describes the problem of detecting malicious programs in running systems of users of mobile applications. Because users can download any application on their phone, which over time can pull up additional settings, which can store malicious routines for monitoring both personal life and their personal data, such as logins, passwords, bank data. The detection of such routines is based on dynamic analysis and is formulated as a weakly controlled problem. The article contains an analysis of information on the development of researchers who worked on detection models and methods such as: statistical and dynamic intrusion detection methods, anomaly detection model, settings classification methods, machine and deep learning methods. Machine learning, and especially deep learning, has become an extremely useful and interesting topic in cybersecurity over the past few years. In this context, the detection of malicious software has received considerable attention. The article considers the problem of detecting the activity of malicious software of mobile operating systems in the time domain by analyzing behavioral sequences of a large amount of industrial data. When malware executes on a system, its behavior consists of a series of distinct actions placed along the time axis, and there is only a subsequence of actions that lead to malicious activity. Very often, malicious software does not manifest itself immediately, and at some point in the execution, malicious activity is formed. Therefore, the main task and difficulty is to identify such a subsequence in the entire sequence of events. Due to this, it is proposed to develop a behavior model that would analyze the dynamic behavior of the program in the system during execution. For this, a sequence of API/function calls generated by the program at runtime is used as input data and a Recurrent Neural Network (RNN) architecture is proposed to detect malicious activity. The article describes the training method of the proposed model and provides verification of its performance on a large sample of industrial data consisting of a large number of samples generated on the emulator farm. Many mobile phone vendors strive for hardware acceleration on the device to provide better support. Therefore, it can be considered that the deployment of a model based on RNM directly on the device as one of the security levels can become a viable solution. The test data of the model described in the article show sufficiently high positive results when detecting malicious activities.

Keywords: recurrent neural network; LSTM; machine learning; deep learning, neural networks; malware detection.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Lueg, C. (2017). *8,400 new Android malware samples every day.* https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android malware-samples-every-day
2. Feng, Y., et al. (2014). Apposcopy: Semantics-based detection of android malware through static analysis. *22nd ACM SIGSOFT International Symposium on Foundations of Software EngineeringNovember*, 576–587. https://doi.org/10.1145/2635868.2635869
3. Felt, A., et al. (2012). Android permissions demystified. *18th ACM conference on Computer and communications security*, 627–638. https://doi.org/10.1145/2046707.2046779

4. Mariconti, E., et al. (2016). MaMaDroid: Detecting android malware by building markov chains of behavioral models. https://doi.org/10.48550/arXiv.1711.07477
5. Shabtai, A., et al. (2012). "Andromaly": A behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems, 38*, 161–190. https://doi.org/10.1007/s10844-010-0148-x
6. Canfora, G., et al. (2015). Detecting android malware using sequences of system calls, *3rd International Workshop on Software Development Lifecycle for Mobile*, 13–20. https://doi.org/10.1145/2804345.2804349
7. Ferrante, A., et al. (2016). Spotting the malicious moment: Char- acterizing malware behavior using dynamic features. *11th International Conference on Availability, Reliability and Security*. https://doi.org/10.1109/ARES.2016.70
8. Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: Sentiment classification using machine learning tech- niques, *ACL-02 Conference on Empirical Methods in Natural Language Processing, 10*, 79–86.
9. Turney, P. (2002) Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews. *40th Annual Meeting on Association for Computational Linguistics, ACL '02,* 417–424.
10. Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Fourteenth International Conference on Artificial Intelligence and Statistics.*
11. Zhang, R., et al. (2017). Real-time user-guided image colorization with learned deep priors. *TOG*.
12. Arandjelovic, R., et al. (2017). NetVLAD: CNN architecture for weakly su- pervised place recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(6).* https://doi.org/10.1109/TPAMI.2017.2711011
13. Upadhyay, S., et al. (2018). (Almost) Zero-shot cross-lingual spoken lan- guage understanding. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing*. https://doi.org/10.1109/ICASSP.2018.8461905
14. Li, J., et al. (2017). Adversarial learning for neural dialogue generation. *2017 Conference on Empirical Methods in Natural Language Processing*, 2157–2169. https://doi.org/10.18653/v1/D17-1230
15. Dong, L., et al. (2014). Adaptive recursive neural network for target-dependent twitter sentiment classification, *52nd Annual Meeting of the Association for Computational Linguistics, 2,* 49–54. https://doi.org/10.3115/v1/P14-2009
16. Elman, J. (1990). Finding structure in time. *Cognitive Science, 14(2),* 179–211. https://doi.org/10.1016/0364-0213(90)90002-E
17. Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long- term dependencies with gradient descent is difficult, *IEEE Transactions on Neural Networks, 5(2).* https://doi.org/10.1109/72.279181
18. Schuster, M., & Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing, 45(11).* https://doi.org/10.1109/78.650093
19. Hochreither, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*
20. Cho, K., et al. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734. https://doi.org/10.3115/v1/D14-1179
21. Arp, D., et al. (2014). DREBIN: Effective and explainable detection of android malware in your pocket. *NDSS*
22. Tobiyama, S., et al. (2016). Malware detection with deep neural network using process behavior," *IEEE 40th Annual Computer Software and Applications Conference (COMPSAC).* https://doi.org/ 10.1109/COMPSAC.2016.151