УДК 621.391 160164
DOI: 10.31891/CSIT-2020-1-8

НІЧЕПОРУК А. О., НІЧЕПОРУК А. А.,
НЕГА І. А., НІЧЕПОРУК Ю. О., КАЗАНЦЕВ А. Д.
Хмельницький національний університет

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ВИЯВЛЕННЯ МЕТАМОРФНИХ ВІРУСІВ НА ОСНОВІ АНАЛІЗУ ПОВЕДІНКИ ДОДАКІВ В КОРПОРАТИВНІЙ МЕРЕЖІ

В роботі запропоновано інформаційну технологію виявлення метаморфних вірусів на основі аналізу поведінки додатків у корпоративній мережі. Процес виявлення здійснюється на основі аналізу API викликів, що описують потенційно небезпечну поведінку програмного додатку. Після встановлення факту підозрілості поведінки додатку здійснюється порівняння дизасембльованого коду функціональних блоків підозрілого додатку з кодом функціональних блоків його зміненої версії. Для створення зміненої версії програмного додатку на хостах мережі встановлюються модифіковані емулятори. З метою підвищення загальної ефективності виявлення метаморфних вірусів, інформаційна технологія передбачає пошук відповідності між функціональними блоками метаморфного вірусу та його зміненої версії. Для формування висновку про схожість підозрілої програми на метаморфних вірус використовується система нечіткого логічного висновку. При недостатньому прояві шкідливої поведінки та з метою підвищення рівня достовірності для виявлення метаморфного вірусу залучаються інші хости мережі.

Ключові слова: метаморфний вірус, мутація, еквіваленті функціональні блоки

NICHEPORUK A., NICHEPORUK A.,
NEGA I., NICHEPORUK Y., KAZANTSEV A.
Khmelnytskyi National University

INFORMATION TECHNOLOGY FOR DETECTING METAMORPHIC VIRUSES BASED ON THE ANALYSIS OF THE BEHAVIOR OF APPLICATIONS IN THE CORPORATE NETWORK

The problem of cybercrime is one of the greatest threats to the modern information world. Among a wide range of different types of malware, the leading place is occupied by viral programs that use mutations of their own software code, ie polymorphic and metamorphic viruses. The purpose of transforming your own code is for attackers to try to make their previous malware different (in terms of syntax, not in terms of semantics) with each new infection. According to a study conducted by Webroot in 2018, about 94% of all malware performs mutations in their software code. In addition, the problem of the prevalence of mutated software is complicated by the availability of free access to metamorphic generators, which allows you to import into malware metamorphic component. Therefore, the relevance of the development of new methods and information technologies focused on the detection of polymorphic and metamorphic software leaves no doubt.

The paper proposed the information technology for detecting metamorphic viruses based on the analysis of the behavior of applications in the corporate network. The detection process is based on the analysis of API calls that describe the potentially dangerous behavior of the software application. After establishing the fact of suspicious behavior of the application, the disassembled code of the functional blocks of the suspicious application is compared with the code of the functional blocks of its modified version. Modified emulators are installed on network hosts to create a modified version of the software application. In order to increase the overall efficiency of detection of metamorphic viruses, information technology involves searching a match between the functional blocks of the metamorphic virus and its modified version. A fuzzy inference system is used to form a conclusion about the similarity of a suspicious program to a metamorphic virus. In case of insufficient manifestation of harmful behavior and in order to increase the level of reliability for the detection of metamorphic virus, other network hosts are involved.

Keywords: metamorphic virus, mutation, equivalent functional blocks

Вступ. Проблема кібрезлочинності є однією із найбільших загроз сучасному іфнормаційному світові. Серед широкого кола різного виду шкідливого програмного забезпечення провідне місце посідають вірусні програми, що виконують мутацію власного програмного коду, тобто поліморфні та метаморфні віруси. Метою трансформації власного коду є намагання зловмисників зробити їхнє шкідливе програмне забезпечення різним (у плані синтаксису, а не в плані семантики) при кожному новому інфікуванні. Згідно дослідження компінї Webroot, що проведене у 2018 році, близько 94% всього шкідливого програмного забезпечення виконує мутацію свого програмного коду [1]. Окрім того, проблему розповсюдженості мутованого програмного забезпечення ускладнює наявність у вільному доступі метаморфних генераторів, що дозволяє імпортувати у шкідливе програмне забезпечення метаморфну складову. Тому актуальність розробки нових методів та інформаційних технологій, орієнтованих на виявлення поліморфного та метаморфного прогармного забезпечення не залишає сумнівів.

Огляд попередніх досліджень. Сьогодні значна увага приділяється проблемі виявлення метаморфних вірусів. Для виявлення метаморфних вірусів застосовується ряд технік, зокрема: виявлення на основі опкодів інструкцій [2, 3], аналіз потоку інформації [4] та потоку керування [5].

У роботі [2] запропоновано підхід виявлення метаморфних вірусів на основі прихованих марківських моделей (HMM). Основна мета дослідження показати як відрізняється код згенерований в ручному режимі (hand written assembly) від скомпільованого коду та як відрізняється код корисних програм від шкідливого коду. З цією метою були побудовані приховані марківські моделі для корисних програм та метаморфних копій, з використанням метаморфних генераторів NGVCK та MetaPHOR. Для кожної програми імовірність спостереження послідовності опкодів визначається кожною HMM's. Якщо HMM сигналізує про найбільшу

імовірність присутності вірусного коду, тоді програма маркується як вірус. Проте, у разі вставки фрагментів коду корисних додатків у тіло метаморфних вірусів, такий підхід призводить до значних хибних спрацювань [6].

Метод виявлення метаморфних вірусів на основі графу потоку управління, що здійснює аналіз шаблонів в семантиці вірусних сімейств представлено у роботі [5]. Головна ідея методу полягає у формуванні вектора ознак на основі графу системних API викликів. Вектор ознак для кожного програмного графу будується на основі номеру API, як елементу даних та номера ребра як ознаки. Для формування висновку здійснюється навчання різних класифікаторів, серед яких найкращий результат 91,3% продемонстрував Random Forest з рівнем хибних спрацювань в 30,9 %. Тобто, приблизно кожний третій досліджуваний об'єкт буде помилково віднесений до іншого класу.

У роботі [3] виявлення метаморфних вірусів здійснюється на основі побудови гістограми частот інструкції. З цією метою використано найпоширеніші інструкцій асемблерно коду, на основі дослідження частоти появи опкодів у вірусних програмах. Після отримання лістингу інструкцій виконується їх нормалізація, що визначається як співвідношення і-тої інструкції до загальної кількості інструкцій Сформовані гістограми порівнюються на основі Евклідової відстані з отриманням відповідної матриці відстаней для кожної гістограми. Задача класифікації метаморфного вірусу вирішується за допомогою методу опорних векторів. Однак, при використанні метаморфними вірусами переміщення блоків частота появи інструкцій в зміненій версії метаморфного вірусу не зміниться.

Метод виявлення метаморфних вірусів на основі аналізу інформаційних потоків представлений у роботі [4]. В своїй роботі автори використовують метод виявлення, що ґрунтується на аналізі значень програми в процесі її виконання. Підхід передбачає, що кожна програма в момент виконання може бути представлена набором значень комірок пам'яті та регістрів. Невідома програма розміщується в захищене середовище, після чого для кожного API виклику здійснюється аналіз стану регістрів до та після API виклику. На основі зміни значень в регістрах формується вектор ознак для кожного регістру. Запропонований підхід показав високу ефективність, з відсотком хибних спрацювань на рівні 2,9%. Проте, представлений метод не враховує технік ухилення від емуляції, використання яких, може значно знизити відсоток виявлення.

Проведений аналіз відомих методів виявлення метаморфних вірусів показав недостатньо високу їх ефективність.

Зважаючи на те, що на сучасному етапі розвитку технологій приховування та заплутування програмного коду, для виявлення метаморфних вірусів більшість антивірусних засобів використовують метод емуляції виконання програмного коду, метаморфні віруси, інтегрують у свою структуру механізми ухилення від емуляції, що унеможливлює їх діагностування на належному рівні.

Зважаючи на це, актуальним завданням є розробка інформаційної технології виявлення метаморфних вірусів, з використанням модифікованих емуляторів, що дозволить здійснити виявлення нових та копій вже існуючих метаморфних вірусів, враховуючи технології ухилення від емуляції.

Інформаціна технологія виявлення метаморфних вірусів на основі аналізу поведніки додатків в корпоративній мережі

Наведемо основні кроки функціонування інформаційної технології виявлення метаморфних вірусів в корпоративній мережі:

1. Перевірка хостом, що отримав невідому програму аналізатором підозрілості. Перевірка здійснюється на основі евристичних правил в основу яких покладено API виклики, що здійснює програма;
2. Якщо в процесі перевірки невідома програма буде визначена аналізатором підозрілості як *suspicious,* то здійснюється запит до сервера на предмет наявності поведінкової сигнатури для даної невідомої програми;
3. На основі наявної на сервері бази потенційно небезпечних поведінок, здійснюється пошук відповідної поведінки для підозрілої програми. Пошук ведеться в чорному та білому списку. Якщо відповідна поведінка присутня в чорному списку, тоді підозріла програма блокується; у разі наявності відповідної поведінки в білому списку – підозріла програма продовжує власне виконання;
4. Відправлення відповіді хосту, що отримав підозрілу програму; перевірка підозрілої програми на наявність метаморфного вірусу (кроки 5-11);
5. Виконання хостом, що отримав підозрілу програму, її дизасемблювання та отримання зразка коду F_p; розбиття на функціональні блоки зразка коду F_p та отримання множини функціональних блоків для F_p;
6. На основі відстеження API викликів формування поведінки підозрілої програми в модифікованому емуляторі; додавання до бази потенційно небезпечних поведінок сформованої поведінки;
7. Виконання емуляції виконання підозрілої програми в модифікованому емуляторі та отримання зміненого зразка коду F_s; дизасемблювання зміненого зразка коду F_s, розбиття його на функціональні блоки та отримання множини функціональних блоків для F_s;

8. Визначення еквівалентних функціональних блоків для зразків коду F_p та F_s;

9. На основі попарного порівняння еквівалентних функціональних блоків, з використанням алгоритму Вагнера-Фішера, формування матриці векторів ознак схожості копій метаморфних вірусів на основі дистанції Дамерау-Левенштейна; нормування матриці векторів ознак схожості копій метаморфних вірусів;

10. Відправлення матриці векторів ознак схожості копій метаморфних вірусів на сервер для класифікації;

11. Розсилка в захищеному контейнері підозрілої програми на решту хостів в мережі; для кожного хоста в мережі повторити пункти 5,7-9;

12. З використанням системи нечіткого логічного висновку формування висновку про належність підозрілої програми до одного з класів метаморфних вірусів та відправлення інформації на хост, що отримав підозрілу програму; маркування поведінки підозрілої програми, що отримана на кроці 6 як *Class_Metamorphic* та додавання поведінки підозрілої програми у чорний список; блокування підозрілої програми, що була отримана хостом.

Схему інформаційних потоків системи виявлення метаморфних вірусів в корпоративній мережі наведено на рис. 1.

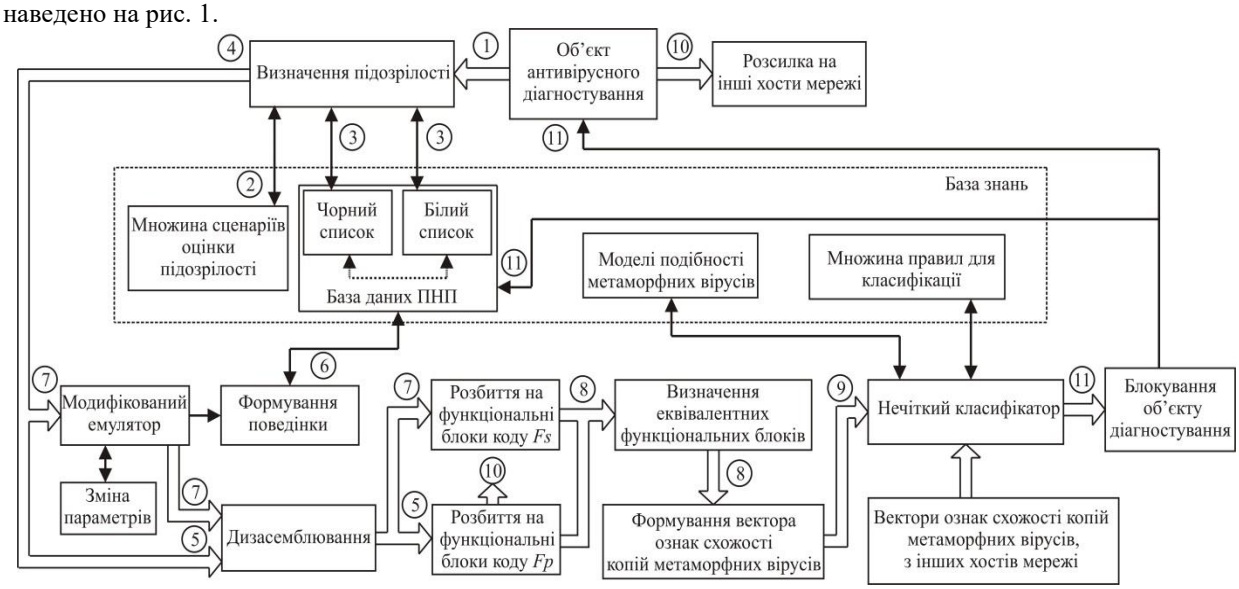

Рис. 1. Схема інформаційних технології

Розглянемо детальніше основні складові модулі запропонованої інформаційної технології.

Аналізатор підозрілості програми та формування поведінки невідомої програми

З метою виявлення підозрілих дій на кожному хості корпоративної мережі використовується аналізатор підозрілості програми. Основним його завданням є відстеження потоку API викликів, що здійснюються в процесі виконання невідомою програмою. Використання API викликів в якості основи для попереднього аналізу програми зумовлено тим, що API виклики є єдиним механізмом взаємодії програми та операційної системи. У випадку застосування до програми технік обфускації програмного коду, її API виклики залишаються незмінними, видозмінюються лише параметри та значення, що повертається відповідною функцією [7].

Ґрунтуючись на особливостях функціонування метаморфних вірусів, тобто пошук та впровадження вірусу у файл, виокремимо групи небезпечних API викликів (табл.1).

Таблиця 1

Групи поведінок для формування сценаріїв визначення підозрілості

Групи поведінки	Елемент впливу	API виклики
1	Запис у файл	CreateFile, OpenFile, WriteFile, CloseHandle, CreateThread
2	Доступ до системного реєстру	RegOpenKey,RegCreateKey,RegSetValue,RegCloseKey
3	Доступ до директорій	GetWindowsDirectory,GetSystemDirectory
4	Робота з відображенням файлу	CreateFileMapping,MapViewOfFile,UnMapViewOfFile
5	Модифікація атрибутів файлу	GetFileAttributes,SetFileAttributes
6	Доступ до системного часу	GetFileTime,SetFileTime
7	Пошук файлів	FindFirstFile,FindNextFile,FindClose, GetProcessAddress
8	Розподіл глобальної пам'яті	GlobalAlloc,GlobalFree
9	Розподіл віртуальної пам'яті	VirtualAlloc,VirtualFree
10	Створення з'єднання з Internet	Socket, connect

Кожна окрема дія, що виконується підозрілою програмою, не є небезпечною. Проте, виконання певної послідовності таких дій може свідчити про можливу небезпеку інфікування метаморфних вірусом. Наприклад, виклик API функція Socket, connect, GetSystemDirectory може свідчити про потенційну небезпеку, що проявляється у створенні нового з'єднання в результаті якого відбувається доступ до системної директорії.

З цією метою в основі аналізатора підозрілості програми закладено базу евристичних сценаріїв, що згруповані по рівням підозрілості – *High, Medium* та *Low*. Рівень підозрілості визначає умову при якій відбувається блокування програми. Наприклад, при заданому рівні підозрілості *High* один з сценаріїв матиме вигляд: *High: Socket→Connect→ GetSystemDirectory*, а у випадку встановлення рівня *Low* – *Low: Socket→Connect*

У випадку збудження програмою одного із правил відбувається припинення виконання невідомої програми, маркування її як *suspicious* та здійснення запиту до сервера на предмет наявності поведінкової сигнатури для даної невідомої програми.

На основі наявної на сервері бази потенційно небезпечних поведінок, здійснюється пошук відповідної поведінки (згідно активізації відповідного сценарію в аналізаторі підозрілості програми) для підозрілої програми.

В результаті, хосту, що отримав підозрілу програму, може бути надісланий один з трьох результатів:

- блокувати підозрілу програму – якщо відповідна поведінка присутня в чорному списку;
- дозволити виконання – якщо відповідна поведінка присутня в білому списку;
- формувати поведінку підозрілої програми.

Формування поведінки підозрілої програми, як і визначення її підозрілості, здійснюється на основі відслідковування API викликів хостом, що отримав підозрілу програму в модифікованому емуляторі.

З цією метою виокремлено групи потенційно небезпечних поведінкових (ПНП) дій виконуваного файлу, що виконується в системі. Групування небезпечних поведінкових дій здійснюється на основі впливу на відповідний компонент системи. Кожна група ПНП складається з множини ознак, що представляються у вигляді API функції:

1. Множина функцій роботи з файлами $\Phi = \{\varphi_1, \varphi_2, .., \varphi_n\}$, де $n = |\Phi|$. Здійснення будь-якої дії по відношенню до файлів, тобто створення, пошук, відкриття, видалення.

2. Множина API функцій, що здійснюють перевірку виконання в захищеному середовищі $Z = \{\xi_1, \xi_2, ..., \xi_k\}$, де $k = |Z|$. Наприклад перевірка ключа системного реєстру, що визначає наявність віртуальної машини.

3. Множина функцій необхідних для здійснення встановлення нових компонентів в: $\Psi = \{\psi_1, \psi_2, ..., \psi_l\}$, де $l = |\Psi|$ (перезавантаження КС, зміну прав доступу, додавання ключів в системний реєстр, створення закладок, тощо).

4. Множина функцій, що передбачають доступ до мережі Інтернет: $I = \{\iota_1, \iota_2, ..., \iota_b\}$, де $b = |I|$ (здійснення відвідування URL посилань, створення RPC каналів, відкриття/закриття системних портів).

5. Множина дій з процесами та потоками $\Pi = \{\pi_1, \pi_2, ..., \pi_u\}$, де $u = |\Pi|$. Послідовність дій, що маніпулюють процесами та потоками, наприклад, створення багатопочності.

6. Множина API викликів, що здійснюють визначення інформації системи. $\Xi = \{\zeta_1, \zeta_2, ..., \zeta_m\}$, де $m = |\Xi|$. Перелік дій, для визначення версії ОС, архітектури CPU (x86 або x64), історії web браузера, системного часу та дати.

Представимо поведінку програми у вигляді орієнтованого графа $G = (V, Q)$, вершинами якого виступають групи ПНП, а дуги відображають стани підозрілої програми при виклику API функції (рис. 2).

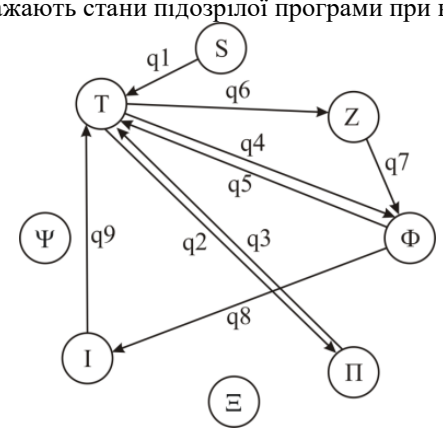

Рис. 2. Приклад Фрагменту графу викликів API функцій метаморфним вірусом

У наведеного графу вершина S визначає початок виконання підозрілої програми, та у низькорівневому представлені відповідає точці входу у програму, вершина T – визначає виклик легітимної API функції, тобто $T = \{\forall a, a \in A \mid a \notin W\}$, де $A = \{a_1, a_2 ..., a_k\}$ – множина API викликів, що здійсню підозріла програма; $W = \{\Phi \cup Z \cup \Psi \cup I \cup \Pi \cup \Xi\}$ – множина потенційно небезпечних API викликів.

Для формування та додавання поведінки метаморфного вірусу у базу ПНП, представимо граф викликів API функцій метаморфного вірусу у вигляді поведінкової матриці викликів API функцій (табл. 2). Поведінкова матриця складається з двох частин: верхня частина визначає матрицю інцидентності для графу підозрілої програми $G = (V, Q)$, нижня – визначає значення τ_T та τ_{W_j}.

Значення τ_T визначає кількість послідовних API викликів з множини T, а значення τ_{W_j} – кількість послідовних API викликів з *j*-тої множини W.

Таблиця 2

Поведінкова матриця API викликів

	q1	q2	q3	q4	q5	q6	q7	q8	q9
S	-1	0	0	0	0	0	0	0	0
Z	0	0	0	0	0	+1	-1	0	0
Φ	0	0	0	+1	-1	0	+1	-1	0
Π	0	+1	-1	0	0	0	0	0	0
Ξ	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	+1	-1
Ψ	0	0	0	0	0	0	0	0	0
T	+1	-1	+1	-1	+1	-1	0	0	+1
τ_T	5	0	3	0	7	0	0	0	2
τ_{W_j}	0	1	0	3	0	4	1	2	0

Таким чином, отримана поведінка додається до бази ПНП, причому приналежність даної поведінки до одного із класів метаморфних вірусів чи довірених додатків на даному етапі невідома.

Виначення еквівалентних функціональних блоків між зразками коду до та після емуляції

Після отримання сервером дизасембльованого лістингу опкодів для програми до F_P та після F_S емуляції, здійснюється визначення еквівалентних функціональних блоків (ЕФБ) між ними.

Процес визначення ЕФБ складається з двох етапів. Перший етап визначає на основі обчислення статистичної оцінки появи інструкцій у блоці. На другому етапі здійснюється уточнення ЕФБ та вибір найбільш підходящого блоку, що буде використаний для формування векторів ознак схожості копій метаморфних вірусів.

Розбиття на ФБ здійснюється за інструкціями умовного або безумовного переходу. Для пошуку ЕФБ використаємо метрику TF-IDF, яка застосовуватиметься до кожного окремого ФБ для програм F_p та F_s:

$$s_{FB} = \frac{n_i}{\sum_k n_i} * \log(\frac{N + 1.0}{n_j}) \quad (1)$$

де n_i – кількість входжень *i*-го опкоду у ФБ; $k = \overline{1, k_a}$ – кількість опкодів у ФБ, де k_a – загальна

кількість асемблерних інструкцій; N – загальна кількість ФБ, причому $N_{F_p} \neq N_{F_S}$; n_j – кількість ФБ в якому присутній i-й опкод.

В результаті виконання даного етапу кожний ФБ з F_p та F_s буде представлений у вигляді матриці, рядки яких визначають ФБ програми, а стовпці – опкоди, що присутні в ФБ. Кожна комірка матриці визначає оцінку появи i-го опкода в j-му ФБ.

З метою визначення ЕФБ, на наступному кроці обчислюється оцінка схожості двох ФБ з програми F_P та програми F_S з використанням квадрату Евклідової метрики:

$$E\left(FB_i^{Fp}, FB_j^{Fs}\right) = \sum_{i=0,\, j=0}^{k} (s_i - s_j)^2, \tag{2}$$

де s_i – оцінка появи опкодів в i-тому блоці програми F_P, s_j – оцінка появи опкодів в j-тому ФБ програми F_S, FB_i^{Fp} – i-тий ФБ з програми F_P, FB_j^{Fs} – j-тий ФБ з програми F_S.

Якщо значення оцінки схожості двох ФБ менше визначеного порогового значення δ, то виконується повторне обчислення оцінки схожості для ФБ з програми FB_i^{Fp} та наступного блоку, що слідує за блоком FB_j^{Fs}, тобто $E(FB_i^{Fp}, FB_j^{Fs} + FB_{j+1}^{Fs})$. Зазначені вище дії повторюються поки значення оцінки схожості буде менше або рівне порогового значення.

На наступному етапі вирішується задача уточнення еквівалентних блоків, що передбачає вибір найбільш підходящого блоку на основі формування та порівняння матриці імовірності слідування опкодів, що дозволить однозначно поставити у відповідність ФБ з програми до емуляції з ФБ після емуляції.

Вектор ознак схожості копій метаморфних вірусів

Після отримання пар ЕФБ, наступним етапом є попарне їх порівняння з використанням метрики Дамерау-Левенштейна та формування матриці векторів ознак схожості копій метаморфних вірусів. Для кожної пари ЕФБ подамо вектор ознак наступним чином:

$$\overline{S} = \langle L, T, D, I, R, M \rangle \tag{3}$$

де L – відстань Дамерау – Левенштейна для функціонального блоку між програмою до та після емуляції; T – кількість необхідних операцій обміну опкодів; D – кількість необхідних операцій видалення опкоду; I – кількість необхідних операцій вставки опкоду; R – кількість необхідних операцій заміни відповідних опкодів; M – кількість співпадінь між опкодами в функціональних блоках програми до та після емуляції.

Модифіковані емулятори

З метою уникнення розпізнання вірусною програмою виконання у віртуальному середовищі створюються модифіковані емулятори, що створюються на кожному хості у мережі [8].

Модифікованим емулятором будемо називати віртуальне середовище, в якому здійснюється емуляція виконання підозрілої програми з метою отримання зміненої її версії.

Основними функціями модифікованих емуляторів є забезпечення емуляції підозрілої програми та її дизасемблювання (копії), дизасемблювання підозрілої програми, формування поведінки підозрілої програми та занесення її у базу ПНП.

Для забезпечення змінного середовища виконання в модифікованих емуляторах відбувається зміна наступних параметрів: тип операційної системи, її розрядність, MAC адреса гостьової ОС, приховування процесу виконання модифікованого емулятора та захист процесу від читання/запису в хостовій ОС, відключення модуля обміну даними між віртуальною машиною та хостовою ОС (Virtual Machine Communication Interface), зміна параметру ключа реєстру в хостовій ОС HKEY_LOCAL_ MACHINE \SYSTEM\ControlSet001\Services\Disk\Enum, зміна конфігураційного файлу та встановлення правил.

Також, з метою успішного розпізнання та завершення процесу емуляції, інструкції, що не представлені у множині асемблерних команд замінюються на інструкцію NOP. Така модифікація дозволить здійснити емуляцію програми навіть якщо зустрінеться невідома команда і, відповідно, не буде здійснено аварійне завершення програми емуляції. Після завершення процесу емуляції, у сформованому дизасембльованому файлі на місці NOP, буде міститися відповідна команда, емуляцію якої, віртуальний процесор не зміг здійснити.

Класифікація векторів ознак схожості копій метаморфних вірусів

Для формування висновку щодо інфікування системи метаморфним вірусом, отримнні вектори ознак схожості з кожного хоста підлягають класифікації шляхом залучення системи нечіткого логічного висновку.

Кожна підозріла програма може бути віднесена до одного з трьох класів метаморфних вірусів або до класу довірених додатків.

Система нечіткого логічного висновку оперує вхідними та вихідними лінгвістичними змінними . У якості вхідних лінгвістичних змінних приймемо: «ступінь схожості підозрілогї програми з її копією за

дистанцією Левенштейна» (*L*), «ступінь схожості підозрілогї програми з її копією за кількістю операцій вставки» (*I*), «ступінь схожості підозрілогї програми з її копією за кількістю операцій видалення» (*D*), «ступінь схожості підозрілогї програми з її копією за кількістю операцій заміни» (*R*), «ступінь схожості підозрілогї програми з її копією за кількістю операцій перестановки» (*T*), та «ступінь схожості підозрілогї програми з її копією за кількістю операцій співпадіння» (*M*) . Вихідною лінгвістичною змінною приймемо ступінь схожості з класом метаморфних вірусів. Для кожної лінвістичної змінної задано терм-множину *Low*, *Medium* та *High*. Для визначення приналежності метаморфного вірусу до одного із класів в системі задіяно 38 правил. В якості функцій приналежності для входів було обрано трапецевидну, для виходу – трикутну.

Після формування висновку здійснюється запит до хоста, який отримав підозрілу програму, що передбачає блокування або дозвіл на виконання підозрілої програми. Таким чином, сформований результат додається до поведінки, що міститься у базі ПНП та маркує відповідну поведінку як *Class_Metamorphic*, після чого здійснюється додавання отриманої поведінки до чорного (або білого – у випадку корисної програми) списку.

Експерименти

Для визначення ефективності запропонованої інформаційної технології було проведено ряд експериментів. З цією метою було залучено університетську мережу. Вона складається з 30 робочих станцій. На кожній станції було встановлено модифіковані емулятори на основі Qemu [9]. В якості дизасемблера в модифікованих емуляторах було використано IDA Pro [10].

З метою отримання змінених версій метаморфних вірусів взятих з [11] було використано метаморфні генератори 3 типів: NGVCK, VCL32 та G2. Всі метаморфні версії, що створювались за допомогою зазначених генераторів були скомпільовані з опціями anti-debugging та anti-emulation. Слід відзначити, що модулі інфікування у створених копій метаморфних вірусів були деактивовані. Загальна кількість тестових зразків складала 150 одиниць (по 50 зразків кожного типу). Всі згенеровані метаморфні віруси інсталювалися на хости корпоративної мережі. При проведенні експерименту порогові значення δ, було встановлено на рівні 0,5, 0,6 та 0,7.

Експерименти передбачали визначення ефективності виявлення метаморфних вірусів, визначення рівня False Positives and True Positives. Загальна ефективність виявлення визначалась як відношення суми вірно класифікованих як метаморфний вірус та корисний додаток до повної групи події. Результати проведеного експерименту наведено у таблиці 3.

Таблиця 3

Результати проведення експерименту

Тип	Кількість зразків	Порогове значення δ	Ефективність, %	Хибні спрацювання, %
NGVCK	50	0,5	64	4,1
		0,6	**79**	3,9
		0,7	73	**3,2**
VCL32	50	0,5	79	2,9
		0,6	**84**	3,5
		0,7	82	**3,8**
G2	50	0,5	91	1,2
		0,6	**95**	**1,2**
		0,7	94	1,3

Результати проведеного експерименту свідчать, що найвищий рівень ефективності виявлення досягається для метаморфних генераторів G2, і становить 95% з рівнем хибних спрацювань 1,2% . В той час найнижчу ефективність виявлення (79%) було зафіксовано для метаморфних генераторів NGVCK. Слід відзначити, що для всіх метаморфних генераторів, що були залучені для проведення експерименту, значення δ 0,6 є оптимальним при порівнянні функціональних блоків між програмами до та після емуляції.

Висновки. В роботі запропоновано інформаційну технологію виявлення метаморфних вірусів на основі аналізу поведінки додатків у корпоративній мережі.

Процес виявлення здійснюється на основі відстеження API викликів, що описують потенційно небезпечну поведінку метаморфного вірусу, та порівнянні дизасембльованого коду функціональних блоків метаморфного вірусу з кодом функціональних блоків його зміненої версії. Для створення зміненої версії метаморфного вірусу на хостах мережі встановлюються модифіковані емулятори. З метою підвищення загальної ефективності виявлення метаморфних вірусів, метод передбачає пошук відповідності між функціональними блоками метаморфного вірусу та його зміненої версії. Для формування висновку про схожість підозрілої програми на метаморфних вірус використовується система нечіткого логічного висновку. У випадку недостатнього прояву шкідливої поведінки та підвищення рівня достовірності для виявлення метаморфного вірусу залучаються інші хости мережі.

Експериментальним шляхом було доведено, що рівень прояву метаморфних властивостей зростає при збільшенні кількості хостів в мережі. Окрім того, вибір еквівалентних функціональних блоків для порівняння дозволяє зменшити рівень хибних спрацювань порівняно з попереднім дослідженням. Критичним є вибір порогового значення при визначенні еквівалентних функціональних блоків, що прямо

впливає на ефективність виявлення метаморфних вірусів. В ході дослідження було виявлено, що при значенні поргової оцінки 0,6 досягається найвищий рівень виявлення метаморфних вірусів. При такому значенні загальна ефективність виявлення для метаморфного сімейства NGVCK становить 79%.

Література

1. Webroot 2018. Threat report [Online]: URL https://www-cdn.webroot.com/9315/2354/6488/2018-Webroot-Threat-Report_US-ONLINE.pdf

2. Austin T. H. Exploring Hidden Markov Models for Virus Analysis: A Semantic Approach / T. H. Austin, E. Filiol, S. Josse, M. Stamp // In Proc. of the 2013 46th Hawaii International Conference on System Sciences. – 2013, pp. 5039-5048.

3. Mahawer D. K. Metamorphic malware detection using base malware identification approach / D. K. Mahawer, A. Nagaraju Journal Security and Communication Networks. – 2014. – vol. 7. – pp. 1719-1733.

4. Ghiasi M. Dynamic malware detection using registers values set analysis / M. Ghiasi, A. Sami, Z. Salehi // Information Security and Cryptology. – 2012. – pp. 54-59.

5. Eskandari M. A graph mining approach for detection unknown malwares / M. Eskandari., S. Hashemi // Journal of Visual Languages & Computing. – 2012. – vol. 23(2). – pp. 154-162.

6. Tamboli T. Metamorphic code generation from LLVM bytecode / T. Tamboli, T.H. Austin, M. Stamp // Journal of Computer Virology and Hacking Techniques. 2013. – vol. 10(3). – pp. 177–187.

7. Patanaik C. K. Obfuscated malware detection using API call dependency / C. K. Patanaik, F.A. Barbhuiya, S. Nandi // In Proc. the First International Conference on Security of Internet of Things. – 2012. – pp. 185-193.

8. Нічепорук А.О. Використання модифікованих емуляторів для виявлення метаморфних вірусів в корпоративній мережі / А. О. Нічепорук, Ю. О. Нічепорук, Б. О. Савенко, М. В. Стецюк // Вісник Хмельницького національного університету. Технічні науки. – 2017. – № 2. – С.199-206.

9. Qemu. Open source processor emulator [online]: URL http://wiki.qemu.org/Main_Page

10. Hex-Rays, S.A.:IDA Pro 5.5 [online]: URL https://www.hex-rays.com/products/ida/

11. VX Heavens [online]: URL http://vxheaven.org/

Надійшла / Paper received: 04.08.2020
Надрукована / Paper Printed : 02.09.2020