УДК 621.391
DOI: 10.31673/2409-7292.2023.010007

Агачі О. О.

ТЕХНОЛОГІЯ ВИЯВЛЕННЯ ПОЛІМОРФНИХ КОМП'ЮТЕРНИХ ВІРУСІВ

Традиційні віруси являли собою комп'ютерні програми зі статичною структурою, які виявляли дуже обмежену функціональність. Після першого виявлення їх структура використовується антивірусною програмою (АВП) як інструмент для виявлення подібних вірусів зі схожими моделями. Однак сучасні віруси можуть самостійно налаштовуватися та навіть змінювати структуру своєї функціональності, що ускладнює їх виявлення антивірусним програмним забезпеченням. Поліморфний вірус – це складний комп'ютерний вірус, який впливає на типи даних і функції, що ускладнює перевірку його внутрішньої структури. У статті розглянуто загальні методи, які використовуються цими вірусами для демонстрації поліморфізму; сучасний рівень виявлення поліморфних вірусів; антивірусне програмне забезпечення для виявлення таких вірусів. Результати цього дослідження можуть стати джерелом знань для дослідників і компаній, що займаються антивірусним програмним забезпеченням.

Ключові слова: поліморфний вірус, шкідливе ПЗ, антивірус.

Вступ

Один з перших вірусів відомий як Elk Cloner [1] – програма, написана 15-річним Річардом Скрентою, учнем середньої школи у 1982 році, яка демонструвала на екрані маленький вірш. Вона не пошкоджувала жодних ресурсів на комп'ютері, але дратувала людей набридливим повідомленням. Програма могла поширюватися й заражати інші операційні системи. Пізніше Джон фон Нейман [2] створив перші в історії комп'ютерні програми, що самовідтворюються. Вірус міг проникати на комп'ютер через електронну пошту, вкладення текстових повідомлень, соціальні посилання, безкоштовні програми, веселі зображення, аудіо- та відеофайли. Щоб уникнути виявлення, автор вірусу використовував різні методи приховування коду, найбільш простий з яких – не змінювати дату «останньої зміни» файлу хосту під час його зараження.

Інший вірус, наприклад Chernobyl [3], використовував невикористані області виконуваних файлів, перезаписуючи їх шкідливим кодом, що дозволяє зберегти той самий розмір заражених файлів. Більш просунута техніка, Conficker [4], припиняє виконання завдань, пов'язаних з антивірусним програмним забезпеченням, перш ніж воно було виявлено. Оскільки операційні системи постійно оновлюються, що не дозволяє змінювати файли або зупиняти процес без відповідного дозволу, авторам вірусу довелося використовувати інші технології приховування своїх програмних кодів.

Перша техніка була відома як самомодифікація [5], ця техніка була розроблена для протидії антивірусним програмам (АВП), які сканують вірус за сигнатурою. По суті, АВП підтримуватиме базу даних, яка містить список сигнатур для кожного виявленого вірусу. Коли вона сканує файл, вона порівнює сигнатур файлу з базою даних сигнатур – коли рядок збігається, цей файл вважається зараженим, тоді цей файл можна видалити, заблокувати або очистити (видалити сигнатуру). Щоб уникнути виявлення, вірус модифікує себе за допомогою нової сигнатури для кожного інфікованого файлу. У такий спосіб автори вірусів можуть створювати нескінченну кількість сигнатур.

Ще один спосіб уникнути виявлення сигнатури – це шифрування. Цей підхід використовує простий метод для шифрування тіла шкідливого коду. Кожен ключ шифрування створює зашифрований текст, тож вірус може розмножуватися у багатьох різних файлах, лише змінюючи ключ шифрування. Кожен заражений файл міститиме зашифрований шкідливий код, модуль дешифрування та ключ шифрування. Унікальний зашифрований шкідливий код призведе до іншої сигнатури, що ускладнить виявлення АВП. Основним недоліком цього методу є модуль дешифрування, який залишається постійним у всіх заражених файлах, відкриваючи можливий шлях для виявлення програмним забезпеченням АВП.

Щоб подолати обмеження методу шифрування з постійним модулем дешифрування, було розроблено нову техніку, яка передбачає динамічний модуль дешифрування, який буде

змінюватися при кожному зараженні. Такий метод називається поліморфним кодом [6]. Цей поліморфний вірус став одним із найскладніших завдань для виявлення антивірусними програмами, оскільки він є вірусом із самошифруванням і здатний дублюватися, створюючи дещо модифіковані версії самого себе.

Ще більш досконалою технікою є метаморфічний код [7], у якому вірус повністю переписує себе під час кожного виконання. Однак цей метод надзвичайно дорогий, оскільки вимагає метаморфічного двигуна, що робить його непрактичним на практиці.

Огляд літератури

Як правило, для розуміння шаблону та поведінки шкідливої програми в аналізі використовуються два загальні підходи: (1) статичний аналіз і (2) динамічний аналіз. Статичний аналіз передбачає аналіз бінарних сигнатур зловмисного програмного забезпечення без його виконання; тоді як динамічний аналіз спостерігає за поведінкою запущеного шкідливого коду в контрольованому середовищі.

Серед методів статичного аналізу виділяють: підхід на основі сигнатур [8]; аналіз системних викликів [9]; граф потоку керування [3, 10]; перевірка моделі [11] аналіз потоку даних [12]; аналіз машинного навчання [13].

З методів динамічного аналізу можна виділити: архітектуру для виявлення поліморфного вірусу, яка включає три компоненти: (1) емулятор, (2) оперативний аналізатор коду, і (3) евристичний аналізатор [14]. Також, у [15] представлено евристичний метод виявлення, який сканує потоки мережевого трафіку на наявність поліморфного шелл-коду. У [16] запропоновано пристрій для виявлення шкідливого коду, який використовує виклики операційної системи для пошкодження комп'ютерних систем. У [17] розглядаються додаткові програмні інструкції в емуляторі для виявлення підозрілого коду. У [18] представлено апарат для виявлення шкідливого програмного забезпечення шляхом аналізу шаблонів системних викликів, створених під час емуляції. У [19] запропоновано метод виявлення зловмисного програмного забезпечення шляхом динамічного розбирання шкідливого коду, а потім компіляції цього коду для націлювання на центральний процесор.

Метою цього дослідження є розгляд способів виявлення поліморфного вірусу.

При цьому варто розглянути такі питання: 1. Які загальні методи використовуються вірусами для реалізації поліморфізму? 2. Які існують технології виявлення поліморфних вірусів? 3. Що потрібно зробити, щоб АВП виявила ці віруси?

Поліморфний вірус

Перший поліморфний вірус був написаний Марком Уошберном у 1990 році [20], він був відомий як вірус 1260 або V2PX через його довжину (1260 байт). Уошберн хотів показати антивірусній спільноті, чому сканери рядків ідентифікації працюють не у всіх випадках. Довжина заражених файлів буде збільшена на 1260 байт і зашифрована. Ключ шифрування змінюється з кожним зараженням. V2PX не був резидентним у пам'яті, він заражав файли *.COM у поточному або PATH-каталогах під час виконання. Для розшифровки тіла вірусу було використано два ковзних ключа, але, що більш важливо, у дешифратор було додано зайві інструкції. Вони працювали як камуфляж для коду. Залежно від кількості вставленого зайвого коду дешифратор може бути коротшим або довшим. Крім того, кожна група інструкцій у дешифраторі може бути переставлена в будь-якому порядку. На рис. 1 показаний приклад дешифратора, де видно що в кожну групу інструкцій додано набір непотрібних інструкцій (INC SI, CLC, NOP та інші інструкції, які нічого не роблять).

Наступним кроком розвитку поліморфного вірусу стала поява Mutation Engine (MtE) [21], цей механізм був написаний болгарським Dark Avenger. Ідея мутаційного механізму базувалася на модульній розробці. Концепція MtE полягала в тому, щоб зробити виклик функції до функції MtE та передачу параметрів керування в попередньо визначені регістри. MtE створить поліморфну оболонку навколо простого вірусу всередині нього. Коли вірус використовує механізм для запису у файл, шифрувальник MtE змінює код вірусу, щоб він виглядав як випадкове сміття. Дешифратор виключить цей код після його виконання.

```
inc     si      ; optional, variable junk
mov     ax,0E9B ; set key 1
clc     ; optional, variable junk
mov     di,012A ; offset of Start
nop     ; optional, variable junk
mov     cx,0571 ; this many bytes - key 2

; Group 2 - Decryption Instructions
Decrypt:
xor     [di],cx ; decrypt first word with key 2
sub     bx,dx   ; optional, variable junk
xor     bx,cx   ; optional, variable junk
sub     bx,ax   ; optional, variable junk
sub     bx,cx   ; optional, variable junk
nop     ; non-optional junk
xor     dx,cx   ; optional, variable junk
xor     [di],ax ; decrypt first word with key 1
; Group 3 - Decryption Instructions
inc     di      ; next byte
nop     ; non-optional junk
clc     ; optional, variable junk
inc     ax      ; slide key 1
; loop
loop    Decrypt ; until all bytes are decrypted - slide key 2
; random padding up to 39 bytes
```

Рис. 1. Приклад дешифратора для вірусу 1260

Дешифратор – це та частина вірусу, яка залишається незашифрованою. Під час запуску зараженого файлу дешифратор спочатку отримує контроль над системою, а потім розшифровує як тіло вірусу, так і MtE. Потім він передає контроль над системою вірусу, який, у свою чергу, знайде новий файл для зараження. Параметри механізму MtE включають [22]: 1) робочий сегмент; 2) покажчик на код для шифрування; 3) довжину тіла вірусу; 4) базу дешифратора; 5) адресу точки входу хоста; 6) цільове розташування зашифрованого коду; 7) розмір дешифратора (малий, середній або великий); 8) бітове поле регістрів.

Дешифратор, створений MtE, як показано на рис. 2, повертає дешифратор із зашифрованим тілом вірусу в наданому буфері. З цього моменту MtE і сам вірус копіюються в оперативну пам’ять (RAM).

```
        ; to "Start"-delta
mov     cl,03   ; (delta is 0x0D2B in this example)
ror     bp,cl
mov     cx,bp
mov     bp,856E
or      bp,740F
mov     si,bp
mov     bp,3B92
add     bp,si
xor     bp,cx
sub     bp,B10C ; Huh ... finally BP is set, but remains an
        ; obfuscated pointer to encrypted body

Decrypt:
mov     bx,[bp+0D2B]    ; pick next word
        ; (first time at "Start")
add     bx,9D64 ; decrypt it
xchg    [bp+0D2B],bx    ; put decrypted value to place

mov     bx,8F31 ; this block increments BP by 2
sub     bx,bp
mov     bp,8F33
sub     bp,bx   ; and controls the length of decryption

jnz     Decrypt ; are all bytes decrypted?

Start:
```

Рис. 2. Приклад дешифратора, згенерованого MtE

© Агачі, О. О. (2023). Технологія виявлення поліморфних комп’ютерних вірусів. Сучасний захист інформації, 1(53), 52–58. https://doi.org/10.31673/2409-7292.2023.010007.

Спочатку було запущено механізм мутації, а потім він випадковим чином згенерував новий дешифратор, здатний розшифрувати вірус. Далі MtE та вірус шифруються. Нарешті вірус додав цей новий дешифратор разом із нещодавно зашифрованим вірусом і MtE до нової цілі. У результаті цієї стадії дешифратор змінювався в кожній інфекції, що ускладнювало антивірусному сканеру пошук сигнальної послідовності байтів, яка ідентифікує конкретний дешифратор, оскільки немає фіксованої сигнатури дешифратора чи двох подібних інфекцій.

Загалом поліморфні віруси можна класифікувати за різними рівнями складності:

Рівень 1. Для створення поліморфного вірусу з певного набору обирається схема шифрування/дешифрування. Екземпляр вірусу матиме одну з цих схем у вигляді звичайного тексту, як показано на рис. 3. Відкритий ключ для цього шифрування можна надати багатьом користувачам для шифрування повідомлення. Такий вірус вважається «напівполіморфним».

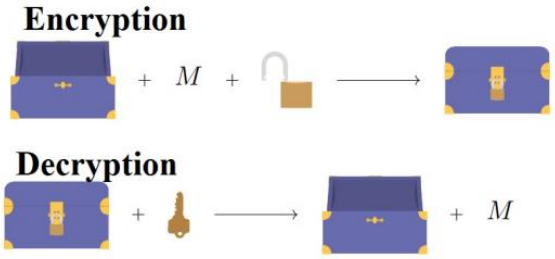

Рис. 3. Приклад напівполіморфного вірусу

Рівень 2. Програма розшифровки вірусу містить одну або кілька постійних інструкцій, решта можна змінювати, як показано на рис. 4, алгоритм використовує змінні A і B, але не змінну C, що дозволяє нескінченно змінювати C.

Рівень 3. Дешифрувальник вірусів містить невикористовувані функції або інструкції, такі як NOP, CLI та STI тощо, як показано на рис. 1.

```
    lots of encrypted code
    ...
Decryption_Code:
    C = C + 1
    A = Encrypted
Loop:
    B = *A
    C = 3214 * A
    B = B XOR CryptoKey
    *A = B
    C = 1
    C = A + B
    A = A + 1
    GOTO Loop IF NOT A = Decryption_Code
    C = C^2
    GOTO Encrypted
 CryptoKey:
    some_random_number
```

Рис. 4. Простий поліморфний вірус

Рівень 4. Дешифратор вірусів використовує взаємозамінні інструкції та змінює їх порядок (змішування інструкцій), як показано на рис. 5.

Рівень 5. На цьому рівні поліморфний вірус використовує усі перераховані вище методи. Крім того, алгоритм дешифрування може бути змінений.

Рівень 6. Перманентні віруси. Це найвищий рівень поліморфного вірусу, і його слід називати поліморфним вірусом тіла або метаморфічним вірусом. На цьому етапі весь основний код вірусу може бути змінений поколіннями:

$$G_0 \to G_1 \to G_2 \to ... \to G_n .$$

```
Decrypt:
 add  $4, %bh          ; junk
 xor  %edx, %edx       ; junk
 xor  %al, (%esi)      ; decrypt a byte with key in AL
 inc  %al              ; slide the key up
 xchg %edx, %ebx       ; junk
 inc  %esi             ; go to next byte
 cmp  %ecx, %edx       ; junk
 dec  %ecx             ; decrement the byte counter
 jnz  Decrypt          ; loop back if more to decrypt
```

Рис. 5. Порядково-незалежні інструкції

Інфекційний процес при поліморфному вірусі

Отже, поліморфний вірус складається з: 1) процедури дешифрування; 2) зашифрованого тіла вірусу; 3) механізму мутації, який генерує рандомізовані процедури дешифрування.

У поліморфних вірусах механізм мутації та тіло вірусу зашифровані. Коли заражена програма виконується, програма дешифрування вірусу отримує контроль над комп'ютером, дешифрує тіло вірусу та механізм мутації. Потім управління передається вірусу, який знаходить нову програму для зараження. Оскільки тіло вірусу зашифровано, а процедура розшифровки також різна від зараження до зараження, сканери вірусів не можуть сканувати фіксовану сигнатуру або фіксовану процедуру розшифровки, що ускладнює виявлення.

Щоразу, коли АВП виявляє вірус, він потрапляє в чорний список, а будь-який інший вірус із подібними характеристиками автоматично блокується. У випадку поліморфного вірусу з кожною мутацією основна функція, яку виконує вірус, залишається незмінною, навіть якщо змінюється процедура сигнатури або дешифрування. АВП, яка використовує традиційне виявлення на основі сигнатур, не може знайти та заблокувати зловмисний код після того, як зміниться процедура сигнатури та дешифрування. Таким чином, вірус робить копію самого себе та механізму мутації. Потім запускається механізм мутації, і генерується нова процедура дешифрування, яка не схожа на попередню процедуру дешифрування. Далі вірус шифрує своє тіло та механізм мутації та додає нову процедуру дешифрування, зашифрований вірус і механізм мутації до нової програми.

Виявлення поліморфного вірусу

Поліморфні віруси можуть легко обдурити звичайне антивірусне програмне забезпечення за допомогою виявлення на основі сигнатур. Однак ці віруси можна виявити за допомогою нових технологій безпеки, які використовують машинне навчання та виявлення поведінки для виявлення будь-якої аномальної поведінки в системі.

1. *Виявлення на основі поведінки*: ця техніка аналізує не лише код, але й поведінку вірусу. Це допомагає виявити віруси з подібною поведінкою.

2. *Евристичне сканування*: ця техніка шукає компоненти, які мають різні загрози, замість пошуку точної відповідності загрозі. Це допомагає виявити нові різновиди вірусів.

Як приклади поліморфних вірусів можна навести: URSNIF; CryptoWall; VIRLOCK; VOBFUS; Beebone; Storm Worm та ін.

Зміни в програмному забезпеченні АВП

Через зміни поліморфного вірусу АВП мають різні стратегії боротьби з ним. Найзручнішим є аналіз вірусів один за одним, рядок за рядком, але цей метод займає багато часу, дорогий і непрактичний, а також призводить до помилкової ідентифікації одного поліморфу за інший.

Тому, на практиці використовується загальний метод [22], який передбачає, що: 1) тіло поліморфного вірусу шифрується; 2) перед тим, як вірус зможе запуститися має бути виконано

© Агачі, О. О. (2023). Технологія виявлення поліморфних комп'ютерних вірусів. Сучасний захист інформації, 1(53), 52–58. https://doi.org/10.31673/2409-7292.2023.010007.

процес розшифровки; 3) після виконання зараженого файлу поліморфний вірус повинен взяти під контроль систему, щоб розшифрувати тіло вірусу, а потім передати контроль над комп'ютером розшифрованому вірусу.

На основі такої поведінки АВП завантажує заражений файл у самостійний віртуальний комп'ютер, створений з оперативної пам'яті. Інфікований файл запускається так, ніби він працює на реальному комп'ютері. Виконання контролюється сканером, щоб вірус не міг завдати шкоди реальному комп'ютеру. Коли вірус запускається, він віддає своє тіло сканеру, який, у свою чергу, може шукати сигнатури в тілі вірусу, які точно ідентифікують штам вірусу. Якщо вірусу немає, АВП швидко припиняє запуск файлу, видаляє його з оперативної пам'яті та переходить до сканування наступного файлу.

Швидкість є ключовою проблемою загального підходу до дешифрування. Буде непрактично, якщо поліморф розшифровується та виконується в оперативній пам'яті протягом кількох годин. З іншого боку, якщо процес виявлення незабаром зупиниться, АВП може пропустити основний зловмисний код, перш ніж він зможе розкрити його достатньо, щоб сканер виявив сигнатури.

Щоб подолати цей недолік, використовується евристичний метод. Цей метод містить набір правил, які допомагають відрізнити невірусну поведінку від вірусної. Цей метод працює на основі припущення, що під час звичайної роботи комп'ютер виконує деякі математичні обчислення та використовує ці результати. З іншого боку, поліморфний вірус може виконувати подібні обчислення, але викидати (не використовувати) результати. Загальне дешифрування на основі евристики шукає таку суперечливу поведінку, щоб вирішити, чи продовжити тривалість виконання підозрілого файлу всередині віртуального комп'ютера, даючи потенційно зараженому файлу достатньо часу, щоб розшифрувати себе та викрити прихований вірус. Однак цей метод дає високий хибний негатив, коли він змінює свою базу правил для виявлення нових вірусів. Коли автори вірусів намагаються зробити вірус схожим на чисту програму, сканер збільшує час, необхідний для перевірки підозрілого файлу. Отже, цей підхід швидко стає неточним і неефективним на практиці.

Система Striker

Ця система надається компанією Symantec Cooperation Anti-virus. Перший крок подібний до попереднього, тобто завантажує заражений у віртуальний комп'ютер із оперативної пам'яті. Однак він покладається не на евристичні припущення, а на профілі вірусу або правила, визначені для кожного вірусу, щоб не розрізняти невірусну поведінку та поведінку вірусу.

Під час перевірки нового файлу система спочатку намагається виключити з розгляду якомога більше вірусів. Наприклад, деякі віруси можуть інфікувати лише файли .COM, .EXE або .SYS. При перевірці зараженого файлу з розширенням .EXE Striker розробляє поліморфний вірус, який заражає лише файли .COM або .SYS. Якщо всі віруси виключено з розгляду, то файл вважається чистим, система закриється та просканує наступний файл. Після попереднього кроку, якщо зараження не виявлено, Striker продовжує працювати з файлом у віртуальному комп'ютері, доки цей файл має зіставлення поведінки принаймні з одним відомим поліморфним вірусом або MtE.

Перевагою підходу Striker є швидкість, оскільки профілі вірусів не тільки дозволяють системі швидко виключати деякі поліморфні віруси, але й швидко обробляти неінфіковані файли, таким чином мінімізуючи робоче навантаження на систему. Наразі загальне розшифровування вважалося єдиним найефективнішим методом виявлення поліморфного вірусу, і система Striker вдосконалює цей підхід.

Оскільки нові антивірусні системи були розроблені, автори вірусів також отримали нові способи написання коду, завдяки чому ця битва ніколи не закінчується.

Останніми роками штучний інтелект став новим трендом у багатьох сферах, зокрема у виявленні шкідливих програм. Цей багатообіцяючий метод було доведено в дослідженні [23]. Запропонована ними модель має середній показник точності виявлення 80 %.

Висновки

На теперішній час поліморфні віруси є складним явищем, яке важко піддається діагностуванню. Поліморфні віруси покладаються на механізми мутації, щоб змінювати свої процедури дешифрування кожного разу, коли вони заражають машину. Таким чином, традиційні рішення безпеки можуть нелегко вловити їх, оскільки вони не використовують статичний, незмінний код. Використання складних механізмів мутації, які генерують мільярди процедур дешифрування, ще більше ускладнює їх виявлення. Однак розширені антивірусні програми, які використовують евристичний аналіз, можуть виявляти нові загрози, такі як поліморфні віруси. При цьому ретельно вивчається структура потенційної загрози, логіка програмування та зміст небажаного коду, незвичних інструкцій і поведінка загрози.

Перелік посилань:

1. Spafford, E. H., Heaphy, K. A., and Ferbrache, D. J. 1989. A computer virus primer.
2. Von Neumann, J., and Burks, A. W. 1996. Theory of selfreproducing automata. University of Illinois Press Urbana.
3. Christodorescu, M., and Jha, S. 2006. Static analysis of executables to detect malicious patterns. Tech. rep., Wisconsin Univ-Madison Dept of Computer Sciences.
4. Porras, P., Saidi, H., and Yegneswaran, V. 2009. Conficker c analysis. SRI International.
5. Anckaert, B., Madou, M., and De Bosschere, K. 2006. A model for self-modifying code. In International Workshop on Information Hiding, Springer, 232–248.
6. Torrubia-Saez, A., 2003. Polymorphic code generation method and system therefor, July 8. US Patent 6,591,415.
7. Borello, J.-M., and Me´, L. 2008. Code obfuscation techniques for metamorphic viruses. Journal in Computer Virology 4, 3, 211–220.
8. Griffin, K., Schneider, S., Hu, X., and Chiueh, T.-C. 2009. Automatic generation of string signatures for malware detection. In International workshop on recent advances in intrusion detection, Springer, 101–120.
9. Sung, A. H., Xu, J., Chavez, P., and Mukkamala, S. 2004. Static analyzer of vicious executables (save). In Computer Security Applications Conference, 2004. 20th Annual, IEEE, 326– 334.
10. Bonfante, G., Kaczmarek, M., and Marion, J.-Y. 2007. Control flow graphs as malware signatures. In International workshop on the Theory of Computer Viruses.
11. Chaumette, S., Ly, O., and Tabary, R. 2011. Automated extraction of polymorphic virus signatures using abstract interpretation. In Network and System Security (NSS), 2011 5th International Conference on, IEEE, 41–48.
12. Agrawal, H., Bahler, L., Micallef, J., Snyder, S., and Virodov, A. 2012. Detection of global, metamorphic malware variants using control and data flow analysis. In Military Communications Conference, 2012-MILCOM 2012, IEEE, 1–6.
13. Moskovitch, R., Elovici, Y., and Rokach, L. 2008. Detection of unknown computer worms based on behavioral classification of the host. Computational Statistics & Data Analysis 52, 9, 4544–4566.
14. Yann, T., and Petrovsky, O., 2006. Detection of polymorphic virus code using dataflow analysis, June 27. US Patent 7,069,583.
15. Polychronakis, M., Anagnostakis, K. G., and Markatos, E. P. 2006. Network–level polymorphic shellcode detection using emulation. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 54–73.
16. Rogers, A. J., Yann, T., and Jordan, M., 2012. Detection of viral code using emulation of operating system functions, Dec. 25. US Patent 8,341,743.
17. Muttik, I., and Long, D. V., 2005. Detecting computer viruses or malicious software by patching instructions into an emulator, June 14. US Patent 6,907,396.
18. Muttik, I., 2004. Detecting malicious software by analyzing patterns of system calls generated during emulation, Aug. 10. US Patent 6,775,780.
19. Stepan, A. E. 2005. Defeating polymorphism: beyond emulation. In Proceedings of the Virus Bulletin International Conference.
20. Szor, P. 2005. The art of computer virus research and defense. Pearson Education.
21. Bontchev, V. 1992. Mte detection test. Virus News Int, 26–34.
22. Nguyen, Vinh. (2018). A study of polymorphic virus detection. 10.13140/RG.2.2.19853.79842.
23. Asiru, O., Dlamini, M., and Blackledge, J. 2017. Application of artificial intelligence for detecting derived viruses. In European Conference on Cyber Warfare and Security, Academic Conferences International Limited, 647–655.

Надійшла: 30.12.2022 Рецензент: д.т.н., професор Ахрамович В.М.