УДК 681.3.07

Савчук Т.О.
Вінницький національний технічний університет

Паламарчук В.Л.
Вінницький національний технічний університет

АНАЛІЗ ВПЛИВУ ОБФУСКАЦІЇ ПРОГРАМНОГО КОДУ НА ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Поширення шкідливого програмного забезпечення на мобільних платформах заважає користувачам. Задля уникнення необхідності застосування механізму виявлення шкідливого ПЗ на основі цифрового підпису достатньо застосувати звичайну обфускацію програмного коду. У статті запропоновано підходи до виявлення шкідливого програмного забезпечення на основі машинного навчання, що базуються на статичному й динамічному аналізі.

Ключові слова: *шкідливе програмне забезпечення, Android, обфускація, безпека, машинне навчання.*

Постановка проблеми. Шкідливе програмне забезпечення (ПЗ), спрямоване на завдання шкоди користувачу, швидко та сильно поширюється останнім часом. Це сприяє діяльності зловмисників, оскільки користувачі зберігають все більше важливої інформації на своїх мобільних пристроях. При цьому слід взяти до уваги те, шо база користувачів платформи Android склала на жовтень 2018 року 37,4% проти 36,2% у настільних комп'ютерах під управлінням Windows [1].

Сучасні засоби виявлення шкідливого ПЗ, що базуються на перевірці цифрового підпису ПЗ, є недостатніми для захисту користувача від нових загроз з боку зловмисників. Перевірка цифрового підпису ПЗ зазвичай базується на класифікації шкідливого програмного забезпечення, що є трудомісткою операцією та займає багато часу [2].

Сучасні системи виявлення шкідливого програмного забезпечення використовую як сигнатурний, так й евристичний методи виявлення, поєднуючи їх переваги та недоліки [3]. Основним способом виявлення більшості компонентів шкідливого програмного забезпечення все ще є перевірка сигнатур [4]. Однак сигнатурний метод непридатний для захисту від нових та поліморфних вірусних програм, оскільки до того, як шкідливе програмне забезпечення потрапить на аналіз експертам, створити його сигнатуру неможливо. Наявні евристичні технології, покликані допомогти у визначенні нових модифікацій шкідливого програмного забезпечення, не дають належного рівня розпізнавання у зв'язку з їх слабкою ефективністю під час роботи з обфускованими об'єктами. До недоліків наявних методів виявлення шкідливого програмного забезпечення також можна віднести вразливість до нових атак, низьку точність виявлення та швидкість роботи [5].

Сучасні системи виявлення шкідливого програмного забезпечення переважно не пристосовані до роботи в реальному часі, тоді як можливість обробляти великий обсяг даних в реальному часі є визначальним фактором використання таких систем.

Нині популярністю користуються комплексні рішення, що поєднують всі методи захисту проти більшості шкідливих програм. Це негативно впливає на швидкість роботи операційної системи та збільшує кількість ресурсів, що споживає антивірусна система. До найбільш відомих програм вітчизняного та зарубіжного виробництва для повного захисту пристроїв слід віднести такі, як Norton AntiVirus Internet Security, Panda Antivirus Internet Security, Trend Micro Enterprise Protection Strategy, McAfee Active Virus Defense, Sophos Anti-Virus, NOD 32, а також найбільш популярні вітчизняні засоби захисту, такі як поліфаг Doctor Web та антивірусний комплекс «Доктор Касперський» [6].

Вказані недоліки важко усунути з використанням тільки класичних методів в галузі комп'ютерної безпеки [7]. У складі евристичних аналізаторів шкідливого програмного забезпечення активно використовуються штучні нейронні мережі (ШНМ) та штучні імунні системи (ШІС) [8].

Наявні ШНМ та ШІС дають змогу з великою часткою ймовірності виявити та ідентифікувати широкий клас шкідливого програмного забезпечення. Однак повністю ефективних способів боротьби із загрозами сьогодні не існує. Це визначає актуальність завдання розроблення нових підходів до виявлення та аналізування шкідливого програмного забезпечення, за допомогою яких можна

ефективно виявляти як старі, так і нові модифікації шкідливого програмного забезпечення з мінімальним навантаженням на систему.

Постановка завдання. Метою статті є формування підходів до виявлення шкідливого програмного забезпечення на основі машинного навчання, що базуються на статичному й динамічному аналізі.

Виклад основного матеріалу дослідження.

Виявлення шкідливого програмного забезпечення на основі машинного навчання.

Розглянемо форми виявлення шкідливого ПЗ на основі машинного навчання, що базуються на результатах статичного та динамічного аналізу [9]. В обох випадках підхід складається з таких фаз:

– фаза класифікації, у якій вхідні дані програми А класифікуються як шкідливі або довірені (нешкідливі);

– фази навчання, в якій класифікатор навчається на основі двох наборів інформації $A_Т$ і $A_М$, які включають шкідливе та довірене ПЗ відповідно.

В обох фазах вектор вхідних даних, який отримується з програми А, буде отримуватись на етапі попереднього оброблення [10].

Статичний аналіз.

Статичний аналіз коду -На етапі машинного навчання здійснюється процедура вибірки, оскільки отриманий вхідний вектор може виявитись занадто потужним. Для кожної процедурної послідовності обчислюється частота відповідно до наборів $A_Т$ і $A_М$:

$$f_M(o) = \frac{1}{|A_M|}\sum_{A\in A_M} f(A,o);$$

$$f_T(o) = \frac{1}{|A_T|}\sum_{A\in A_T} f(A,o),$$

де *o* – послідовність системних кодів; А – програма, для якої здійснюється вибірка; $A_Т$ – набір довірених програм; $A_М$ – набір шкідливих програм. Відносна різниця *d(o)* може бути визначена так [11]:

$$d(o) = \frac{abs\left(f_M(o) - f_T(o)\right)}{\max\left(f_M(o), f_T(o)\right)},$$

де *o* – послідовність системних кодів.

Отже, формується набір вибраних послідовностей О, який включає послідовності *o* з найбільшим значенням $d(o)$. Послідовності, для яких *d(o) = 1* (тобто ті послідовності, які присутні тільки в $A_Т$ або тільки в $A_М$), не розглядаються, аби уникнути отримання класифікатора, який не можна визначити. Крім того, всі послідовності *o* в наборі О, які є підпослідовностями або частиною іншої послідовності *o*, також відкидаються, оскільки є зайвою інформацією.

Отриманий на етапі класифікації вектор V вибраних послідовностей вхідної програми А спочатку обчислюється на основі частоти появи операційних кодів в О. Потім отримана інформація оброблюється за допомогою методу опорних векторів SVM, який формує відповідь у вигляді {довірене ПЗ, шкідливе ПЗ}.

Динамічний аналіз.

Динамічний аналіз програмного коду проводиться під час виконання програм на реальному або віртуальному процесорі (на відміну від статичного аналізу). Утиліти динамічного аналізу можуть вимагати завантаження спеціальних бібліотек, перекомпіляцію програмного коду, а також інструментувати код, що виконується. Задля підвищення ефективності динамічного аналізу програмного коду слід забезпечити достатню кількість вхідних даних досліджуваної програми [12].

Процес виявлення шкідливого ПЗ включає такі етапи.

На етапі попередньої обробки, системні виклики програми А впродовж запису її виконання створюють сліди виконання. Після цього для вхідного вектору обчислюється частота кожної послідовності n системних викликів у програмі А.

Як і в разі використання статичного аналізу, фаза машинного навчання починається з вибору та формування вхідного вектору. Щоби зменшити кількість вхідних послідовностей програми А, для аналізу вибирається тільки *k* послідовностей з найбільшим δ_s:

$$\delta s = \frac{\left|\frac{1}{|A_T|}\sum_{A\in A_T} f(A,s) - \frac{1}{|A_M|}\sum_{A\in A_M} f(A,s)\right|}{\max_{A_u}(A,s)},$$

де *s* – це кількість послідовність системних викликів; A_u – об'єднаний набір з $A_Т$ та $A_М$.

Кількість вхідних послідовностей в подальшому зменшується шляхом визначення кожної послідовності *s*, обчислюється взаємна інформація I_s (ступінь взаємної залежності кількості послідовностей *s* між наборами $A_Т$ та $A_М$) з *f(A,s)* і значенням *A* для будь-якого $A \in A_u$. Залишаються тільки послідовності з найвищим I_s, з яких отримується набір S з вибраних *s* послідовностей.

На етапі машинного навчання бінарний класифікатор (на основі SVM за законом розподілу Гауса та похибкою c=1) навчається на основі наборів даних, отриманих з наборів $A_Т$ та $A_М$, а також вибраних послідовностей, визначених набором S.

На етапі класифікації попередньо вибрані послідовності програмних кодів вилучаються з програми до непозначеного набору даних, до яких застосовується натренований класифікатор. В результаті формується відповідь у вигляді {довірене ПЗ, шкідливе ПЗ}.

Вхідні дані дослідження.

Для дослідження використано набір із 7 000 програм, які порівну розділені на набір довірених A_T і набір шкідливих програм A_M. Набір довіреного програмного забезпечення був взятий з магазину Google Play, а набір шкідливого програмного забезпечення був взятий з Drebin dataset [13]. Також був сформований набір A_O, який включає набір обфускованого програмного забезпечення, із застосуванням техніки обфускації на програмному забезпеченні набору A_M. Під час динамічного аналізу кожна програма була запущена на реальному пристрої під управлінням Android OS щонайменше 1 хвилину. Для симуляції програмних викликів під час роботи програми відбувались випадкові взаємодії з інтерфейсом програми.

Для формування набору A_O були використані такі техніки морфінгу програмного коду:

– розбирання та перезбирання коду;
– перепакування коду;
– зміна назви пакунку;
– перейменування ідентифікатора;
– кодування інформації;
– переспрямування програмних викликів;
– зміна порядку програмного коду;
– вставка «сміттєвого» коду.

Для спрощення оброблення результатів набори програм були розбиті на частини по 10 програм у кожній. Середні значення кожного набору були усереднені під час остаточного оброблення результатів.

Оброблення результатів.

Оброблення результатів проводиться у значенні точності правильно класифікованих програм: False Positive Rate (FPR) – відсоток довірених програм, що визначені як шкідливі; False Negative Rate (FNR) – відсоток шкідливих програм, що визначені як довірені; True Negative Rate (TNR) – відсоток довірених програм, що визначені як шкідливі; True Positive Rate (TPR) – відсоток довірених програм, що визначені як довірені.

Середні значення результатів TNR та TPR для тестового набору з 10 програм наведені на рис. 1.

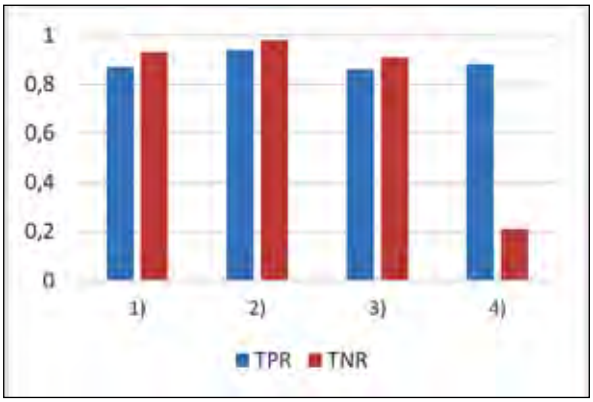

Рис. 1. Значення середнього TNR та TPR для тестового набору програм, де є такі складові: 1) обфусковане ПЗ/динамічний аналіз; 2) обфусковане ПЗ/статичний аналіз; 3) необфусковане ПЗ/динамічний аналіз; 4) необфусковане ПЗ/статичний аналіз

Середнє значення FNR та FPR, μ та стандартного відхилення σ FNR та FPR у двох сценаріях машинного навчання (на обфускованих та необфускованих шкідливих програмах) наведено в табл. 1.

Як видно з табл. 1, обидва методи виявлення шкідливого програмного забезпечення на основі машинного навчання ефективні у класифікації необфускованих програм, FPR та FNR перебувають нижче 10%. При цьому, статичний аналіз є більш точним при FPR < 4% та FNR < 3%, тоді як динамічний аналіз показує FPR ≈ 10% та FNR ≈ 6% відповідно. На точність динамічного аналізу негативно вплинула наявність великої кількості виконуваних програмних викликів.

Аналізуючи різницю між FNR та FNR_0, можемо дійти висновку, що на ефективність виявлення шкідливого програмного забезпечення на основі статичного аналізу суттєво впливає обфускація, тоді як на ефективність виявлення шкідливого програмного забезпечення на основі динамічного аналізу обфускація значно не впливає.

Для статичного методу $FNR_0 \approx 90\%$, тобто 9 з 10 програм зловмисних програм неправильно класифікуються як довірені. Це можна пояснити тим, що методи обфускації програмного коду, які використовуються в цьому дослідженні, сильно змінюють

Таблиця 1

Значення μ та σ для FNR та FPR

Метод		FPR		FNR		FNR_0	
		μ	σ	μ	σ	μ	σ
Обфусковане програмне забезпечення	Статичний	3,7	1,3	2,6	0,8	89,8	0,2
	Динамічний	9,9	1,0	5,8	1,4	7,5	0,3
Необфусковане програмне забезпечення	Статичний	6,6	1,8	0,6	0,1	0,1	0,1
	Динамічний	10,7	10	3,2	0,2	4,5	0,2

частоту та порядок операційних кодів у додатку, особливо під час переадресації системних викликів, впорядкування та вставки неприпустимого коду, що приводить до іншого розподілу послідовностей s, які більше не визначаються статичним класифікатором.

Висновки. У статті порівняно ефективність застосування методів виявлення обфускованого шкідливого програмного забезпечення на основі машинного навчання, що базуються на статичному або динамічному аналізі. Основне припущення в обох випадках полягає в тому, що обфускація коду додатка має залишити його трасування практично незмінним, що забезпечує надійність динамічного класифікатора під час обфускації, але повністю змінює послідовність кодів операцій, що випливають з її коду, а також робить статичний класифікатор абсолютно неефективним.

Це припущення підтверджене застосуванням двох сучасних методів легітимізації додатків, програм зловмисного програмного забезпечення та шкідливих програм, які піддаються 8 різним методам морфінгу програмного коду. Результати показують, що виявлення шкідливого програмного забезпечення на основі статичного аналізу є неефективнимпри обфускованому шкідливому програмному забезпеченні, що можна покращити за рахунок надання доступу додекомпільованого програмного коду шкідливих програм для машинного навчання.

Список літератури:

1. Number of available apps in Google Play. URL: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store (дата звернення: 28.11.2018).
2. Wang P., Bao Q., Wang L. Software Protection on the Go: A Large-Scale Empirical Study on Mobile App Obfuscation. URL: https://faculty.ist.psu.edu/wu/papers/obf-i.pdf (дата звернення: 28.11.2018).
3. Modern malware and threats. URL: https://www.slideshare.net/MarketingArrowECS_CZ/modern-malware-and-threats (дата звернення: 28.11.2018).
4. Гаврилов А.В. Применение постоянно модифицирующихся нейронных сетей для защиты программного обеспечения. *Нейрокомпьютеры, разработка, применение*. 2008. С. 90–101.
5. Obfuscation technics. URL: https://en.wikipedia.org/wiki/Obfuscation_(software) (дата звернення: 28.11.2018).
6. Modern anti-malware software. URL: https://stud.com.ua/62478/menedzhment/viyavlennya_virusiv_shkidlivih_program_usunennya (дата звернення: 28.11.2018).
7. Antivirus software. URL: https://en.wikipedia.org/wiki/Antivirus_software (дата звернення: 28.11.2018).
8. Machine learning malware detection. URL: https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf (дата звернення: 28.11.2018).
9. Метод відносних різниць. URL: https://studme.com.ua/137311011620/menedzhment/metod_absolyutnyh_raznits.html (дата звернення: 28.11.2018).
10. Artificial neural networks. URL: https://en.wikipedia.org/wiki/Artificial_neural_network (дата звернення: 28.11.2018).
11. Machine learning malware analysis. URL: https://www.ccsinet.com/blog/machine-learning-malware-analysis (дата звернення: 28.11.2018).
12. Dynamic analysis. URL: https://uk.wikipedia.org/wiki/Динамічний_аналіз_коду (дата звернення: 28.11.2018).
13. Drebin dataset. URL: https://www.sec.cs.tu-bs.de/~danarp/drebin (дата звернення: 28.11.2018).

АНАЛИЗ ВЛИЯНИЯ ОБФУСКАЦИИ ПРОГРАММНОГО КОДА НА ВЫЯВЛЕНИЕ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Распространение вредоносного программного обеспечения на мобильных платформах мешает пользователям. С целью устранения необходимости применения механизма выявления вредоносного ПО на основе цифровой подписи достаточно использовать обыкновенную обфускацию программного кода. В статье предложены подходы к выявлению вредоносного программного обеспечения на основе машинного обучения, которые базируются на статическом и динамическом анализе.

Ключевые слова: *вредоносное программное обеспечение, Android, обфускация, безопасность, машинное обучение.*

IMPACT ANALYSIS OF SOFTWARE CODE OBFUSCATION ON MALWARE DETECTION

The spread of malicious software on mobile platforms prevents users. In order to eliminate the need to use a mechanism for detecting malware based on a digital signature, it is sufficient to use ordinary code obfuscation. The article suggests approaches to detecting malware based on machine learning, which are based on static and dynamic analysis.

Key words: *malware, Android, obfuscation, security, machine learning.*