УДК 621.391 160164

А.О. НІЧЕПОРУК, Ю.О. НІЧЕПОРУК, Б.О. САВЕНКО, М.В. СТЕЦЮК
Хмельницький національний університет

ВИКОРИСТАННЯ МОДИФІКОВАНИХ ЕМУЛЯТОРІВ ДЛЯ ВИЯВЛЕННЯ МЕТАМОРФНИХ ВІРУСІВ В КОРПОРАТИВНІЙ МЕРЕЖІ

В роботі здійснено аналіз антиемуляційних технологій, що використовуються для виявлення штучного середовища метаморфними вірусами, розроблена їх класифікація. Ґрунтуючись на особливостях застосування антиемуляційних технологій розроблено її модель, що дозволяє здійснити визначення основних вразливостей віртуального середовища. На основі моделі застосування антиемуляційних технологій сформовано набір правил та налаштувань для модифікованих емуляторів, які використовуються для виявлення метаморфних вірусів в корпоративній мережі.

Ключові слова: антиемуляційні техніки, модифікований емулятор, змінена версія вірусного коду.

A.O. NICHEPORUK, Y.O. NICHEPORUK, B.O. SAVENKO, M.V. STETSYUK
Khmelnytskyi National University

INVOLVEMENT OF MODIFIED EMULATORS FOR METAMORPHIC VIRUS DETECTION IN CORPORATE NETWORK

The paper analyzes anti evasion techniques which are used for detection artificial environment by metamorphic viruses. Improved classification of the anti evasion technologies used metamorphic viruses. Based on the features of the application anti evasion technologies is developed a model which allows identification of the main vulnerabilities of the virtual environment. Based on the model of the application anti evasion technologies a set of rules and settings for modified emulators are formed. To detect metamorphic viruses modified emulators are placed on each host in the corporate network, this allowing to increase of the level of metamorphic viruses' demonstrations. Directly detection of metamorphic viruses is based on the comparison of metamorphic virus with its version, which was formed in the modified host emulators on the network.

Keywords: anti emulation techniques, modified emulator, modified version of the virus code.

Вступ

Сучасний розвиток технологій написання та впровадження вірусного коду не дозволяє здійснювати його виявлення простими сигнатурними методами. Для виявлення поліморфних та метаморфних вірусів застосовується метод емуляції виконання програмного коду. Проте, на сьогоднішній день шкідливе програмне забезпечення розширює та урізноманітнює набір інструментів та засобів для ухилення виявлення у віртуальному середовищі. Так, у 2015 році кількість обходів вірусними програмами віртуального середовища зросло на 2000% у порівнянні з 2014 роком [1], що свідчить про загрозливу тенденцію.

Тому, при розробці нових методів виявлення метаморфних вірусів, що використовують віртуальне середовище, доцільно здійснювати врахування технологій та засобів ухилення від емуляції.

Попередні дослідження

Для дослідження прояву шкідливого коду та захисту реальної робочої станції широко використовуються загальнодоступні віртуальні середовища, зокрема Anubis, Qemu, Norton SandBox та інші. Проте, фактор загальнодоступності робить їх вразливими перед розробниками нового шкідливого коду, оскільки зловмисники здатні розробляти алгоритми та процедури перевірки безпосередньо під виявлення конкретного віртуального середовища. Ґрунтуючись на проектах віртуальних машин з відкритим доступом було розроблено ряд методів використання емуляторів, що здатні протидіяти антиемуляційним технологіям.

Один з підходів забезпечення ухилення від ізольованого програмного середовища – створення прозорого середовища, що не відрізняється від реального середовища виконання додатків [2]. Проте, автори роботи [3] доводять твердження про неможливість створення абсолютно ідентичного реальному середовищу, особливо, якщо програмний код має доступ до мережі Internet (здійснення запиту до віддаленого ресурсу, наприклад, для визначення часу). Окрім того, підхід [2] не здатний протидіяти технологіям ухиленням від емуляції на основі перевірки часу виконання інструкцій.

У роботі [4] запропоновано метод виявлення шкідливого коду, що здатен здійснювати виявлення віртуального середовища. З цією метою автори використовують множину пісочниць, що дозволяє отримати різні поведінкові шаблони, отримані в різних середовищах. В роботі запропоновано алгоритми для нормалізації та порівняння поведінкових шаблонів, що дозволяє здійснити визначення частини коду, що відповідають за ідентифікацію віртуального середовища і, які демонструють семантично різну поведінку.

Модель застосування антиемуляційних технологій вірусними програмами

З метою отримання змінного середовища виконання при аналізі метаморфних вірусів, важливим етапом є дослідження та побудова моделей антиемуляційних технологій. У роботі [5] запропоновано класифікацію антиемуляційних технологій. Згідно із цією класифікацією всі технології ухилення поділяються на чотири групи, що визначають об'єкт перевірки: середовище, апаратне забезпечення, додаток та поведінка. Виокремимо ще одну групу, що визначає перевірку мережного з'єднання у віртуальній машині. Запропонована класифікація антиемуляційних технологій представлена на рис. 1.

Рис. 1. Класифікація технологій ухилення від емуляції

Для побудови моделі застосування антиемуляційних технологій вірусними програмами відобразимо предметну область у векторний простір [6]. Тоді, подамо модель застосування антиемуляційних технологій вірусними програмами у вигляді кортежу:

$$M_{aet} = \langle L, F, R \rangle, \tag{1}$$

де $L = \{l_i\}_{i=1}^{5}$ – множина об'єктів КС, що підлягають перевірці антиемуляційнимим технологіями;

$R = \{true, false\}$ – бінарна множина результату застосування антиемуляційних технологій;

$F = \{f_{i,j}\}_{i=1,j=1}^{N_F}$ – множина функцій для перевірки об'єкту КС, N_F – кількість функцій перевірки об'єкту.

Функція перевірки наявності dll бібліотек для функціонування віртуального середовища $l_1 \Rightarrow D_A \xrightarrow{f_{11}} \{d_{em} \mid d_{em} \in D\}$, де $D_A = \{d_{A_i}\}_{i=1}^{N_{D_A}}$ – множина встановлених в системі dll бібліотек, N_{D_A} – кількість dll бібліотек, $D = \{d_i\}_{i=1}^{N_D}$ – множина dll бібліотек для функціонування віртуального середовища, N_D – кількість dll бібліотек для функціонування віртуального середовища.

Функція перевірки розташування таблиці векторів переривань та інших системних структур даних $l_1 \Rightarrow A \xrightarrow{f_{12}} A'$, де $A = \{a_i\}_{i=0}^{0x3FFFFFFF}$ – множина доступних фізичних адрес.

Функція перевірки активних процесів, що відповідають програмам відлагоджувальникам, дизасемблерам $l_1 \Rightarrow P_A \xrightarrow{f_{13}} \{p_{em} \mid p_{em} \in P\}$, де $P_A = \{p_{A_i}\}_{i=1}^{N_{P_A}}$ – множина процесів гостьової ОС в стані виконання, $P = \{p_i\}_{i=1}^{N_P}$ – множина процесів відлагоджувальників та дизасемблерам.

Функція перевірки ID версії системи $l_1 \Rightarrow V \xrightarrow{f_{14}} V'$, де $V = \{v_i\}_{i=1}^{N_V}$ – множина ID версій системи.

Функція визначення часу виконання блоку коду за допомогою інструкцій типу rdpmc, rdtsc $l_5 \Rightarrow C \times M_r \xrightarrow{f_{51}} \{true, false\}$, де $C = \{c_i\}_{i=1}^{N_C}$ – множина команд виконуваної програми, $M_r = \{m_{r_i}\}_{i=1}^{N_{M_r}}$ – множина команд підрахунку тактів процесора.

Функція визначення часу виконання блоку коду за допомогою виклику API функцій GetTimeTick() $l_5 \Rightarrow C \times M_a \xrightarrow{f_{52}} \{true, false\}$, де $M_a = \{m_{a_i}\}_{i=1}^{N_{M_a}}$ – множина API викликів визначення часу з моменту запуску системи.

Функція перевірки маніпуляцій комп'ютерною мишею $l_5 \Rightarrow A_{api} \times B \xrightarrow{f_{53}} \{true, false\}$, де

$A_{api} = \{a_{api_i}\}_{i=1}^{N_{Aapi}}$ – множина API викликів для при роботі з комп'ютерною мишею, $B = \{b_i\}_{i=1}^{N_B}$ – множина можливих операцій з використанням комп'ютерної миші.

Сформована модель застосування антиемуляційних технологій вірусними програмами дозволяє здійснити визначення основних вразливостей віртуального середовища, врахування яких, підвищить загальну ефективність динамічних методів виявлення шкідливого програмного забезпечення з використанням емуляторів.

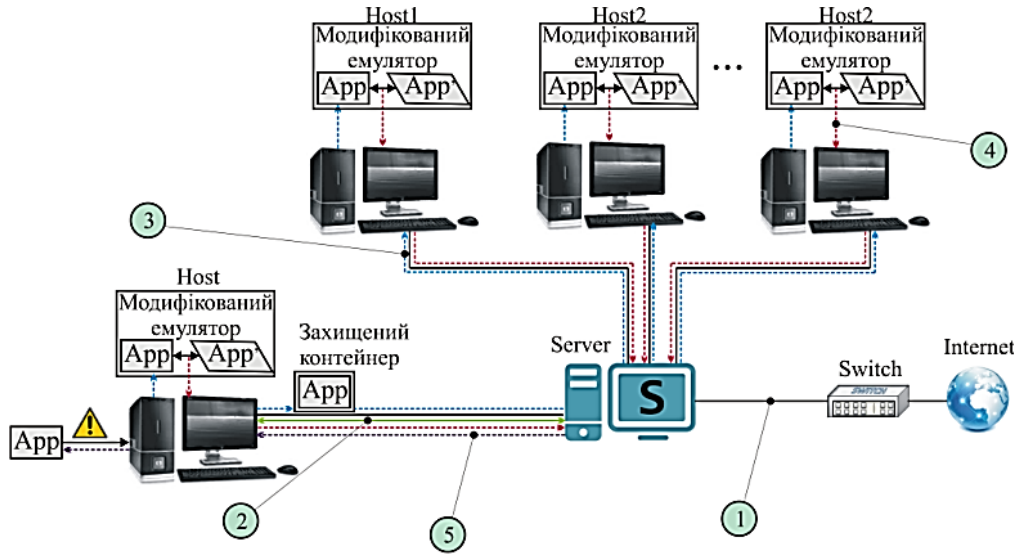
Технологія виявлення метаморфних вірусів з використанням модифікованих емуляторів в корпоративній мережі

З метою підвищення ефективності виявлення метаморфних вірусів запропоновано нову технологію, що заснована на аналізі поведінки програмного забезпечення на хості в корпоративній мережі [7].

Розглянемо основні кроки функціонування технології:

1. Перевірка підозрілості кожної нової програми на хості засобами аналізатора підозрілості програми. Перевірка здійснюється на основі евристичних правил, в основу яких покладено API виклики, що здійснює програма. Якщо в процесі перевірки ця програма буде визначена аналізатором підозрілості як suspicious, то здійснюється запит до сервера на предмет наявності поведінкової сигнатури для даної програми; на основі наявної на сервері бази потенційно небезпечних поведінок (ПНП), здійснюється пошук відповідної поведінки для підозрілої програми. Якщо відповідна поведінка присутня в чорному списку, тоді підозріла програма блокується; у разі наявності відповідної поведінки в білому списку – підозріла програма продовжує власне виконання; відсутність підозрілої поведінки в базі ПНП вимагає подальшого аналізу підозрілої програми.

2. Запуск на виконання програми, визначеної як підозріла, в середовищі модифікованого емулятора, який присутній на кожному хості; виконання дизасемблювання підозрілої програми та отримання зразка коду; здійснення емуляції виконання підозрілої програми та формування зміненого зразка коду, шляхом його дизасемблювання, на основі відстеження API викликів формування поведінки підозрілої програми. Відправлення на сервер для формування висновку щодо присутності метаморфного вірусу на хості зразків коду до та після емуляції (лістинги опкодів), підозрілої програми (в захищеному контейнері) та її поведінки (лістинг API викликів).



Інформаційні потоки в мережі:

1. фізичне з'єднання мережі;
2. запит до сервера на наявність поведінкової сигнатури; формування сигнатури;
3. розсилка та емуляція підозрілої програми на всіх хостах мережі;
4. передача вектору ознак схожості копій метаморфних вірусів на сервер;
5. отримання результату з сервера (блокування / дозвіл);

Рис. 2. Схема інформаційних потоків в мережі для технології виявлення метаморфних вірусів з використанням модифікованих емуляторів в корпоративній мережі

3. Опрацювання сервером отриманих результатів з хоста: розбиття отриманих з хоста зразків коду до та після емуляції на функціональні блоки, визначення еквівалентних функціональних блоків для зразків коду до та після емуляції; формування векторів ознак схожості зразків коду метаморфних вірусів для пар еквівалентних функціональних блоків зразків коду до та після емуляції на функціональні блоки; формування результату про ступінь подібності підозрілої програми до метаморфного вірусу на основі аналізу обфускації коду та поведінки з використанням нечіткого класифікатора. Якщо ступінь подібності до метаморфного вірусу має значення *High* то здійснюється блокування підозрілої програми на хості та додавання підозрілої

поведінки до чорного списку бази ПНП. Якщо ступінь подібності до метаморфного вірусу підозрілої програми отримав значення *Low* або *Medium,* то здійснюється розсилання підозрілої програми в захищеному контейнері на інші хости в мережі з метою їх запуску в модифікованих емуляторах.

4. Збір сервером інформації з хостів щодо поведінки, попередньо розісланої підозрілої програми: зразків коду до та після емуляції та поведінки. Здійснення нечітким класифікатором висновку щодо схожості підозрілої програми на метаморфних вірус. Якщо бодай на одному з хостів рівень схожості підозрілої програми на метаморфний вірус High, то здійснюється блокування підозрілої програми. Якщо рівень схожості – Low або Medium, то дозвіл виконання для цієї програми.

На рис. 2 представлено схему інформаційних потоків в мережі для технології виявлення метаморфних вірусів з використанням модифікованих емуляторів в корпоративній мережі.

Параметри та структура модифікованих емуляторів

В запропонованій технології виявлення метаморфних вірусів [7] критично важливим компонентом системи є модифіковані емулятори, що розміщуються на хостах мережі. Основною їх функцією є здійснення емуляції виконання підозрілої програми з метою отримання зміненої її версії. В процесі функціонування, метаморфні віруси створюють власну копію, забезпечуючи тим самим механізм власного поширення у комп'ютерній системі. У випадку розпізнання віртуального середовища метаморфним вірусом викликається функція очікування або переривання процесу інфікування [2], що унеможливлює отримання зміненої його версії. Тому, з огляду на розроблену модель застосування антиемуляційних технологій метаморфними вірусами, доцільно сформувати змінне середовище виконання.

Застосування методу емуляції полягає в імітації виконання аналізованого коду за допомогою емулятора – програмної моделі процесора і середовища виконання програм. Емулятор оперує з захищеною областю пам'яті (буфером емуляції). При цьому інструкції не передаються на центральний процесор для реального виконання. Якщо код, що обробляється емулятором, інфікований, то результатом його емуляції стане відновлення вихідного шкідливого коду, доступного для подальшого аналізу евристичними чи сигнатурними методами.

Роботу емулятора можна представити у вигляді циклічної послідовності:вибір і дизасемблювання чергової команди та моделювання виконання чергової команди.

Для здійснення процесу моделювання чергової команди виконується ряд етапів:

1) аналіз, визначення розміру та параметрів інструкції, що моделюється;

2) перерахунок адрес, повинен бути виконаний для всіх інструкцій, що виконують операції з пам'яттю;

3) перевірка адрес, тобто виявлення помилок адресації;

4) перевірка додаткових умов. Наприклад, при виконанні операції ділення необхідна перевірка дільника на предмет наявності значення 0;

5) виконання моделювання інструкції, здійснюється за рахунок зміни емулятором стану віртуальних регістрів, віртуального стеку, купи, буфера з кодом і даними програми та адрес у віртуальній пам'яті;

6) переведення вказівника віртуального регістра EIP на наступну інструкцію.

З метою визначення як код виконується центральним процесором змоделюємо роботу CPU за допомогою абстрактного автомата. Стан абстрактного автомата $s \in S$ складається з програмного лічильника pc, стану регістрів процесора R, стану пам'яті M та стану виконання E. Представимо стан абстрактного автомата кортежем:

$$s = \langle pc, R, M, E \rangle$$

Стан регістрів процесора R є повним відображенням регістрів процесора на їхнє значення. Стан пам'яті M є загальним відображенням $M : A \to [0...255]$ адрес пам'яті до однобайтного значення, де $A = [0...2^N - 1]$ – множина адрес пам'яті, і N кількість бітів процесора що використовується для адресації пам'яті. Програмний лічильник $pc \in A \cup \{halt\}$ може використовувати будь-яку адресу пам'яті; *halt* – спеціальна адреса, що використовується для завершення процесу виконання. З метою спрощення запропонованої моделі різниця між кодом та даними не враховується, тобто будь-яка ділянка пам'яті потенційно може бути виконана. Стан виконання $E \in \{\perp, illegal\ instruction, division\ by\ zero, general\ protection\ fault\}$ визначає виключні ситуації, що виникають в процесі виконання останньої інструкції; стан $\perp$ визначає нормальне виконання інструкції (без виключної ситуації).

Тоді, абстрактний автомат, що моделює центральний процесор буде транзитивна система (S, δ). Транзитивна функція стану $\delta : S \to S$ здійснює відображення стану процесора у новий стан $s' = \langle pc', R', M', E' \rangle$ шляхом виконання поточної інструкції у pc. Транзитивну функцію δ визначимо наступним чином:

$$\delta(pc,R,M,E) \overset{def}{=} \begin{cases} (pc,R,M,E) & if\ pc = halt \vee E \neq \perp \\ (pc,R,M,E') & if\ exception \\ (pc',R',M',\perp) & otherwise \end{cases}$$

Якщо лічильник інструкцій вказує на синтаксично та семантично вірну інструкцію і виконання цієї інструкції не призводить до жодних виключних ситуацій тоді $\delta(pc,R,M,E) = (pc',R',M',\perp)$. Стан регістрів R' та пам'яті M' оновлюються відповідно до семантики інструкції, що виконується, лічильник інструкцій pc' вказує на наступну інструкцію, і $E'=\perp$. З іншого боку, якщо відбувається виключна ситуація тоді $\delta(pc,R,M,E) = (pc,R,M,E')$ і $E' \neq \perp$. Виникнення виключної ситуації вказує на те, що інструкція не може бути виконання центральним процесором, і, відповідно, лічильник інструкцій, регістри процесора та пам'ять зберігають свій попередній стан. Коли остання інструкція буде виконана, в лічильнику інструкцій встановлюється значення *halt*, і з цієї точки стан середовища виконання не змінюється. Подібна ситуація виникає коли відбувається виключна ситуація.

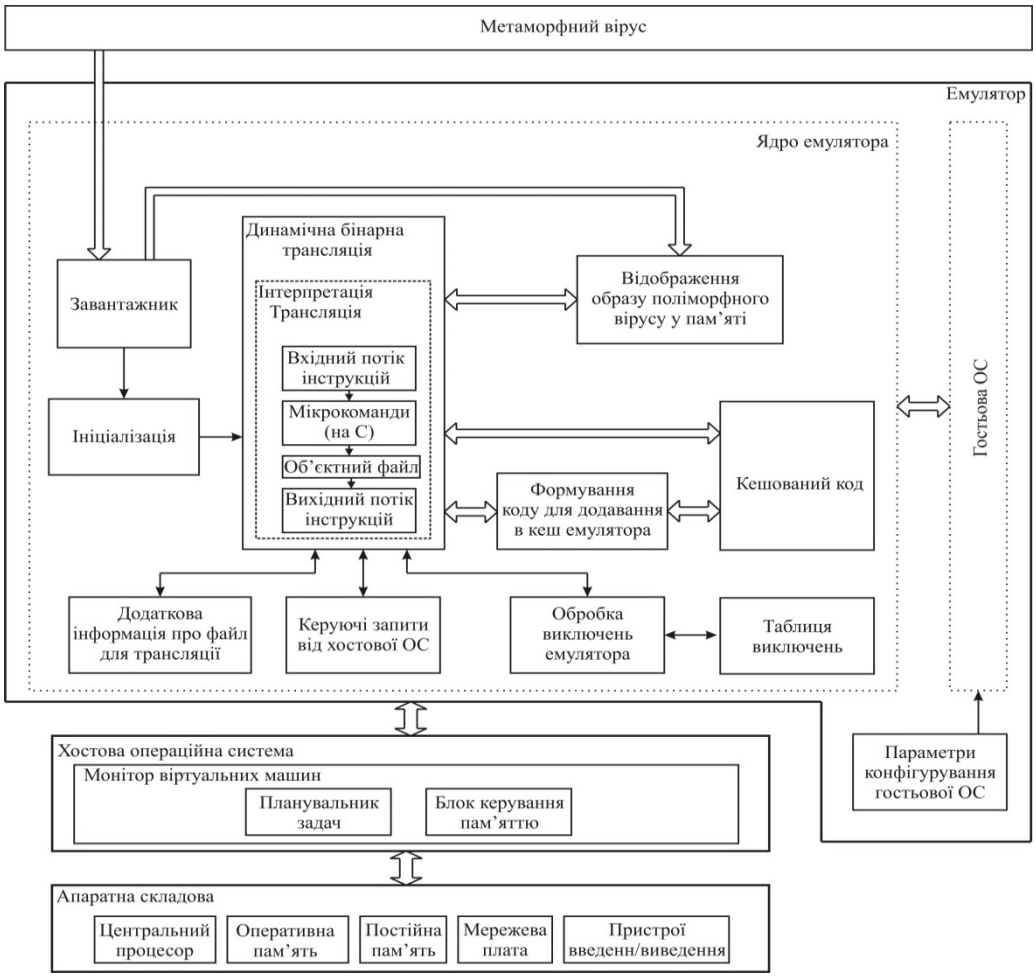


Рис. 3. Узагальнена структурна схема модифікованого емулятора для виявлення метаморфних вірусів

Розглянемо структуру та параметри модифікованих емуляторів, що використовуються для технології виявлення метаморфних вірусів.

На початковому етапі функціонування, завантажник (рис. 3) записує вхідний код метаморфного вірусу та дані з гостьової ОС в область пам'яті емулятора для подальшого його запуску та виконання. У якості формату виконуваних файлів для виявлення метаморфних вірусів у корпоративній мережі з використанням модифікованих емуляторів, визначено PE-EXE формат. Ядро емулятора сприймає вхідний код та дані у пам'яті як потік даних для процесу інтерпретації та/або трансляції, оскільки вхідний код не виконується на реальному апаратному забезпеченні хоста.

На наступному етапі завантажник здійснює виділення простору пам'яті для буфера під кеш код та інших таблиць даних, що використовуються в процесі емуляції. Процес ініціалізації також передбачає встановлення зв'язку з ОС для встановлення обробників сигналів умов переривання роботи, що можуть виникнути під час роботи емулятора.

Для здійснення попереднього аналізу вся множина вхідного коду розбивається на блоки для трансляції (translated blocks) до інструкцій переходу (jump) або інструкцій, що змінюють стан процесора. З метою підвищення швидкодії, блоки, що найчастіше використовуються заносяться у кеш пам'ять емулятора,

розміром 16 Мб, при переповненні якого здійснюється очищення.

У випадку появи сигналу про помилки чи переривання під час динамічної трансляції здійснюється обробка виключень. Виключна ситуація може виникнути при наявності в тілі поліморфного вірусу антиемуляційних методів. Наприклад, наявність ділянки коду вірусу, що здійснює інкремент/декремент змінної велику кількість разів (мільйон). У випадку відсутності генерації виключної ситуації при обробці такого блоку коду буде зависання процесу емуляції та, відповідно, відсутність інформації про підозрілу програму. Всі ці значення зберігаються у таблиця виключень.

Блок опрацювання керуючих запитів від ОС включає в себе поведінку емулятора при взаємодії з ОС та з зовнішніми пристроями (клавіатура, миша та ін.) шляхом перетворення викликів ОС у відповідні виклики емулятора. Це передбачає відстеження станів програмного лічильника, регістрів та умов переривання.

Для забезпечення змінного середовища виконання використовується блок здійснюється конфігурування параметрів гостьової ОС, що включають в себе:

- тип операційної системи, її розрядність;
- налаштування MAC адреси гостьової ОС;
- налаштування імені користувача, серійного номеру встановленої 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;
- з метою ухилення від перевірки часу виконання (rdtsc та GetTimeTick()) здійснення створення простого додатку з назвою процесу та вікна OllyDbg, що дозволить ввести в оману метаморфний вірус про існування відлагоджувальника OllyDbg та не застосовувати дану техніку ухилення [5];
- запуск виконуваного файлу здійснюється з використання додатку, що виконує симуляції руху та натискань миші;
- модифікація функції IsDebuggerPresent(). Значення що повертається завжди true;
- зміна конфігураційного файлу та встановлення правил, (табл. 1).

Таблиця 1

Набір правил для конфігураційного файлу модифікованого емулятора (на хостовій ОС)

№	Правило
1	isolation.tools.getPtrLocation.disable = «TRUE»
2	isolation.tools.setPtrLocation.disable = «TRUE»
3	isolation.tools.setVersion.disable = «TRUE»
4	isolation.tools.getVersion.disable = «TRUE»
5	monitor_control.disable_directexec = «TRUE»
6	monitor_control.disable_chksimd = «TRUE»
7	monitor_control.disable_ntreloc = «TRUE»
8	monitor_control.disable_selfmod = «TRUE»
9	monitor_control.disable_reloc = «TRUE»
10	monitor_control.disable_btinout = «TRUE»
11	monitor_control.disable_btmemspace = «TRUE»
12	monitor_control.disable_btpriv = «TRUE»
13	monitor_control.disable_btseg = «TRUE»

Окрім того, 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.

Експерименти

З метою оцінки ефективності методу виявлення метаморфних вірусів було досліджено залежність рівня виявлення метаморфних вірусів від кількості хостів, на яких розташовувались модифіковані емулятори. Для цього було залучено університетську мережу. Вона складається з 80 робочих станцій. На кожній станції було встановлено модифіковані емулятори на основі Qemu [8]. В якості дизасемблера в модифікованому емуляторі було використано IDA Pro [9].

На кожному хості було проведено налаштування параметрів модифікованих емуляторів, що наведені у табл. 2.

Для проведення цього експерименту було згенеровано по 3 різних копії метаморфних вірусів трьох

типів – NGVCK, VCL32 та G2 [10]. Всі метаморфні версії, що створювались за допомогою зазначених генераторів були скомпільовані з опціями anti-debugging та anti-emulation.

Спочатку здійснювалась встановлення одного метаморфного вірусу на 10 хостів (рис. 4). Кожен наступний експеримент передбачав збільшення кількості хостів на десять. Таким чином, для згенерованих 9 вірусів було проведено по 8 експериментів. На рис. 4 представлено усереднені результати виявлення проявів метаморфного вірусу в модифікованих емуляторах.

Таблиця 2

Параметри модифікованих емуляторів

Параметри, що змінюються на кожному модифікованому емуляторі				
№	Параметр	Початок діапазону	Кінець діапазону	Кількість можливих значень
1	Зміна MAC адреси	00-15-5D-01-80-00	00-15-5D-01-8F-FF	4096
2	Тип ОС	Лінійка ОС Windows NT		5
3	Розрядність ОС	32	64	2
4	Доступ до мережі Internet	Так	Ні	2
5	Налаштування системних портів для емуляторів в яких є доступ до мережі Internet*	Відкритий	Закритий	2 (для кожного порту з переліку)
6	Зміна параметру ключа реєстру HKEY_LOCAL_ MACHINE \SYSTEM\ControlSet001\Services\ Disk\Enum	WD3200BEAT	WD3200BEVT	24 (назви жорсткого диску)
7	Розмір встановленої оперативної пам’яті, Мб	1024	4096	4
8	Встановлення паролю адміністратора	111111	999999 (а також password, qwerty, abc)	48
9	Створення простого додатку з назвою процесу OllyDbg	Так	Ні	2
Початкові параметри для всіх модифікованих емуляторів				
10	Конфігурування файлу модифікованого емулятора (на хостовій ОС – табл. 1)			
11	Відключення модуля обміну даними між віртуальною машиною та хостовою ОС			
12	Модифікація API IsDebuggerPresent()**			
13	Заміна інструкції, що не може бути виконана на інструкцію NOP			
14	Запуск виконуваного файлу через додаток, що виконує симуляції руху та натискань миші			

*В емуляторах, що мали з’єднання з мережею Internet було здійснено конфігурування найбільш вразливих до атаки системних портів: 20, 21, 2,3 25, 53, 80, 110, 137, 138, 139, 143, 445, 8000, 8080, 3128, 3389, 6588, 1080, 5900, 8888.
** Модифікація API функції IsDebuggerPresent()** передбачала завжди встановлення поверненого значення у true.

Результати проведеного експерименту свідчать про те, що зі збільшення хостів в мережі, на яких встановлено модифіковані емулятори, збільшується рівень прояву метаморфних властивостей.

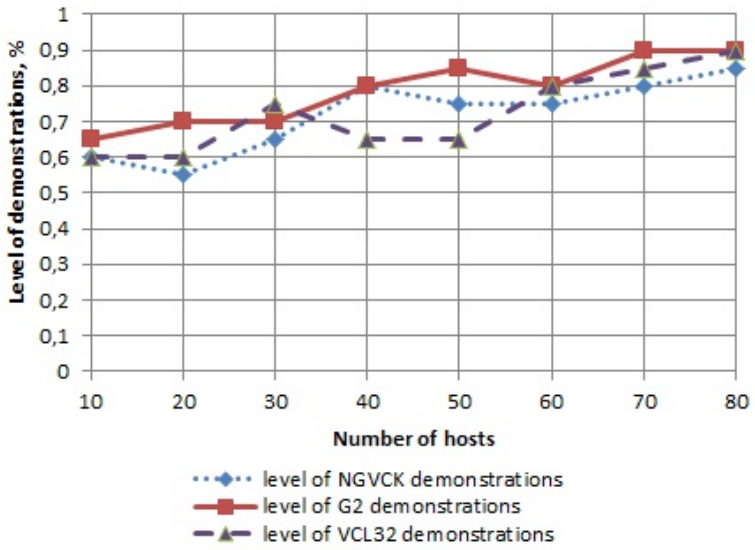


Рис. 4. Залежність рівня прояву метаморфних вірусів від кількості хостів в мережі

Висновок

В роботі проведено аналіз антиемуляційних технологій, що використовуються для виявлення штучного середовища метаморфними вірусами, розроблена їх класифікація. Ґрунтуючись на особливостях застосування антиемуляційних технологій розроблено її модель, що дозволяє здійснити визначення основних вразливостей віртуального середовища. На основі моделі застосування антиемуляційних технологій сформовано набір правил та налаштувань для модифікованих емуляторів, які використовуються для виявлення метаморфних вірусів в корпоративній мережі. Результати проведених експериментів свідчать про те, що зі збільшенням кількості модифікованих емуляторів рівень прояву метаморфних вірусів зростає.

Література

1. Kruegel, C.: Evasive Malware Exposed and Deconstructed, RSA Conference, pp. 1-70, (2015).

2. Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions. In: Proceedings of the ACM Conference on Computer and Communications Security, pp. 51-62, CCS (2008).

3. Pek, G., Bencsáth, B., Buttyán, L.: nEther: In-guest Detection of Out-of-the-guest Malware Analyzers. In: ACM European Workshop on System Security, pp. 85-91, EUROSEC (2011).

4. Lindorfer, M., Kolbitsch, C., Comparetti, P.M.: In Proc. of the 14th International Symposium, vol. 6964, pp. 338-357, RAID, (2011).

5. Chen, X., Andersen, J., Mao, M., Bailey, M., Nazario, J.: Towards an Understanding of Anti-virtualization and Anti-debugging Behaviorin Modern Malware, In Proc. of the 5th IEEE International Conference on Malicious and Unwanted Software, pp. 177-186, MALWARE, (2008).

6. Raffetseder, T., Kruege, C., Kirda, E.: Detecting System Emulators: In Proc. of the 10th international conference on Information Security, pp. 1-18, (2007).

7. Pomorova, O., Savenko, O., Lysenko, S., Nicheporuk, A.: Metamorphic virus detection technique based on the modified emulators: In proc. of the 12th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer, pp. 375-383, ICTERI (2016).

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

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

10. VX Heavens [online] Available: http://vxheaven.org/

Рецензія/Peer review : 5.3.2017 р. Надрукована/Printed :17.4.2017 р.
Рецензент: д.т.н. Мартинюк В.В.