

МЕТОД ВИЯВЛЕННЯ СКРИПТ-ВІРУСІВ НА ОСНОВІ ДИНАМІЧНОГО АНАЛІЗУ ПОВЕДІНКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В КОМП'ЮТЕРНИХ СИСТЕМАХ

В роботі представлено метод виявлення скрипт-вірусів на основі динамічного аналізу поведінки програмного забезпечення в комп'ютерних системах. Метод дозволяє забезпечити реагування на нові скрипт-віруси, забезпечуючи захист комп'ютерних систем від як відомого, так і невідомого шкідливого програмного забезпечення типу скрипт-вірусів. Робота системи виявлення скрипт-вірусів здійснюється на основі обробки зібраних в комп'ютерній системі множини ознак скрипт-вірусів, виділення з неї підмножини таких ознак і створення таких необхідних для використання одного з множини класифікаторів, які дозволять виявити скрипт-віруси. Процес використовує апарат «багаторукого бандита» для вибору оптимального класифікатора програмного забезпечення, що дозволяє підвищити достовірність процесу виявлення скрипт-вірусів в комп'ютерних системах.

Ключові слова: скрипт-вірус, кібер-загрози, шкідливе програмне забезпечення, кібер-атака, комп'ютерна система.

S. LYSENKO, A. TSIUZIK
Khmelnytskyi National University

TECHNIQUE FOR SCRIPT-VIRUSES' DETECTION BASED ON THE DYNAMIC ANALYSIS OF THE SOFTWARE BEHAVIOUR IN THE COMPUTER SYSTEMS

The purpose of this paper is to develop a method for the script-viruses' detection based on the dynamic analysis of the software behaviour in the computer systems. The method allows to respond to new samples of script-viruses, providing the protection of the computer systems from both known and unknown samples of malicious scripts. The implementation of the method of detecting the script-viruses is based on the analysis of the set of script-virus features collected in the computer system. In order to detect the script-viruses, the method uses «the multi-armed bandit problem» for the purpose of the selection of the best classifier. The method consists of two stages: the training and the detection stages. In brief the technique involves the following steps: formation of the knowledge about functioning of the script-viruses in the computer systems; the formation of feature vectors of the malicious software; performing analysis of the script-viruses and distinguishing them into the classes; execution of the deletion of the detected malicious samples of the script-viruses.

Keywords: script-virus, computer system, malware, cyber thread, cyber- attack, software.

Вступ. За останні десять років кількість кібер-атак, здійснюваних шкідливим програмним забезпеченням (ШПЗ), невинно збільшується. Існує дві основні причини: все більше користувачів комп'ютерних систем (КС) та компаній проводять соціальні заходи та здійснюють ділові операції в цифровій формі; і все менше зусиль необхідно для успішного проведення кібер-атаки через збільшення кількості інструментів для виконання нападу. Зловмисне програмне забезпечення, зокрема, шкідливе програмне забезпечення, є однією з основних причин випадків кібер-атак. Згідно до звіту безпеки [1], у 2017 році, 16% з 45 мільйонів файлів, що відстежуються в кількох корпоративних мережах, є шкідливими програмами.

Стрімке збільшення шкідливих програм завдає серйозної шкоди широкій громадськості, бізнесу та національній безпеці. Це представляє загрозу цілісності і захисту персональних даних та обчислювальних систем. Розповсюдження шкідливих програм може мати різні наслідки, такі як порушення персональних або корпоративних даних та відмови критичної інфраструктури [2].

Одним з типів ШПЗ скрипт-віруси (script virus) – шкідливе програмне забезпечення, написане на мовах програмування Visual Basic, Basic Script, Java Script, Jscript, Shell тощо. На комп'ютерну систему користувача таке ШПЗ, найчастіше, проникає у вигляді поштових повідомлень, що містять у вкладеннях файли-сценарії. Програми на мовах Visual Basic і Java Script можуть розташовуватися як в окремих файлах, так і вбудовуватися в HTML – документ і в такому разі інтерпретуватися браузером, причому не лише з видаленого сервера, але і з локального диска [3].

Щоб здолати кібер-атаки, здійснювані шкідливим програмним забезпеченням типу скрипт-віруси, методи їх виявлення повинні мати можливість ідентифікувати та блокувати шкідливе програмне забезпечення з комп'ютерних систем або мережевого трафіку.

Одним з підходів до виявлення кібер-загроз, здійснюваних шкідливим програмним забезпеченням типу скрипт-віруси, є залучення механізмів машинного навчання, що надає можливості не лише виявляти, але й запобігати такі загрози. Такі підходи покладаються на евристичні механізми і нечіткі відповідності, що надає перевагу перед звичайними методами, зокрема перевіріці сигнатур [4–10].

З огляду на стан речей, що склався на даний момент, актуальною є задача розроблення методу виявлення ШПЗ типу скрипт-віруси, який мав би евристичну природу. Виявлення нових загроз має здійснюватися на основі динамічного аналізу поведінки програмного забезпечення в КС.

Метод виявлення скрипт-вірусів на основі динамічного аналізу поведінки програмного забезпечення в комп'ютерних системах

В роботі пропонується метод виявлення скрипт-вірусів на основі динамічного аналізу їх поведінки. Метод дозволяє забезпечити реагування на нові зразки ШПЗ типу скрипт-віруси, забезпечуючи захист комп'ютерних систем від як відомих, так і невідомих зразків шкідливих скриптів. Виконання методу виявлення ШПЗ типу скрипт-віруси здійснюється на основі обробки зібраних в комп'ютерній системі множини ознак скрипт-вірусів, а також вибору найоптимальнішого класифікатора для виявлення.

Для виявлення ШПЗ типу скрипт-віруси метод використовує так звану «задачу про багаторукого бандита» вибору оптимального класифікатора даних.

Метод складається з двох етапів:

- етап навчання системи виявлення кібер-загроз;
- етап виявлення ШПЗ типу скрипт-віруси.

Розглянемо кроки функціонування етапу навчання як складового методу виявлення скрипт-вірусів на основі динамічного аналізу їх поведінки:

- формування знань щодо ШПЗ типу скрипт-віруси на основі відомих поведінок;
- формування векторів ознак шкідливого програмного забезпечення типу скрипт-віруси;
- виконання аналізу шкідливого програмного забезпечення типу скрипт-віруси та поділ їх на класи;
- здійснення навчання наявних класифікаторів, необхідних для виявлення шкідливого програмного забезпечення типу скрипт-віруси та віднесення їх до відповідного класу, ґрунтуючись на навчальній вибірці з множини векторів ознак ШПЗ типу скрипт-віруси.

Розглянемо кроки функціонування етапу виявлення кібер-загроз на основі еволюційних алгоритмів:

- збір системної інформації, що вказує на можливу присутність ШПЗ типу скрипт-віруси;
- здійснення виявлення ШПЗ типу скрипт-віруси на основі застосування динамічного аналізу їх поведінки та вибору оптимального класифікатора для виявлення;
- на основі одержаного результату здійснення відповідного реагування на можливу присутність ШПЗ типу скрипт-віруси в КС.

Розглянемо більш детально етап навчання методу виявлення скрипт-вірусів на основі динамічного аналізу їх поведінки в КС більш детально.

Кожен конкретний навчальний зразок ШПЗ типу скрипт-вірус буде виконуватися кілька разів з різною довжиною виконання, яким позначимо множиною $\{\tau_1, \tau_2, \dots, \tau_k\}$. Прийmemo кінцевий набір навчальної вибірки моделей навчання $F = \{f_1, f_2, \dots, f_k\}$.

Тренувальні зразки можуть бути сумішшю корисного ПЗ та деяких множини відомих зразків ШПЗ типу скрипт-вірус, які були аналізовані вручну антивірусними організаціями або іншими дослідниками зловмисного програмного забезпечення. Існуючі класифікатори (або алгоритми) можуть бути застосовані для виявлення в функціональному просторі для детектора з найменшою помилкою перехресного підтвердження матимуть вигляд $f_k : X_k \rightarrow Y$. Тут X_k – функціональний простір, отриманий з T_k періоду динамічного аналізу представлених зразків, а Y – простір мітки.

Вибір оптимального класифікатора для виявлення ШПЗ типу скрипт-вірус

З метою вибору оптимального класифікатора для виявлення ШПЗ типу скрипт-вірус метод використовує апарат «багаторукого бандиту» [11–13] (передбачається вибір коротшого періоду динамічного аналізу) на основі контекстної інформації вибірки. Навчання кращого класифікатора серед багатьох є необхідним, тому що на відміну від задач, таких як розпізнавання мови та тексту, де функції аудіо та зображення залишаються відносно постійними, поведінка шкідливих програм типу скрипт-віруси еволюціонує, й існує ймовірність оновлення функціоналу ШПЗ, що спричинить здатність ШПЗ стати невидимим для системи виявлення. Отже, необхідно формування декількох моделей з різною довжиною поведінкових профілів і дозволяти системі вибирати найкращий класифікатор для досягнення високої точності виявлення шляхом захоплення більшої поведінкової активності.

Класифікація шкідливого програмного забезпечення на основі поведінкових функцій.

Класифікація шкідливих програм на основі поведінкових функцій зазвичай моделюється як задача навчання з учителем. Беручи це до уваги, класифікатор шкідливого програмного забезпечення типу скрипт-вірус, який пройшов навчання з позначеними векторами ознак, буде застосовано до векторизованих функцій, щоб обчислити ймовірність того, що зразок є шкідливим. Особливості поведінки шкідливого програмного забезпечення значною мірою залежать від тривалості моніторингу, продуктивність класифікатора, у свою чергу, залежить від τ . Взагалі, збільшення значення τ призводить до точнішого результату роботи класифікатора, а зменшення – до погіршення результатів роботи класифікаторів. Для покращення продуктивності класифікатора шкідливих програм, збільшення тривалості моніторингу поведінки τ призведе до зростання ресурсоемності процесу виявлення.

Для ефективного застосування методу в цілому необхідним є дотримання балансу та компроміс між досягненням високої точності та затратених ресурсів, пов'язаних зі встановлення значення параметра τ .

Таким чином, метод передбачає залучення множини класифікаторів $f_{\tau_1}, \dots, f_{\tau_k}$, що пройшли навчання з функціями поведінки з іншого періоду моніторингу τ_1, \dots, τ_k . Для кожної окремої задачі система в режимі реального часу вивчає, який класифікатор найкращий для вибору. Вказаний процес дозволяє інтерпретувати задачу як «задачу про багаторукого бандита» із застосуванням контекстної інформації про шкідливе

програмне забезпечення типу скрипт-віруси.

Формулювання задачі вибору класифікатора. «Задача про багаторукого бандита» включає множину K дій $A = \{a_1, \dots, a_K\}$. У кожній ітерації виконання дій $t = 1, \dots, T$, зроблено одну конкретну дію a_k та отримано винагороду $r_t(k)$ для даної дії. Нагорода $r_t(k)$ вибирається з стаціонарного розподілу ймовірностей, який залежить від дії k . Мета полягає у розробці стратегії, яка максимізує загальну винагороду шляхом вибору повторних дій.

Задача вибору класифікатора шкідливих програм може бути змодельована за допомогою контекстної багаторукої структури бандита.

Модуль вибору контекстної моделі демонструє підтримку кінцевого набору класифікаторів ШПЗ типу скрипт-віруси $F = \{f_1, f_2, \dots, f_k\}$, де $K = \{1, 2, \dots, k\}$, для якого кожен класифікатор $f_k \in F$, який динамічно навчається на основі поведінкових особливостей ШПЗ типу скрипт-віруси з певного часу виконання τ_k і пов'язаний з невідомим і фіксованим розподілом D $[0, 1]$.

У рамках процесу обробки запитів система виявлення вибере та застосує один з k класифікаторів до вектору поведінкових функцій даної вибірки x^t для виведення результату класифікації $y^t = f_k(x^t) \in Y = \{0, 1\}$. Це відповідає ручному вибору, для застосування апарату «задачі про багаторуких бандитів». У нашій моделі винагорода, отримана для модуля, вибравши f_k , є індикаторною функцією $r_t = 1$ ($y^t = y^{*t}$), в якій $y^{*t} \in Y$ є міткою зразка. Значок "1" у двозначному ярлику, на якому Y позначає шкідливе програмне забезпечення, та "0" для корисного програмного забезпечення.

Результат виявлення y^t також може бути прогнозуванням ймовірності (наприклад, $y^t \in Y' = [0, 1]$, значення з діапазону представляють найнижчу та найвищу можливість присутності шкідливого програмного забезпечення).

Варто зазначити, що векторний знак x^t та тренувальні особливості класифікатора f_k походять від поведінкових функцій, зібраних під час динамічного виконання для довжини часу τ_k .

Кожен з класифікаторів має очікувану або середню нагороду, оскільки вона обрана в декілька заходів – точність класифікатора. Прийmemo позначення класифікатора, вибраного за часом t як F_t , таким чином, точність довільного класифікатора f_k , позначеного $q(f_k)$, є очікуваною нагородою, якщо вибрано f_k :

$$q(f_k) = E[r_t | F_t = f_k].$$

Точність – це простий та інтуїтивно зрозумілий показник для оцінки системи класифікації. Насправді, більшість динамічних систем, що базуються на системах класифікації шкідливих програм, представили результат оцінки в аналогічній формі вимірювання: точність, відкликання, оцінка FI тощо, ігноруючи витрати, пов'язані з проведенням аналізу динамічних поведінкових особливостей.

Для забезпечення умови збалансованого компромісу між високою точністю аналізу та ресурсоемністю аналізу метод застосовує метрику «якості досвіду» Q_{fk} .

Метрика «якість досвіду», одержуваного користувачем a^t на момент $t + \tau_k$, вибравши k -й класифікатор з F у момент часу t , є зважена сума точності класифікатора та витрати через τ_k часу динамічного аналізу:

$$Q(f_k) = q(f_k) - \beta c(\tau_k),$$

де $B \in [0, 1]$ – параметр компромісу, який залежить від прикладних вимог.

Метрика «якості досвіду» як вимірювання того, як працює система виявлення на основі хмар. Коротко, ми хочемо максимально збільшити очікування метрика «якості досвіду», визначивши, як довго виконувати кожен зразок, щоб застосувати один із підтримуваних класифікаторів на основі їх оціненого фідбеку, тобто історії метрика «якості досвіду». Оскільки у нас немає істинного значення $Q(f_k)$, ми повинні оцінити його для кожного k , щоб знайти максимум. Один із можливих методів – почати з великого значення k для вивчення якомога більшої кількості поведінкових функцій, доки не буде досягнута ситуація, коли найменший k , що призводить до найвищих емпіричних значень метрики «якості досвіду» серед K , і переходить до конкретного вибору оптимального значення k^* , щоб скористатися перевагами швидкого та точного виявлення. Проте цей жадний метод сприяє неоптимальному результату, оскільки в усіх майбутніх виявленнях вони використовують лише свій попередній відомий найкращий класифікатор, модель поведінки якої може бути неприйнятною для захоплення поведінкової характеристики нових зразків.

Хоча експлуатація корисна для максимального збільшення значення метрики «якості досвіду» на один крок, також необхідно вивчити інші класифікатори, не вибрані жадібним методом, щоб покращити передбачувану точність, оскільки дослідження в перспективі може забезпечити більшу загальну метрика «якості досвіду». Наприклад, якщо було виявлено, що f_1 – це класифікатор з жадібним вибором, тоді як деякі інші класифікатори оцінюються як такі, але невизначені. Невизначеність полягає в тому, що в майбутньому може існувати один із цих інших класифікаторів, які кращі, ніж f_1 , але невідомо, який з них у часі t . Метод передбачає здійснення вибору класифікатора на кожному етапі, і тоді необхідно краще вивчити інші класифікатори та визначити, які з них корисніші в довгостроковій перспективі.

В процесі навчання порівняння метрика «якості досвіду» з різними класифікаторами не є практичним. Краще вимірювання успіху в навчанні – це поняття втрат.

Втрати алгоритму навчання. Враховуючи загальну кількість запитів на виявлення зразка T , яка обробляється відповідно до алгоритму виявлення, сумарні втрати являють собою різницю між загальною метрикою «якості досвіду» шляхом застосування кращого класифікатора та загального метрика «якості досвіду», дотримуючись алгоритму у всіх T -виявленнях.

Метою розробки алгоритму для задачі «багаторукого бандита» є мінімізація сукупності очікуваних втрат, які очікуються за лінійністю:

$$E[\text{Re } g_A(T)] = T\mu^* - E\left[\sum_{t=1}^T Q_t(A)\right]$$

де $\mu^* = \max_{1 \leq i \leq k} \mu_i$ очікується метрика «якості досвіду» кращого класифікатора. Складність алгоритму $O(\sqrt{KT \log T})$.

Застосування апарату контекстних «багаторуких бандитів» для виявлення скрипт-вірусів

Застосування підходу до виявлення скрипт-вірусів «багаторукими бандитами» на основі залучення метрики «якості досвіду» дозволяє визначити лише класифікатор, який обраний в момент часу t . Будь-яка додаткова інформація про зразок не враховується.

Щоб досягти поставленої мети, метод повинен вивчати овертайми, які класифікатор виконує найкраще для наступного запиту на виявлення. Важливо використати подібність сторонньої інформації та враховувати інформацію в майбутньому процесі навчання. Цю проблему можна природно моделювати як контекстуальну задачу «багаторуких бандитів». Тобто метрика «якості досвіду» виявлення залежить не тільки від обраного класифікатора, але також залежить від того, як вивчається наявна контекстна інформація, щоб отримати найкраще навчання.

Для того, щоб включити контекст зразка скрипт-вірусу в налаштування, метод передбачає спочатку формування контекстної функції із отриманого клієнтського запиту та виконується кластеризація контексту, щоб вибрати класифікатор відповідно до кластерного результату. Для цього необхідно абстрагувати функцію контексту зразка в момент t , використовуючи позначення $\theta^t \in \Theta$, де Θ являє собою d -вимірний функціональний простір. Контекст може містити інформацію про різні властивості файлу зразка скрипт-вірусу, наприклад заголовок файлу та інші стандартні метадані специфікації файлів.

Під цією контекстним формулювання необхідним є невідомий розподіл P через $(\Theta, Q_\Theta(f_1), \dots, Q_\Theta(f_k))$. На кожному кроці вибірка $(\Theta, Q_\Theta(f_1), \dots, Q_\Theta(f_k))$ береться з P , контекст Θ оголошується, а потім для одного класифікатора вибирається алгоритм бандитів, його значення метрики вираз $Q_\Theta E Q_\Theta(f_k)$ визначеним. Алгоритм контекстних бандитів вибирає класифікатор для кожного часового інтервалу t на основі попередньої послідовності спостережень $(\Theta^1, f_k^1, Q_{\Theta^1}(f_k^1)), \dots, (\Theta^{t-1}, f_k^{t-1}, Q_{\Theta^{t-1}}(f_k^{t-1}))$ та поточний контекст Θ^t . Очікувані сукупні втрати (втрати навчання) алгоритму B відносно бенчмарку за часом T складають:

$$E[\text{Re } g_\beta(T)] = T\mu_\theta^* - E\left[\sum_{t=1}^T Q_\theta^t(\beta(\theta))\right].$$

Очікувана кумулятивна винагорода дає швидкість конвергенції загальної очікуваної винагороди алгоритму навчання до вартості оптимального рішення. Будь-який алгоритм навчання має складність $O(N^\gamma)$ (для $\gamma < 1$) сходиться до оптимального рішення з точки зору очікуваної винагороди. Іншими словами, метою алгоритму B є мінімізація його втрат, що еквівалентно максимізації загальної винагороди.

Припущення: Контекстна задача «багаторукого бандита» Ліпшица є парою просторів – контекстний простір Θ та функція F . Екземпляр завдання – це функція $\mu_{\theta \in \Theta} : \Theta \times F \rightarrow [0, 1]$, яка є Ліпшица функцією кожній координаті, тобто $\forall \theta, \theta' \in \Theta, \forall f_k \in F$, існує $M > 0$ і $0 < \alpha \leq 1$ таким, що

$$|\mu_\theta(f_k) - \mu_{\theta'}(f_k)| \leq M|\theta - \theta'|^\alpha$$

Зауваження: вищесказане припущення буде виконуватися, якщо очікуваний метрика «якості досвіду» $\mu_\theta(f)$ обмежена $\forall \theta, f$ і M обраний достатньо великим, а α суттєво малим. хоча перевага надається меншим розмірам M і α , тому що в цьому випадку існує більше інформації про подібність, яка може бути використана, і може призвести до кращої продуктивності системи.

Здійснення навчання контекстних «бандитів» для оптимізації метрики «якості досвіду». З метою виконання оптимізації метрики «якості досвіду» необхідним є здійснення навчання контекстних «бандитів» з цією метою необхідно розглянути питання кластеризації контексту та принципи використання контекстної інформації для оптимізації значення метрики «якості досвіду» через запропоновану контекстуальну модель «багаторукого бандита». На фазах розвідки відбираються різні

класифікатори, щоб вивчати їх очікувану винагороду. У фазах експлуатації вибирається класифікатор з найкращою оцінною нагородою, щоб максимізувати класифікаційні нагороди. Етапи дослідження та експлуатації змішані, на відміну від звичайних навчальних підходів, де виконується лише одна фаза тренувань, а потім фаза експлуатації.

Метрика «якості досвіду» для різних класифікаторів буде відрізнятися, оскільки тривалість поведінкових збірок має значний вплив на очікуване значення метрики «якості досвіду» $\mu(f_k^*)$ класифікатора ШПЗ типу скрипт-віруси на основі поведінки. Збільшення часу виконання буде записувати більш комплексні поведінкові функції, які зазвичай призводять до більш точних результатів. Покращення в основному застосовується для виявлення шкідливих програм, які навмисно або ненавмисно затримують шкідливу поведінку після аналізу. Зрештою, короткий аналіз не охоплюватиме будь-які корисні функції поведінки для такого типу шкідливого програмного забезпечення та, призведе до хибного результату виявлення.

Дослідження контекстної кластеризації функцій. Розглянемо зв'язок між контекстною інформацією та точністю класифікатора $q(fk), q(fk)$, що, у свою чергу, впливає на значення метрики «якості досвіду». Наприклад, дві групи зразків ШПЗ типу скрипт-віруси зі значними властивостями файлів можуть отримувати різні значення метрики «якості досвіду» при застосуванні однакової політики відбору. Наприклад, однією групою є упаковані скрипт-віруси, а інша група неупаковані. Проблема навчання була б простою, якщо б не було контекстної інформації. Але, не використовуючи контекстну інформацію, ефективність алгоритму навчання може бути поганою, оскільки найкращі класифікації «оракула» можуть бути різними для різної контекстної інформації. Оскільки контекстний простір Θ може бути дуже великим і навіть безперервним, навчання найкращого класифікатора «оракула» для кожного окремого контексту $\theta \in \Theta$ є надзвичайно складним, якщо не неможливим. Щоб подолати цю перешкоду, алгоритм навчання спочатку розділить контекстний простір на менші підпростори (тобто кластери контексту) і вивчатиме найкращий класифікатор «оракул» у кожному підпросторі.

Як приклад, розглянемо алгоритм кластерного аналізу K-means як засіб розділу простору контексту. Алгоритм ітераційно виконується для кожної функції контексту навчального зразка скрипт-віруса, щоб віднести його до найближчому центроїду кластера за допомогою метрики евклідової відстані і перерахує середнє значення кожного центроїду за допомогою заданої йому точки.

Алгоритм K-means завжди буде збігатися з деяким кінцевим набором засобів для центроїдів.

Зауважте, що рішення не завжди може бути оптимальним і залежить від початкового налаштування центроїдів. Тому на практиці алгоритм K-means запускається кілька разів з різними випадковими початковими значеннями. Метод передбачає вибір найкращих центроїдів між різними рішеннями, мінімізуючи функцію витрат:

$$J(\ell^1, \dots, \ell^T, v_1, \dots, v_L) = \frac{1}{T} \sum_{t=1}^T \|\theta^t - v_{\ell^t}\|^2$$

де $\ell^t \in L = \{1, \dots, L\}$ $\ell^t \in L = \{1, \dots, L\}$ – це індекс кластера, до якого віднесено контекст θ^t вибірки, а v_{ℓ^t} – центроїди контекстного кластера.

Для конкретного запиту система виявлення отримує контекстні функції від клієнта як перший крок у транзакції виявлення. Отримана функція повинна бути нормалізованою. Наприклад, якщо включено лише розмір файлу в контекстну функцію, контекстне поле буде нормалізовано щодо максимального розміру файлу та мінімального розміру файлу, отриманого до цих пір. Нормалізовані контекстні функції будуть запускатися за допомогою попередньо вбудованої моделі кластеризації, і кластерний мітку вхідної вибірки буде виявлено. Алгоритм навчання визначатиме оптимальну тривалість трасування для цього зразка на основі мітки контексту та нагороди історії доступних класифікаторів.

Для кожної конкретної вибірки з кластерною міткою y_θ реалізована метрика **метрика** $Q_\theta(f_k) Q_\theta(f_k)$, вибравши f_k , – як випадкову величину, взятую з невідомого розподілу із середнім $\mu_\theta(f_k) \mu_\theta(f_k)$, що також спочатку невідома. Однак, можна оцінити очікувану метрика «якості досвіду», спостерігаючи за багатьма реалізаціями винагороди від тестування зразків. Зокрема, найкращим класифікатором у контексті $\theta \in f^*(\theta) := \arg \max_{f_k \in F} \mu_\theta(f_k)$ $\theta \in f^*(\theta) := \arg \max_{f_k \in F} \mu_\theta(f_k)$ і найкраще значення очікуваної метрики «якості досвіду» для контекстного кластера $y_\theta \in \mu_\theta^* := \mu_\theta(f^*(\theta))$ $y_\theta \in \mu_\theta^* := \mu_\theta(f^*(\theta))$. Прийmemo $f^*(\theta) f^*(\theta)$ класифікатором «оракула» для контекстного кластера y_θ . Класифікатори «оракула» не визначені перед виявленням, але замість цього мають бути визначені. Навчання здійснюється шляхом багаторазового тестування зразків з класифікаторами з політикою вибору класифікатора, який повинен бути застосований.

Формальний опис алгоритму виявлення

Пропонований метод залучає алгоритм більшої вірогідності (UCB) [14, 15], але з інформацією про контекстну інформацію та оновлення класифікатора. Формальний опис алгоритму представлений на рисунку 1 (ConUCB). Він використовує зразок контекстної інформації, щоб визначити кращий класифікатор для контексту (таким чином, оптимальну динамічну довжину аналізу) протягом часового горизонту, максимально збільшуючи очікуваний метрика «якості досвіду» користувача служби виявлення шкідливих програм.

Під час процедури навчання алгоритм підтримує декілька лічильників та оцінювану точність $q^{-1}(f_k)$ та метрику $Q^{-1}(f_k)$ для кожного доступного класифікатора $F = \{f_1, \dots, f_k\}$ з різним контекстом типу v_l . Лічильник N_k^l записує, скільки разів класифікатор f_k був обраний для класифікації зразків, контекст яких v_l до округленого значення t . Лічильник N_k позначає загальну кількість класифікатора f_k , яка вибирається у всіх турах t . Лічильник N – це загальна кількість зразків, які були передані для аналізу. У початковому алгоритмі кожен класифікатор застосовується для кожного типу контексту для ініціалізації передбачуваної метрики $\bar{Q}(f_k)$. Для кожного майбутнього представленого зразка алгоритм спочатку запускає процедуру кластеризації, щоб отримати тип кластеру, а потім обрати класифікатор, беручи до уваги як максимально близькі поточні оцінки, так і дисперсію оцінки. Після вибору класифікатора та запуску виявлення, оцінку метрика «якості досвіду» та відповідних лічильників буде оновлено.

Input: $\alpha \in \mathbb{R}^+$, $\mathcal{S} = \{(\theta^1, \mathbf{x}^1), (\theta^2, \mathbf{x}^2), \dots, (\theta^t, \mathbf{x}^t)\}$,

$\mathcal{F} = \{f_1, f_2, \dots, f_K\}$, $\mathcal{K} = \{1, \dots, K\}$, $\beta \in [0, 1]$,

$\mathcal{M} = \{\nu_1, \nu_2, \dots, \nu_L\}$, $\mathcal{L} = \{1, \dots, L\}$

Output: $\{y^1, \dots, y^t\} \in \{0, 1\}$

1: Initialization:

2: for $\ell \in \mathcal{L}$ do

3: for $k \in \mathcal{K}$ do

4: Randomly select (θ^m, \mathbf{x}^m)

5: Set $\bar{q}_\ell(f_k) \leftarrow f_k(\mathbf{x}^m)$

6: Set $\bar{Q}_\ell(f_k) \leftarrow \bar{q}_\ell(f_k) - \beta c(\tau_k)$

7: Set $N_k^\ell \leftarrow 1$

8: end for

9: Set $N^\ell \leftarrow K$,

10: end for

11: Set $N \leftarrow LK$,

12:

13: for each malware detection request (θ^t, \mathbf{x}^t) do

14: $\ell^* = \arg \min_{\ell \in \mathcal{L}} \|\theta^t - \nu_\ell\|^2$

15: $k^* = \arg \max_{k \in \mathcal{K}} (\bar{Q}_{\ell^*}(f_k) + \sqrt{\frac{\alpha \ln N^{\ell^*}}{N_k^{\ell^*}}})$

16: Set $r_t = f_{k^*}(\mathbf{x}^t)$

17: Set $\bar{q}_{\ell^*}(f_{k^*}) \leftarrow \bar{q}_{\ell^*}(f_{k^*}) + \frac{1}{N_{k^*}^{\ell^*}} [r_t - \bar{q}_{\ell^*}(f_{k^*})]$

18: Set $\bar{Q}_{\ell^*}(f_{k^*}) \leftarrow \bar{q}_{\ell^*}(f_{k^*}) - \beta c(\tau_{k^*})$

19: Set $N^{\ell^*} \leftarrow N^{\ell^*} + 1$

20: Set $N_{k^*}^{\ell^*} \leftarrow N_{k^*}^{\ell^*} + 1$

21: Set $N \leftarrow N + 1$

22: end for

Рис. 1. Формальний опис алгоритму ConUCB

Кількість, яка є максимальною в рядку 15 даного алгоритму, – це верхня межа довіри до можливого значення метрика «якості досвіду» класифікатора f_k для конкретного типу контексту, де параметр α контролює ширину довірчого інтервалу. Кожен раз, коли класифікатор f_k вибирається для типу контексту v_l , дисперсія \bar{Q}_l^* зменшується, оскільки N_k^{l*} знаходиться в знаменнику дисперсії. З іншого боку, кожен раз, коли класифікатор, крім f_k , вибирається для типу контексту v_l , термін дисперсії оціночної метрика «якості досвіду» для f_k буде зберігатися незмінним. У той час, коли для певного типу контексту буде

більше часу для класифікатора з нижчою оцінкою вартості, або вже вибраним буде довший час очікування, і, отже, нижча частота вибору.

У алгоритмі 1 видно, що якщо обрано класифікатор з великою варіативністю дисперсії, то цю дію можна розглядати як рішення, оскільки в такому випадку верхня межа є вільною і беручи $\overline{Q_{l^*}}$ як оцінку справжньої очікуваної нагорода є сумнівною. Ймовірно, деякі інші класифікатори перевершують значення f_k . І навпаки, якщо вибрано задачу з великим значенням метрики «якості досвіду», цю дію можна розглядати як шукане рішення. І навпаки, якщо вибирається рука з великим розрахунком $QoE \overline{Q_{l^*}}(f_k) QoE \overline{Q_{l^*}}(f_k)$, можна розглядати дію як експлуаторське рішення. Враховуючи те, що $\sqrt{\frac{\alpha \ln N_k}{N_k^{l^*}}} \sqrt{\frac{\alpha \ln N_k}{N_k^{l^*}}}$ швидко зменшується з кожним вибором k , кількість пошукових рішень обмежена.

Коли $\sqrt{\frac{\alpha \ln N_k}{N_k^{l^*}}}$ стає меншим, середній $\overline{Q_{l^*}}(f_k) \overline{Q_{l^*}}(f_k)$ наближається до справжнього очікуваного $QoE \overline{Q_{l^*}}(f_k)$, і з високою ймовірністю, що класифікатор, що відповідає максимальному метрика «якості досвіду» для типу контексту, дійсно є оптимальним класифікатором контексту.

Формування множини поведінок ШПЗ типу скрипт-віруси для виконання експериментальних досліджень

Набір даних експерименту включав в себе 3000 портативних виконуваних зразків типу скрипт, серед яких 1500 – скрипт-віруси, а інші 1500 – безпечне програмне забезпечення. Мітка "Істина" отримується за допомогою будь-якого онлайн-сканера, наприклад Virus Total [14].

Експериментальна вибірка розподілена на три підмножини з кожним зразком по 1000.

Перша підмножина для початкової підготовки, друга підмножина для початкового тестування та постійного оновлення класифікаторів, а третя – підмножина для безперервного тестування.

На рисунку 2 показано розрізнення контекстної характеристики перших двох підмножин. У якості контекстної функції метод оперує розміром файлу.

Кількість кластерів визначається на підставі оцінки показника Silhouette, яка дозволила визначити кластери, які є щільними та добре розділеними, що відповідає вимогам кластеризації контексту.

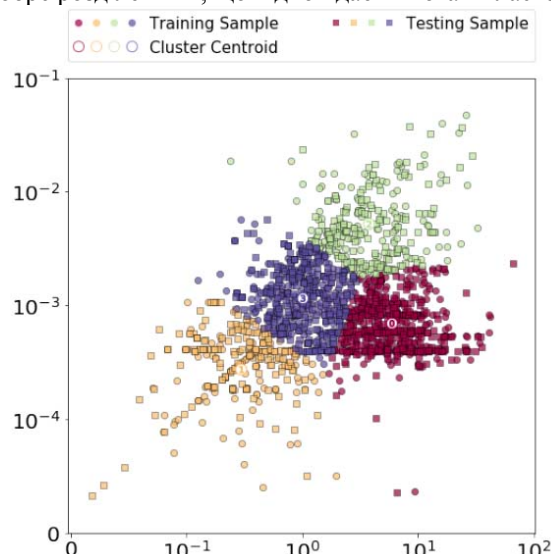


Рис. 2. Розрізнення контекстної характеристики програмного забезпечення

Для оцінки ефективності було використано функцію лінійної вартості, тобто $c(\tau_k) = \tau_k c(\tau_k) = \tau_k$ у визначенні метрика «якості досвіду» та порівнюємо передбачувану очікувану метрика «якості досвіду», отриману шляхом застосування пропонуваніх алгоритмів, а саме ConUCB та \mathcal{E} -ConUCB, над класифікаторами та очікуваними метрика «якості досвіду», отримані кожним окремим класифікатором.

Перший експеримент використовував початковий набір навчальних даних щодо ШПЗ типу скрипт-віруси з 1000 зразків для побудови динамічних поведінкових класифікаторів та проводив оцінку, використовуючи початковий набір тестів з 1000 зразків. На рисунку 3 показано нормоване значення метрики «якості досвіду» при $\beta = 0.01$. Криві метрика «якості досвіду» отримуються шляхом розрахунку очікуваного значення ста партій за рандомізованими послідовностями вибірки тесту з запропонованим алгоритмом. Два запропонованих алгоритми навчання перевершують всі окремі класифікатори. Алгоритм

ConUCB покращив максимальне значення метрики «якості досвіду» з чотирьох окремих класифікаторів з 91% до 94% після 1000 партій класифікації шкідливих програм. ϵ -ConUCB алгоритм також отримує до 2% нагород. Оскільки експеримент має сталу кількість партій, а використовувана контекстна інформація обмежена розміром секції PE-коду та розміром файлу PE, значно вищий приріст продуктивності може очікуватися, коли буде відтворено більше партій і доступно більше контекстної інформації.

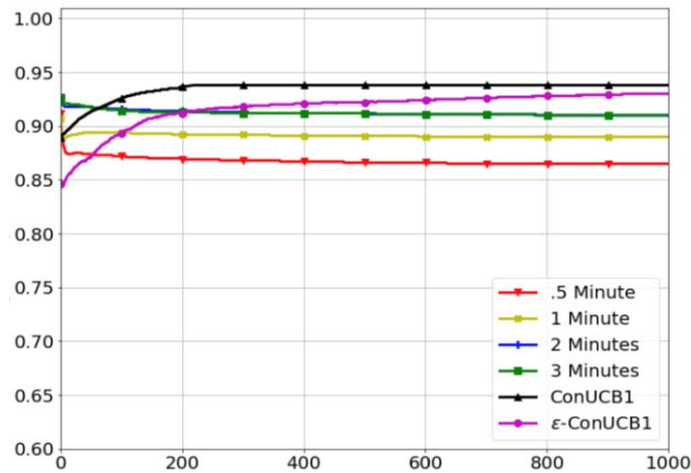


Рис. 3. Нормоване значення метрики «якості досвіду» при $\beta = 0.01$.

На рисунку 4 надано порівняння ефективності алгоритмів з продуктивністю кожного окремого класифікатора. Як і ConUCB, так і ϵ -ConUCB досягли нижчого рівня достовірності виявлення, ніж окремі класифікатори. ConUCB і ϵ -ConUCB збільшили площу під кривою ROC до 96% і 94%.

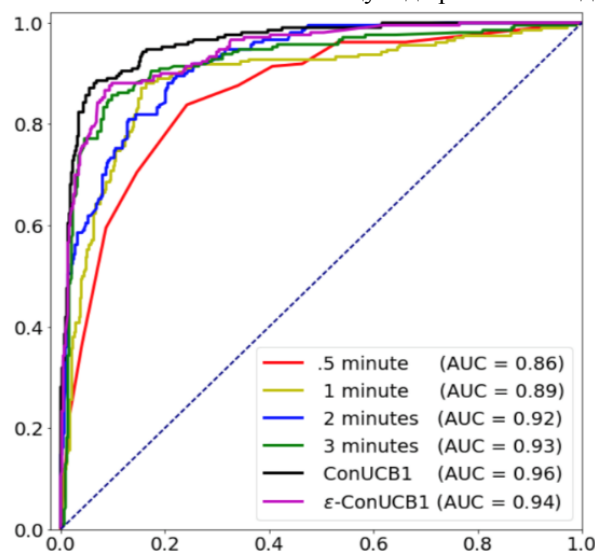


Рис. 4. Порівняння ефективності алгоритмів з продуктивністю кожного окремого класифікатора

На рисунку 5 представлена нормоване значення метрики «якості досвіду» для $\epsilon = 0.1$. У порівнянні з рисунком 4, збільшення значення коефіцієнта витрат зменшить значення метрики «якості досвіду» всіх чотирьох основних класифікаторів, таким чином, зменшиться значення метрики «якості досвіду» для ConUCB і ϵ -ConUCB, оскільки алгоритм, як правило, оптимізується до менш точного класифікатора, який пройшов навчання на 30 секунд пріоритетних характеристик функцій.

Динамічне навчання з контекстною інформацією

З метою дослідження ефективності методу було вивчено кожну дію окремо, із застосуванням алгоритму ConUCB. Рисунок 6 показує етапи вибору класифікатора протягом експерименту з 1000 раундів і показує отриману метрику «якості досвіду». Нижня панель кольорів на малюнку ілюструє всі 1000 дій з використанням чотирьох різних кольорів, кожен з яких представляє окремий класифікатор, який вибирається на кроці. Чотири панелі кольорів, розташовані над ним, ілюструють дії, здійснені під кожним різним контекстним кластером. Кожен рядок з цих чотирьох панелей кольорів включає в себе дії, взяті над зразками, належить до одного контекстного кластера. З чотирьох панелей кольорів на рисунку можна помітити, що кожен контекстний кластер поступово найкраще навчився класифікатором для вибору в конкретному контексті. Наприклад, у перших 200 діях ConUCB не має ніяких переваг для будь-яких

окремих класифікаторів, і кожен класифікатор має таку саму ймовірність вибору. Це відповідає фазі дослідження. З іншого боку, коли «гра» триває до 800-го раунду, алгоритм вступає в фазу експлуатації, оскільки найкращий класифікатор для контексту виділений з високою ймовірністю.

Рис. 7 показує відсоток найкращої дії в різному раунді, застосовуючи ConUCB з $\epsilon = 0.01$.

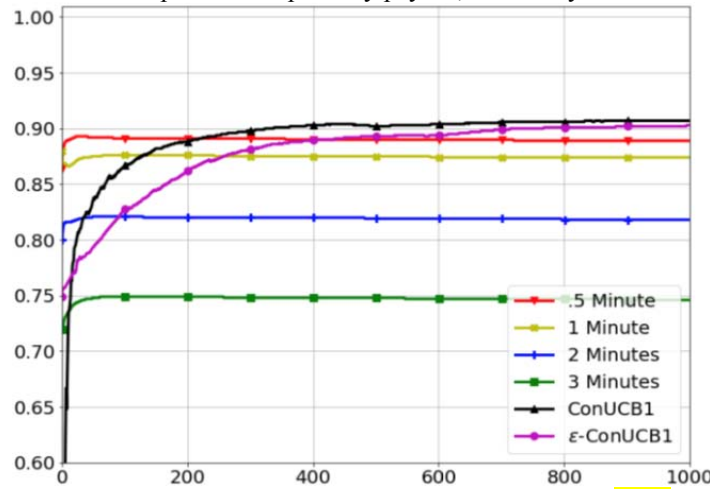


Рис. 5. Нормоване значення метрики «якості досвіду» для $\epsilon = 0.1$

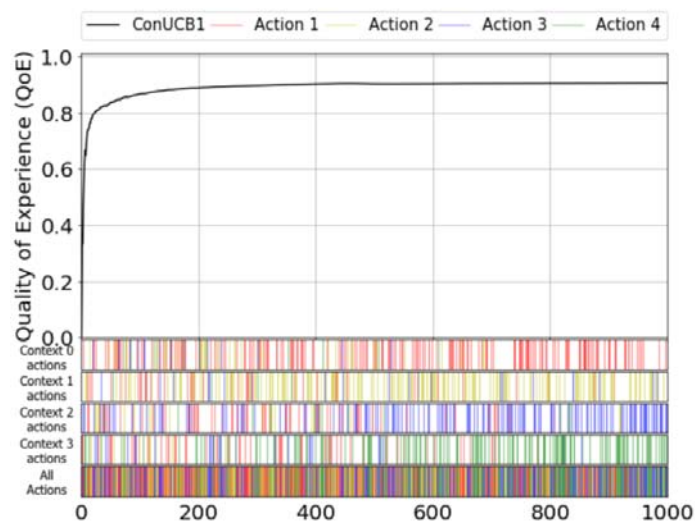


Рис. 6. Нормоване значення метрики «якості досвіду» для кожного раунду

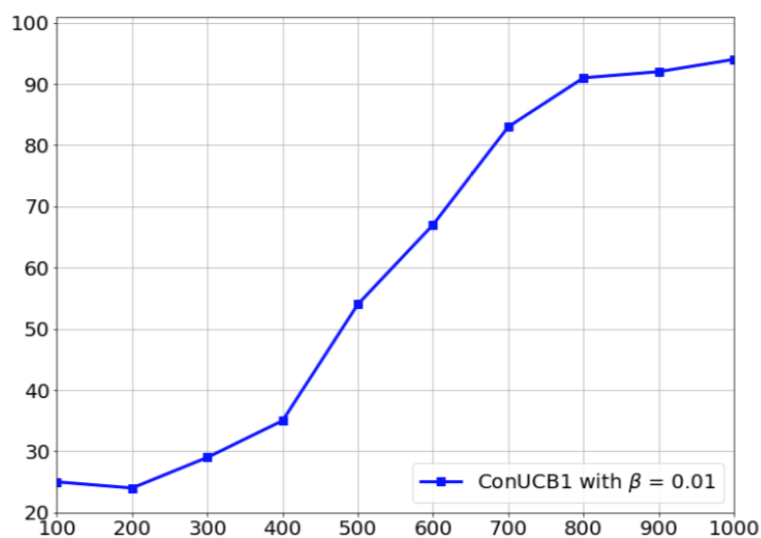


Рис. 7. Відсоток найкращої дії в різному раунді, застосовуючи ConUCB з $\epsilon = 0.01$

Висновки

Запропоновано метод виявлення скрипт-вірусів на основі динамічного аналізу поведінки програмного забезпечення в комп'ютерних системах.

Метод дозволяє забезпечити реагування на нові скрипт-віруси, забезпечуючи захист комп'ютерних систем від як відомого, так і невідомого шкідливого програмного забезпечення типу скрипт-вірусів.

Робота системи виявлення скрипт-вірусів здійснюється на основі обробки зібраних в комп'ютерній системі множини ознак скрипт-вірусів, виділення з неї підмножини таких ознак і створення таких необхідних для використання одного з множини класифікаторів, які дозволять виявити скрипт-віруси.

Процес використовує апарат «багаторукого бандита» для вибору оптимального класифікатора програмного забезпечення, що дозволяє підвищити достовірність процесу виявлення скрипт-вірусів в комп'ютерних системах.

Література

1. McAfee Antivirus Solution, "Top 10 malicious programs for PC", 2018. URL: <http://www.securelist.com>
2. Webroot, "Insights from Collective Threat Intelligence", Tech. Rep. April, 2018.
3. R. Perdisci, A. Lanzi, and W. Lee, "McBoost: Boosting Scalability in Malware Collection and Analysis using Statistical Classification of Executables", 2016, pp. 301–310.
4. S. M. Tabish, M. Z. Shafiq, and M. Farooq, "Malware Detection using Statistical Analysis of Byte-Level File Content", CSI-KDD '09 Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics, pp. 23–31, 2011.
5. G. Jacob, P. M. Comparetti, M. Neugschwandtner, C. Kruegel, and G. Vigna, "A Static, Packer-Agnostic Filter to Detect Similar Malware samples", vol. 7591 LNCS, pp. 102–122, 2013.
6. LeDoux and A. Lakhotia, "Malware and machine learning", in Intelligent Methods for Cyber Warfare, 2015.
7. M. A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, and N. Provos, "CAMP: Content-Agnostic Malware Protection", NDSS, 2013.
8. C. Smutz, "GMU-TR-2015-11 Discerning Machine Learning Degradation via Ensemble Classifier Mutual Agreement Analysis," George Mason University, Tech. Rep., 2015.
9. P. Trinius, C. Willems, T. Holz, and K. Rieck, "A Malware Instruction Set for Behavior-Based Analysis", Sicherheit Schutz und Zuverl"assigkeit SICHERHEIT, pp. 1–11, 2011.
10. J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features", in Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), 2015, pp. 11–20.
11. O. Atan, Y. Andreopoulos, C. Tekin, and M. van der Schaar, "Bandit framework for systematic learning in wireless video-based face recognition", IEEE Journal of Selected Topics in Signal Processing, vol. 9, no. 1, pp. 180–194, Feb 2015.
12. P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem", Mach. Learn., vol. 47, no. 2-3, pp. 235–256, May 2002.
13. L. Li, W. Chu, J. Langford, and R. E. Schapire, "A Contextual-Bandit Approach to Personalized News Article Recommendation", 2010, p. 10, 2010.
14. "VirusTotal", URL: <https://www.virustotal.com>.

Рецензія/Peer review : 08.05.2018 р.

Надрукована/Printed : 12.07.2018 р.

Стаття рецензована редакційною колегією