УДК 004.492

О. В. Кавун

ЗАХИСТ ПРОГРАМ ЗА ДОПОМОГОЮ САМОМОДИФІКОВУВАНОГО КОДУ

Вінницький національний технічний університет

Анотація: *У статті представлено дослідження та основні аспекти використання самомодифіковуваних кодів в захисних механізмах програм. Розглянуто вразливості та недоліки методів, що базуються на застосуванні самомодифіковуваних кодів для захисту.*

Ключові слова: *самомодифіковуваний код, самомодифікація, механізм захисту.*

Abstract: *The article describes the main aspects and use self-modify codes in protective mechanisms programs. Also discussed vulnerabilities and weaknesses of methods use self-modify codes for protection..*

Keywords: *self-modify code, selfmodification, protection mechanism.*

У деяких джерелах використання самомодифіковуваного коду вважається поганим прикладом програмування, звинувачуючи його у відсутності переносимості, поганій сумісності з різними операційними системами, необхідності обов'язкових звернень до асемблеру. Дана стаття має ціль – показати, що ці думки невірні. Низька ефективність існуючих захистів програм і величезна кількість хакерів свідчать про необхідність посилення захисних механізмів доступними засобами, в тому числі самомодифіковуваним кодом.

Самомодифіковуваним кодом можна назвати прийом програмування, при якому програма створює або змінює частину свого програмного коду під час виконання. Такий код, зазвичай, використовують в програмах, написаних під процесор з фон-нейманівською організацією пам’яті [1].

Під час процесу модифікації можна зробити поділ на модифікацію при ініціалізації та модифікацію на льоту. В першому випадку модифікація проводиться один раз перед запуском змінного коду, а в другому - змінюється стан програми під час її виконання. В обох випадках зміна проходить безпосередньо в машинному коді, коли нові інструкції перезаписують старі.

Зрозуміло, що недоліком такого коду є те, що при модифікації машинних команд у пам'яті модифікується кеш даних. Потім відбувається екстренне скидання кодового кеша і перезавантаження змінених кеш-лінійок, на що витрачається досить велика кількість процесорних тактів.

Самомодифіковуваний код не слід виконувати у глибоко вкладеному циклі, оскільки це дуже сповільнить роботу програми [1].

Самомодифікація не перешкоджає трасуванню і для налагоджувача вона є повністю "прозорою". Зі статичним аналізом дещо складніше. Дизасемблер відображає програму у тому вигляді, в якому вона була отримана на момент зняття дампа або завантаження вихідного файлу, негласно розраховуючи на те, що жодна з машинних команд не зазнає змін в ході свого виконання. В іншому випадку реконструкція алгоритму буде виконана. Однак, якщо факт самомодифікації буде виявлено, скорегувати дизасемблерний лістинг не складе великих труднощів.

Більш прогресивні захисні технології базуються на динамічному зашифруванні коду. А зашифрування – це один з різновидів самомодифікації. Очевидно, що аж до того моменту, поки двійковий код не буде повністю розшифрований, для дизасемблювання він непридатний. А якщо розшифрувальник нашпигований антивідлагоджувальними прийомами, безпосереднє налагодження стає неможливим також.

Статичне зашифрування, що характерне для більшості навісних протекторів в даний час, визнана абсолютно безперспективною. Дочекавшись моменту завершення розшифрування, хакер знімає дамп і потім досліджує його стандартними засобами. Захисні механізми так чи інакше намагаються цьому протистояти. Вони спотворюють таблицю імпорту, затирають PE-заголовок, встановлюють атрибути сторінок в NO_ACCESS, проте для досвідчених хакерів такі способи захисту не є проблемою. Будь-хто, навіть найвитонченіший навісний протектор, вручну легко зніметься, а для деяких існують і автоматичні зламними [2].

Ні в який момент часу весь код програми не повинен бути розшифрований цілком. Потрібно виконувати правило - розшифровуючи один фрагмент, зашифровувати інший. Причому розшифрувальник повинен бути сконструйований так, щоб хакер не міг використовувати його для розшифрування програми. Це типова вразливість більшості захисних механізмів. Хакер знаходить точку входу в розшифрувальник, відновлює його прототип, і пропускає через нього всі зашифровані блоки, отримуючи на виході готовий до використання дамп. Причому, якщо розшифрувальник являє собою тривіальний XOR, хакеру буде достатньо визначити місце зберігання ключів, а розшифрувати програму він зможе і сам [3].

Щоб цього не сталося, захисні механізми повинні використовувати поліморфні технології та генератори коду. Автоматизувати розшифрування програми, що складається з декількох сотень фрагментів, зашифрованих криптограмами, згенерованими на "льоту", практично неможливо. Однак і реалізувати подібний захисний механізм досить складно.

У Гарвардській архітектурі пам'ять для коду і пам'ять для даних розділені. Відповідно, в них сильно ускладнюється робота самомодифіковуваного коду. Хоча архітектура x86 визначена як фон-нейманівська, більшість сучасних процесорів мають роздільні області кеша для коду та для даних. При цьому кеш коду не підтримує запис, і при зміні закешованої ділянки пам'яті може знадобитися або апаратно проведене часткове або повне скидання кешу коду (x86) або явна інструкція процесору на скидання кешу коду. Через це щойно змінений код може виконуватися повільніше, або вимагати додаткових команд для правильної роботи. Також зміна коду скидає конвеєр процесора [3].

Також, деякі ідеї Гарвардської архітектури реалізуються в ОС і в процесорах. У цих реалізаціях окремі фрагменти пам'яті можуть бути помічені як невиконувані або як виконувані, але немодифіковані, тобто код без права на зміну. Використання самомодифіковуваного коду в таких програмних середовищах ускладнюється, оскільки його доводиться розташовувати або в незахищеній області пам'яті, або явно відключати захист для зміни коду [1, 3].

Процесори сімейства Pentium відстежують модифікацію команд, які вже перебувають на конвеєрі, і тому програмна довжина конвеєра дорівнює нулю. Як наслідок - захисні механізми конвеєрного типу, потрапивши на Pentium, помилково вважають, що завжди виконуються під відлагоджувачем. Це цілком документована особливість поведінки процесора, яка мабуть збережеться і в наступних моделях.

Використання самомодифіковуваного коду з формальної точки зору цілком законно. Однак слід пам'ятати, що надмірне зловживання останнім негативно впливає на продуктивність програми, яку ми захищаємо. Кодовий кеш першого рівня доступний тільки на читання і прямий запис в нього неможливий. При модифікації машинних команд у пам'яті в дійсності модифікується кеш даних. Потім відбувається екстрений скидання кодового кеша і перезавантаження змінених кеш-лінійок, на що витрачається досить велика кількість процесорних тактів. Ніколи не слід виконувати самомодифіковуваний код у глибоко вкладеному циклі, тому, що це дуже сповільнить роботу програми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Крис Касперски. Хакер. / Андрей Комаров, Степан Ильин, Леонид Стройков, Сергей Яремчук, Денис Колесниченко. — 2003., - 104-110 с - Режим доступу до ресурсу : http://book.uz/wp-content/uploads/2010 .
2. Домарев В. Безопасность информационных технологий. Методология создания систем защиты / В. Домарев., 2000. – 688 с. - Режим доступу до ресурсу : http://www.twirpx.com/file/26250.
3. Касперски К. Компьютерные вирусы изнутри и снаружим / Крис Касперски. – Питер, 2006. – 526 с.

Кавун Олег Вікторович факультет інформаційних технологій так комп'ютерної інженерії, студент групи БС-14МС, Вінницький національний технічний університет, Вінниця, olegskavun@mail.ru.
Каплун Валентина Аполінаріївна, ст. викл. кафедри захисту інформації, Вінницький національний технічний університет, Вінниця, valuka8379@gmail.com

Kavun Oleg Faculty of Information Technology and Computer Engineering, student group BS-14MS, Vinnytsia National Technical University, Vinnytsia, olegskavun@mail.ru.
Kaplun Valentina Apolinariyvna, Vinnytsia National Technical University, Vinnytsia, valuka8379@gmail.com.