УДК 004.056.5
DOI: 10.31673/2409-7292.2018.032832

Довбешко С.В., Толюпа С.В., Труш О.В.

ДОДАТКОВИЙ ЗАХИСТНИЙ КОД, ОБФУСКОВАНИЙ ВІРТУАЛЬНОЮ МАШИНОЮ

Обфускація коду є важливим методом захисту від зворотної розробки програмного забезпечення (ПЗ) та для того, щоб захистити свою інтелектуальну власність. Обфускований код чи програму важко розробити зворотно, тому що аналізуючи машинні інструкції програми: статичні або динамічні, важко прослідкувати програмну логіку, котра прихована в інтерпретованому байт-коді. В статті було показано, що обфускований код, розроблений з використанням выртуальної машини (ВМ), задовольняє більшість критеріїв оцінки ефективності обфускованого коду.

Ключові слова: обфускація, інтелектуальна власність, шифрування, захист інформації.

У цифровому світі проблема захисту проприєнтаного програмного забезпечення (ПЗ) від зворотної розробки була завжди актуальною.

Розробники чи ІТ-компанії завжди прагнули забезпечити надійний захист свого продукту від неправомірного копіювання, або запобігти викраденню інтелектуальної власності (ІВ), приміром, унікально створених алгоритмів або модулів, що були розроблені компанією.

Раніше, захист інформації та даних забезпечувався використанням брандмауера та шлюзів в самій операційній системі (ОС) або в мережі. Проте, для захисту від сторонніх структур, кращою ідеєю є використання цих методів і механізмів у прикладному ПЗ.

Одним з таких методів є обфускація, що є новою областю вивчення у захисті ПЗ і набирає все більше значення в епоху цифрових технологій.

Обфускація складається з перетворення коду, який ускладнює розуміння програми, змінюючи структуру програми, і зберігаючи при цьому вихідні функціональні можливості. Саме такий підхід забезпечує захист ПЗ від несанкціонованого зворотного проектування.

Серед існуючих методів, які використовуються для захисту коду від різних атак, обфускація є однією з найпопулярніших альтернатив, аби перешкодити розумінню або підробці коду. Таким чином, обфускація коду в значній мірі – це рішення і множинна кількість методів, що застосовують з метою захисту коду.

У відповідності до деяких інших методів обфускація реалізується за допомогою оригінального коду, що унеможливлює отримання алгоритму або логіки коду під час атаки.

Методи обфускації також включають упорядкування коду, що охоплює: перетворення значимих імен ідентифікаторів, їх заміна в оригінальному коді на імена, які не мають сенсу (перейменування ідентифікаторів), внесення небажаного коду, умовні стрибки, переходи, змінне переназначення, об’єднання локальних цілих чисел, генерація підробленого коду середнього рівня, та подавлення констант.

Обфускація, загалом, відрізняється від шифрування. Перед усім, обфускація не вимагає будь-яких зворотних перетворень. Також, зловмиснику не потрібно постійно шукати вихідний код, тому що атака може бути успішною, без наявності вихідного коду ПЗ. Шифрований текст буде некорисним без ключа, тому що заплутана програма може працювати без будь-якої додаткової інформації.

Одним з відомих прикладів вкраденого ПЗ є розробка компанії Phoenix Technologies Ltd. власного BIOS, що був сумісний з проприєнтарним BIOS від компанії IBM. Під час дослідження специфікації розробки і принципів роботи BIOS від компанії

IBM, команда розробників Phoenix Technologies Ltd. створила його власну версію BIOS, яку пізніше почала продавати.

Обфускація, або ж заплутування коду, це приведення вихідного коду програми до виду, який складно піддати програмному аналізу чи людському розумінню. Концепція, котра слідує підходу до обфускації, полягає в безпеці через неясність (security through obscurity) [2].

Існує ряд причин чому використовують обфускацію коду, наприклад, через заплутування можна покращити захищеність коду, запобігти втручанню в код чи програму, або захистити від викрадення ІВ.

Ще однією перевагою, є те, що обфускація коду дає багато додаткових можливостей для підвищення потенційного потенціального рівня захисту ПЗ.

Базова обфускація коду використовує просту математичну функцію «виключна диз'юнкція (*англ.* eXclusive OR)» XOR. Ця функція дозволяє з легкістю сховати інформацію програми від користувача, використовуючи ключ для шифрування коду, який постійно повторюється.

Формально, якщо дана програма P, яка складається з об'єктів вихідного коду $\{S_1, \dots, S_k\}$ (класи, методи, змінні, структури даних), перетворення $T = \{T_1, \dots, T_n\}$, то програма $P' = T(P)$ буде обфускованою програмою P, якщо:

1) P' має ту ж саму видиму поведінку (так звані динамічні властивості), що і P, тобто перетворення T зберігає семантику;

2) не явність P' максимізована, тобто розуміння і зворотна розробка P' займає більше часу ніж зворотна розробка P при використанні одних і тих же підходів до зворотної розробки;

3) ефективність перетворення $T_i(S_i)$ максимізована, тобто, вкрай важко розробити автоматизовану програму для відміни перетворення, а також використання такої програми дуже затратно у часі;

4) схожість статичних символів властивостей $T_i(S_i)$ і S_i максимізована;

5) відмінність продуктивності P' і P мінімізована.

Програмний код може бути представлений у двійковому вигляді (послідовність байтів, що представляють собою так званий машинний код, який отримується після компіляції вихідного коду програми), або у вихідному вигляді (текст, що містить послідовність інструкцій будь-якої мови програмування, що буде зрозумілим людині, такий текст згодом піддасться компіляції або інтерпретації на комп'ютері користувача).

Процес обфускації може бути здійснений над будь-яким з вище перерахованих видів представлення ПЗ, тому прийнято виділяти наступні рівні процесу обфускації:

- нижчий рівень, це коли процес обфускації здійснюється над асемблерним кодом програми, або навіть безпосередньо над двійковим файлом програми, що зберігає машинний код;
- високий рівень, це коли процес обфускації здійснюється над вихідним кодом програми написаної на мові вищого рівня.

Здійснення обфускації на нижчому рівні рахується менш комплексним процесом, але при цьому важче реалізувати з ряду причин. Одна з цих причин полягає у тому, що мають бути враховані особливості роботи більшості процесорів, так як спосіб обфускації, допустимий в одній програмній архітектурі, може виявитись неприйнятним в іншій.

Також, станом на сьогоднішній день процес обфускації низького рівня мало досліджений.

Більшість існуючих алгоритмів і методів обфускації (включаючи ті, котрі будуть розглянуті нижче), можуть бути використані для здійснення процесу обфускації як на нижчому, так і на вищому рівні.

Також, іноді може бути неефективно піддавати обфускації весь код програми (наприклад, тому що в результаті може значно знизитись час виконання програми), в таких випадках, доцільно здійснювати обфускацію тільки найбільш важливих ділянок коду.

Оцінка процесу обфускації.

Існує багато методів визначення ефективності застосування того чи іншого процесу обфускації, що застосовується до конкретного програмного коду.

Такі методи прийнято розподіляти на дві групи:

- аналітичні;
- емпіричні.

Аналітичний метод ґрунтується на трьох характеристиках, що визначають наскільки ефективний той чи інший процес обфускації:

- стійкість – вказує на ступінь складності здійснення зворотної розробки коду, що пройшов процес обфускації;
- еластичність – вказує на те, наскільки надійним є даний процес обфускації, чи здатен захистити код від використання деобфускатора;
- вартість перетворення – дозволяє оцінити наскільки більше системних ресурсів витрачається для виконання коду, що пройшов процес обфускації.

Такі методи ефективно використовувати при порівнянні різноманітних алгоритмів обфускації, але при цьому, ці методи не можуть дати відповіді, наскільки ефективним є використання того чи іншого алгоритму, саме до заданого програмного коду.

Емпіричний метод може дати відповідь, на таке питання, який метод буде більш ефективним для того, чи іншого програмного коду, так як даний метод базується на статистичних даних, що отримуються у результаті великої кількості досліджень.

Для проведення одного з таких досліджень потрібна група людей (які знайомі зі зворотною розробкою), фрагмент коду, програми, котру потрібно захистити та набір різноманітних алгоритмів обфускації.

Результати такого дослідження будуть включати в себе мінімальну кількість часу, що знадобиться групі людей, для того, щоб вивчити кожен фрагмент коду, що пройшов один з алгоритмів обфускації.

Алгоритми процесу обфускації.

Алгоритм обфускації в більшості випадків розглядається як алгоритм, якого має притримуватись обфускатор (незалежна програма, що здійснює процес обфускації, над переданим їй кодом).

Алгоритм Колберга.

Даний алгоритм оперує наступними вихідними значеннями:

- програма «A», що складається з вихідних або об’єктивних (двійкових) файлів «{C1, C2}»;
- стандартні бібліотеки, що використовуються програмою «{L1, L2}»;
- набір процесів, що трансформуються «T {T1, T2}»;
- певний фрагмент коду «S», що витягується з програми «A», і який безпосередньо буде схильний до трансформації;
- набір функцій «E {E1, E2}», що будуть визначати ефективність застосування певних процесів, що трансформуються «{T1, T2}» до фрагменту коду «S»;
- набір функцій «I {I1, I2}», що будуть визначати важливість фрагмента коду «S», і в залежності від цього будуть задавати певне значення змінної «RequireObfuscation» (чим «S» важливіше, тим ця змінна буде зберігати більше значення) ;

- дві числові змінні «$AcceptCost$» > 0, «$RequireObfuscation$» > 0, де перша змінна, зберігає інформацію про доступне максимальне збільшення системних ресурсів до потрібних програмі, «А» після того, як вона піддасться обфускації; а друга змінна буде зберігати значення необхідне для здійснення обфускації (чим важливіше фрагмент коду «S», тим це значення має бути більше).

Алгоритм Колберга має таку послідовність операцій:

1) завантаження елементів «{C1, C2}» програми «А»;
2) завантаження бібліотек «{L1, L2}»;
3) здійснення обфускації над програмою «А», шляхом виділення фрагмента коду «S» і визначення найбільш ефективного процесу трансформації для нього. Цей етап повторюється до тих пір, поки не буде досягнутий необхідний рівень обфускації «RequireObfuscation» або припустиме збільшення ресурсів «AcceptCost»;
4) генерація трансформованої програми «А`».

Алгоритм Колберга вважається загальним алгоритмом здійснення процесу обфускації (тобто, він не визначає, як саме має здійснюватися той чи інший метод обфускації).

Chenxi Wang's алгоритм

У якості вхідних даних алгоритм здійснює процедуру, що написана на мові високого рівня. Процес обфускації такої процедури складається з трьох етапів:

- створення графу потоку управління цієї процедури (граф задається множиною блоків та множиною зв'язків, що з'єднують процедуру), після чого граф розбивається, шляхом заміни циклічних конструкцій типу «if (умова) goto»;
- нумерація всіх блоків у графі та додання у код процедури змінної, що зберігає номер наступного блока, що виконується;
- приведення графа до однорідного («плоского») виду.

Вище описаний варіант алгоритму обфускації («Chenxi Wang`s algorithm»), є не стійким, так як виявити наступний блок, що виконується, не важко. Тому, для підвищення стійкості вводять масив, що містить крім номерів блоків, не потрібну інформацію.

Процес обфускації можна класифікувати по виду, в залежності від способу модифікації коду програми.

Лексична обфускація – є найбільш простою. Полягає у форматуванні коду програми, зміні структури коду, таким чином, щоб він став нечитабельним, менш інформативним та складним для вивчення.

Обфускація такого виду включає в себе:

- видалення усіх коментарів у коді програми, або їх заміна на дезінформуючі;
- видалення різноманітних пропусків, відступів, що зазвичай використовують для кращого візуального сприйняття коду програми;
- заміна імен ідентифікаторів (імена змінних, масивів, структур, хешів, функцій, процедур) на довільні довгі набори символів, котрі важко сприймаються людиною;
- додання різних зайвих (сміттєвих) операцій;
- зміна положення блоків (функцій, процедур) програми, таким чином, щоб це ні в якому разі не вплинуло на виконання програми;

Зміну глобальних імен ідентифікаторів слід проводити у кожній одиниці трансляції (один файл вихідного коду), так щоб вони мали однакові імена (в іншому випадку програма, що потребує захисту може стати не функціональною). Також, варто врахувати специфічні ідентифікатори, що прийняті в тій чи іншій мові програмування, на котрому написана програма, що потребує захисту, імена таких ідентифікаторів краще не змінювати.

Дана обфускація програмного коду в порівнянні з іншими методами, дозволяє відносно швидко привести вихідний код програми в нечитабельний стан. Одним з

недоліків є те, що вона ефективна тільки для здійснення обфускації високого рівня. Такий метод обфускації пов'язаний з трансформацією структур даних. Цей метод є більш складним та часто використовуваним.

Список використаних джерел

1. Moy, R., A Case Against Software Patents. Santa Clara High Technology Law Journal, 2000, pp. 72-73.
2. Kevin Coogan, G. L.,иDeobfuscation of virtualization-obfuscated software: a semantics-based approach. Proceedings of the 18th ACM conference on Computer and communications security, 2011, p. 1.
3. Rofl, R., Unpacking virtualization obfuscators. 3rd USENIX Workshop on Offensive Technologies, 2009, pp. 1-2.
4. Abdullah Sheneamer, Swarup Royc, Jugal Kalita, A detection framework for semantic code clones and obfuscated code, 2017.
5. Kaiyuan Kuang, Zhanyong Tang, Xiaoqing Gong, Dingyi Fang, Xiaojiang Chen, Zheng Wang, Enhance virtual-machine-based code obfuscation security through dynamic bytecode scheduling, 2017.

Надійшла: 2.07.2018 Рецензент: к.т.н. Курченко О.А.