УДК 004.492.2

САМОКОНТРОЛЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЯК МЕТОД ПІДВИЩЕННЯ СТУПЕНЯ ЙОГО НАДІЙНОСТІ В ПРОЦЕСІ ЕКСПЛУАТАЦІЇ

А. В. Котовський, Р. С. Цвєнтарний, Ю. В. Зілінський
Кременчуцький національний університет імені Михайла Остроградського
вул. Першотравнева, 20, 39600, м. Кременчук, Україна. E-mail: ktp@kdu.edu.ua

На основі аналізу шляхів впровадження руткітів режиму користувача операційної системи Windows NT запропоновано методи реалізації самоконтролю програм для підвищення ступеня їх надійності при експлуатації в умовах активності шкідливого програмного забезпечення, що використовує модифікацію ланцюга системних викликів

Ключові слова: руткіт, перехоплення функцій, режим користувача, Win API.

SOFTWARE SELF-MONITORING AS A METHOD TO INCREASE THE DEGREE OF ITS RELIABILITY IN THE MAINTENANCE

A. V. Kotovskiy, R. S. Tsventarniy, U. V. Zilinskiy
Mykhaylo Ostrohradskiy National University of Kremenchuk
vul. Pershotravneva, 20, 39600, Kremenchug, Ukraine. E-mail: ktp@kdu.edu.ua

Based on analysis of ways of implementing user-mode root-kits Windows NT operating system offered techniques for implementing self-management programs to improve their reliability in operation in the activity of malicious software, which uses a modification of the system calls chain.

Key words: root-kit, intercept function calls, user mode, Win API.

САМОКОНТРОЛЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ КАК МЕТОД ПОВЫШЕНИЯ СТЕПЕНИ ЕГО НАДЕЖНОСТИ В ПРОЦЕССЕ ЭКПЛУАТАЦИИ

А. В. Котовский, Р. С. Цвентарный, Ю. В. Зилинский.
Кременчугский национальный университет имени Михаила Остроградского
ул. Первомайская, 20, 39600, г. Кременчуг, Украина. E-mail: ktp@kdu.edu.ua

На основе анализа путей внедрения руткитов режима пользователя операционной системы Windows NT предложены методы реализации самоконтроля программ для повышения степени их надежности при эксплуатации в условиях активности вредоносного программного обеспечения, которое использует модификацию цепочки системных вызовов.

Ключевые слова: руткит, перехват функций, режим пользователя, Win API.

АКТУАЛЬНІСТЬ РОБОТИ. В широкому сенсі під надійністю програмного забезпечення розуміють його здатність функціонувати без прояву будь-яких негативних наслідків для конкретної комп'ютерної системи та її користувача.

Одним із таких негативних наслідків є порушення цілісності та конфіденційності інформації в програмних комплексах комп'ютерних систем або надання програмним забезпеченням функціонально не придатних результатів обробки інформації. Серед причин цього, окрім різноманітних збоїв апаратури комп'ютерних систем, особливо слід відзначити збої, пов'язані з діями людей, які можна розділити на дві категорії: навмисні і ненавмисні дії.

До першої категорії належать дефекти програмного забезпечення внаслідок помилок програмування на етапі його розробки та помилкові дії користувачів (операторів) в процесі його експлуатації. До другої категорії належать дефекти, привнесені до програмного забезпечення виключно внаслідок зловмисних дій: різноманітні цільові програмні закладки, вбудовані на етапі його розробки або в процесі експлуатації.

Програмні закладки, вбудовані в процесі експлуатації, як правило, є наслідком дії шкідливого програмного забезпечення і є найбільш масовими. Переважна більшість шкідливого програмного забезпечення розробляється сьогодні для операційних систем (ОС) Windows NT. При цьому для реалізації більшості функціональних можливостей шкідливого програмного забезпечення його розробники повсюдно використовують руткіт-технології, що базуються на перехопленні функцій системного програмного інтерфейсу та модифікації системних структур даних.

Програма, яка піддалася атаці шкідливого програмного забезпечення, повинна, принаймні, інформувати про це користувача. В ідеальному випадку вона також повинна протидіяти атаці, вміти поновлюватися і стійко продовжувати своє функціонування.

Сьогодні не викликає сумнівів необхідність внесення до програмного забезпечення захисних функцій протягом всього його життєвого циклу. При цьому особливою складністю поряд із пошуком, локалізацією й усуненням програмних помилок є виявлення дефектів, навмисно внесених на етапі експлуатації програмних комплексів.

Робота [1] охоплює питання безпечного кодування в широкому спектрі програмного забезпечення, що забезпечує його стійкість до експлойтів на етапі експлуатації, однак не може протидіяти шкідливому програмному забезпеченню (malware), що використовує руткіт-технології.

У роботі [2] наведено опис більшості відомих на сьогодні руткіт-технологій, які використовує шкідливе програмне забезпечення.

У роботі [3] розглянуті загальні принципи побудови двомодульних обчислювальних процедур, методології самотестування і оцінки стійкості програм. Також у цій роботі зібрана значна бібліографія з питань розробки таких програм, які можуть самостійно перевірятися (self-testing) і самостійно відновлюватися (self-healing). Але теоретичні результати роботи [3] у контексті задач захисту потребують адаптації і можуть бути використані лише ідейно.

Таким чином, метою роботи є розробка і програмна реалізація методів виявлення руткітів в середовищі ОС Windows NT як засобу самоконтролю програмного забезпечення для підвищення ступеня його надійності в процесі експлуатації.

МАТЕРІАЛ І РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ. Методи виявлення руткітів режиму користувача операційної системи Windows NT розроблені на основі аналізу шляхів упровадження в систему, які вони використовують. Спираючись на розглянуті в [2] руткіти режиму користувача, встановлено, що вони базуються лише на двох методах упровадження: модифікація адрес точок входу функції в IAT та безпосередня модифікація коду самої функції (сплайсінг). Це значно обмежило коло пошуку порівняно з режимом ядра, який не є предметом досліджень цієї роботи.

Програмний інтерфейс додаткам (API) ОС Windows NT надає через клієнтські DLL підсистем оточення, використовуючи для цього два механізми – раннє і пізнє зв'язування. Схема виклику API-функцій при статичній компоновці DLL-бібліотеки (раннє зв'язування) зображена на рис. 1.

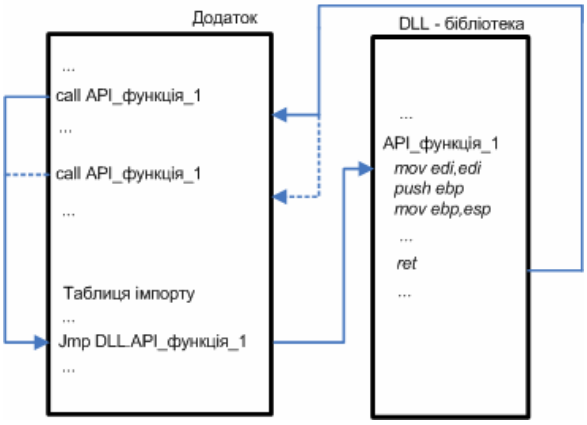

Рисунок 1 – Схема виклику API-функції

Перехоплення функцій шляхом модифікації їх точок входу в IAT базується на заміні адреси оригінальної функції на адресу функції руткіта. У такому випадку при виклику функції керування без відома додатку передається руткіту, якому надається можливість повного контролю результату виконання оригінальної функції до повернення його додатку. Схема такого перехоплення зображена на рис. 2.

З механізму перехоплення API-функцій шляхом модифікації точок входу функцій в IAT розроблено метод виявлення такого перехоплення шляхом перевірки належності адреси функції із таблиці імпорту до адресного простору бібліотеки, з якої вона експортується.

Рисунок 2 – Схема перехоплення API-функції шляхом модифікації IAT додатку

Оскільки в IAT, окрім адреси функції, також знаходиться ім'я DLL-бібліотеки, з якої вона експортується, то це дало можливість отримати дані про початкову адресу модуля та його розмір.

Алгоритм методу виявлення перехоплення API-функцій шляхом модифікації точок входу функцій в IAT полягає в наступному:

– визначити початкову (базову) адресу модуля, що експортує функцію;

– визначити розмір модуля;

– визначити приналежність адреси функції з таблиці імпорту до адресного простору модуля. Ця адреса має бути більша за базову адресу модуля і менша за алгебраїчну суму базової адреси модуля і його розміру.

Базова адреса модуля та його розмір зазвичай визначаються за допомогою API-функції GetModuleInformation. Однак виклик цієї функції також може бути перехоплений руткітом. Тому запропоновано визначати необхідну інформацію без використання виклику вказаної API-функції. Ця інформація отримується з відповідної структури LDR_MODULE, яка містить дані про модуль. Адреса структури LDR_MODULE визначається з даних структури PEB.

Структури LDR_MODULE, які містять інформацію про завантажені модулі, зберігаються у пам'яті у вигляді двозв'язних списків. Ці структури, як і сама структура PEB, є недокументованими. В узагальненому вигляді структури, які необхідні для пошуку інформації про модуль, описуються таким чином:

```
PEB struct
  …
  LoaderData              dd ?
  …
PEB ends
PEB_LDR_DATA struct
  …
  InLoadOrderModuleList        LIST_ENTRY <>
```

```
    InMemoryOrderModuleList     LIST_ENTRY <>
    …
PEB_LDR_DATA      ends
LDR_MODULE struct
    InLoadOrderModuleList       LIST_ENTRY <>
    InMemoryOrderModuleList     LIST_ENTRY <>
    InInitializationOrderModuleList LIST_ENTRY <>
    BaseAddress             dd ?
    EntryPoint              dd ?
    SizeOfImage             dd ?
    FullDllName             UNICODE_STRING <>
    BaseDllName             UNICODE_STRING <>
    …
LDR_MODULE ends
```

Поле LoaderData структури PEB є адресою структури PEB_LDR_DATA, поля InLoadOrderModuleList та InMemoryOrderModuleList якої використовуються для просування по шуканим 2-зв'язним спискам структур LDR_MODULE. При цьому поле InLoadOrderModuleList використовується для просування по списку, який несе інформацію про модулі, що мають бути завантажені до адресного простору додатку. Поле InMemoryOrderModuleList використовується для звернення до списку, який несе інформацію про модулі, що реально завантажені до адресного простору додатку. Поля BaseAddress та SizeOfImage структури LDR_MODULE містять інформацію відповідно про базову адресу модуля та розмір модуля у пам'яті.

Для визначення самого руткіт-модуля перевіряється належність адреси перехопленої API-функції до адресного простору кожного завантаженого модуля шляхом послідовного просування по двозв'язному списку, використовуючи поле InLoadOrderModuleList. За умови належності адреси перехопленої функції адресному простору модуля відповідна структура LDR_MODULE містить інформацію про руткіт-модуль. При цьому слід зауважити про можливість самовиключення руткіта з розглянутих двозв'язних списків структур модулів. У такому випадку виявити факт присутності руткіта можливо, проте визначити його програмний модуль шляхом аналізу списків модулів стає неможливим. Також слід зазначити існування можливості модифікації руткітом дійсного імені його програмного модуля в структурі LDR_MODULE, хендлу та інших полів, що забезпечують руткіту можливість захисту свого модуля та унеможливлюють керування ним чи визначення його місцезнаходження. У такому випадку пошук модуля здійснюється на етапі завантаження модуля до додатку, тобто до зміни руткіт-модулем структури LDR_MODULE. Для цього створюється процес шляхом прямої взаємодії з ядром ОС [4], що виключає можливість керування створенням процесу з боку руткіта. Після цього перехоплюється функція динамічного завантаження бібліотек LoadLibrary системної бібліотеки kernel32.dll. Аналізуючи імена всіх завантажених за допомогою LoadLibrary бібліотек, визначається ім'я руткіт-модуля. Такий метод є більш універсальним та не потребує ручного розбору структури PEB.

Оскільки сам модуль, який експортує функції для додатку, має і свою власну таблицю IAT, то руткіт може модифікувати і адреси входу функцій у IAT цього модуля. Наглядна схема перехоплення функції шляхом модифікації IAT модуля наведена на рис. 3.

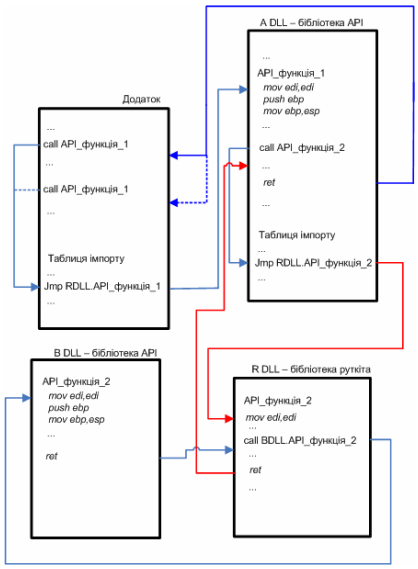

Рисунок 3 – Схема перехоплення API-функції шляхом модифікації IAT використовуваних додатком модулів

Визначення таких перехоплень реалізовано за вищеописаним методом, але аналізуються додатково і таблиці імпорту використовуваних додатком модулів.

Другий відомий метод перехоплення API-функцій базується на безпосередній модифікації їх машинного коду в адресному просторі додатку з можливістю передачі керування у код руткіт-модуля (splicing, сплайсінг). Схема такого перехоплення API-функції наведена на рис. 4.

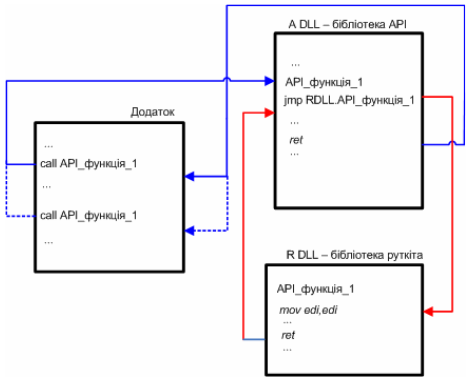

Рисунок 4 – Перехоплення API-функції шляхом модифікації машинного коду (сплайсінг)

Визначення перехоплення функцій за допомогою сплайсінгу ускладнене тим, що код перехоплення може бути записаний майже у будь-яке місце коду функції. Проте найчастіше руткіти використовують модифікацію лише перших п'яти байтів функції, які містять її пролог. Пролог більшості API-функцій, за

винятком функцій з бібліотеки ntdll.dll і декількох інших, має вигляд:

```
mov edi,edi
push ebp
mov ebp,esp
```

Наявність такого прологу дає змогу руткітам перехоплювати функції для своїх потреб.

Для визначення перехоплень такого типу перевіряється незмінність прологу кожної функції.

Перехоплення функцій методом сплайсінгу можливе не лише в бібліотеках, які експортують API-функції, а й у системній бібліотеці ntdll.dll, функції якої вони, у свою чергу, імпортують. Багато функцій системної бібліотеки ntdll.dll є транзитними точками входу до сервісів ядра ОС і мають узагальнений код:

```
MOV EAX, FunctionNumber
MOV EDX, AddrSysenterFunction
CALL DWORD PTR DS:[EDX]
RETN FunctionParametersCount*4
```

Тут FunctionNumber – номер функції, що експортується ядром системи; AddrSysenterFunction – адреса функції, що виконує передачу керування режиму ядра через шлюз; FunctionParametersCount – кількість параметрів функції.

Отже, ігноруючи незначні зміни сигнатури прологу функцій ntdll.dll, пов'язані з відмінностями FunctionNumber, реалізовано контроль сплайсінгу їх коду за наявністю незміненого прологу.

Але окрім модифікації перших п'яти байтів функції, руткіти також застосовують метод модифікації коду в глибині функції, за межею п'яти байтів прологу. У цьому випадку перевірка функції за еталоном стає неможливою. Такий метод перехоплення зводиться до пошуку розробником руткіта деяких специфічних інструкцій, які можна замінити кодом виклику функції руткіта. Перезаписані інструкції API-функції виконуються у коді руткіту для забезпечення непорушності виконання функції. Таке перехоплення здійснюється цілеспрямовано з урахуванням версії ОС та її встановлених оновлень.

Протидію цьому методу перехоплення функцій API організовано шляхом обчислення контрольної суми бібліотеки API в пам'яті та порівняння її з відомою контрольною сумою «чистої» бібліотеки. Відмінність указаних контрольних сум дає змогу стверджувати про порушення оригінального коду бібліотеки.

Код «чистої» бібліотеки API-функцій береться з власноруч створеного процесу, який уникнув атаки руткіту. Такий процес створюється без використання викликів функцій Win API шляхом використання Native API викликів [5], що ускладнює перехоплення, а також шляхом прямої взаємодії з ядром ОС [4], а це повністю виключає можливість перехоплення руткітом.

Для визначення руткіт-модуля знаходяться змінені інструкції в бібліотеці шляхом порівняння коду бібліотеки з «чистим» кодом. Далі дизасемблюється змінена частина для визначення адреси функції руткіта. Руткіт-модуль за знайденою адресою визначається вже описаним методом.

ВИСНОВКИ. На основі аналізу шляхів впровадження руткітів режиму користувача операційної системи Windows NT запропоновано методи реалізації самоконтролю програм для підвищення ступеня їх надійності при експлуатації в умовах активності шкідливого програмного забезпечення, що використовує модифікацію ланцюга системних викликів.

Запропоновані методи не передбачають використання функцій системного програмного інтерфейсу, що підвищило їх працездатність при виявленні EPM-руткітів.

Виконана програмна реалізація запропонованих методів, яка експериментально довела їх працездатність і може використовуватися розробниками програмного забезпечення на стадії кодування для протидії багатьом відомим на сьогодні руткіт-технологіям режиму користувача.

Запропоновані методи і їх програмна реалізація можуть бути використані на етапі проектування програмного забезпечення шляхом прозорого стикування із системою моніторингу та контролю.

ЛІТЕРАТУРА

1. Ховард М., Лебланк Д. Защищенный код. /Пер. с англ. – М.: Издательско-торговый дом «Русская Редакция», 2004. − 704 с.

2. Зайцев О.В. Rootkits, Spyware/Adware, Keyloggers&Backdoors: обнаружение и защита. – СПб.: БХВ-Петербург, 2006. – 304 с.

3. Казарин О. В. Теория и практика защиты программ. – М.: МГУЛ, 2004. − 450 с.

4. Зілінський Ю.В., Бельська В.Ю., Юдіна А.Л. Захист і оптимізація програмного забезпечення шляхом прямих викликів сервісів ядра операційних систем Windows NT // Вісник Кременчуцького державного політехнічного університету імені Михайла Остроградського. – Кременчук: КДПУ, 2009. – Вип. 5/2009 (58), ч. 1. – С. 49–53.

5. Неббет Г. Справочник по базовым функциям API Windows NT/2000. – М.: Издательский дом «Вильямс», 2002. – 528 с.

REFERENCE

1. Howard M., LeBlanc D. Writing secure code. – Microsoft Corporation, 2002. – 800 p. [in Russian].

2. Zaycev O.V. Rootkits, Spyware/Adware, Keyloggers&Backdoors: detection and protection. – SPb.: BHV-Petersburg, 2006. – 304 p. [in Russian].

3. Kazarin O.V. Theory and practice of software protection. – M.: MGUL, 2004. – 450 p. [in Russian].

4. Zilinskiy U.V., Belska V.U., Udina A.L. Software protection and optimization by direct system services calls in Windows NT operating system // Transactions of the Kremenchuk Mykhailo Ostrohradskyi state polytechnic university. – Kremenchuk: KSPU, 2009. – Ed. 5/2009 (58), P. 1. – P. 49–53 [in Ukrainian].

5. Nebbet G. Windows NT/2000. Native API Reference. – New Riders Publishing, 2000. – 482 p.

Стаття надійшла 20.01.2011.
Рекомендована до друку
д.т.н., проф. Гученком М.І.