Застосування методів обфускації для утруднення декомпіляції коду програм

Є.М. Рєхіна, *студент,*
О.П. Дóренський, *викладач*
Кіровоградський національний технічний університет

Обфускацією називають сукупність методик і засобів, направлених на утруднення аналіза програмного кода [1]. Заплутаною (obfuscated) називається програма, що на всіх припустимих для вихідної програми вхідних даних видає той же самий результат, що й оригінальна програма, але більш складна для аналізу, розуміння й модифікації. Заплутана програма виходить у результаті застосування до вихідної програми перетворень, що заплутують (obfuscating transformations) [2].

Завдання заплутування й аналізу заплутаних програм мають три аспекти:

– теоретичний, що включає в себе розробку нових алгоритмів перетворення графа потоку керування або трансформації дані програми, а також теоретичну оцінку складності їхнього аналізу й розкриття;

– прикладний аспект містить у собі розробку конкретних методів заплутування (розплутування), тобто найкращих комбінацій алгоритмів, емпіричний порівняльний аналіз різних методів, емпіричний аналіз стійкості методів тощо;

– психологічний, поки не піддається формалізації, але не може ігноруватися.

"Заплутування" коду може здійснюватися на рівні алгоритму, вихідного тексту й/або ассемблерного тексту. Для створення заплутаного ассемблерного тексту можуть використовуватися спеціалізовані компілятори, що використовують неочевидні або недокументовані можливості середовища виконання програми. Існують також спеціальні програми, що здійснюють обфускацію, і називаються обфускаторами.

Метою науково-дослідної роботи є дослідження методів обфускації з метою їх практичного застосування для розробки програмного забезпечення утруднення декомпіляції коду програм.

За результатами здійснених теоретичних досліджень [1, 2] варто зазначити, що теоретичні висновки повинні підтверджуватися результатами практичного застосування запропонованих методів. Тому у даній роботі досліджується прикладний аспект задачі заплутування коду програм.

У наш час широко поширені мови програмування, такі як Java, у яких формою програми, що виконується, є не машинний код для деякого типу процесорів, але машинно-нейтральне подання. Завдання декомпіляції програми з такого подання назад у програму мовою Java значно простіше, ніж декомпіляція з машинного коду. Існує велика кількість декомпіляторів для мови Java, які розповсюджуються вільно або комерційно, що спрощує несанкціоноване використання, зворотну інженерію й модифікацію Java-програм. Тож, як однин зі способів боротьби з декомпіляцією можна використати заплутування програм.

На сьогодні відомо понад два десятки різних заплутувачів Java-програм. Прості заплутувачі видаляють таблиці символів і інформацію про відлагодження зі скомпільованих класів і заміняють вихідні імена методів безглуздими короткими іменами. У результаті цього розмір файлів зменшується до 50%, а швидкість виконання програми значно зростає, тому таке заплутування може розглядатися і як один зі способів оптимізації програм.

Більш нові заплутувачі програм мовою Java, а також заплутувачі програм на інших мовах програмування, виконують перетворення графа потоку керування програми і її структур даних. Методи, використовувані в них, як правило, підібрані емпірично й слабко обґрунтовані теоретично.

Можливі різні рівні постановки завдання заплутування й аналізу перетворень, що заплутують. По-перше, заплутування може розглядатися в рамках мови Java. У цьому випадку вихідна програма, написана на Java, і заплутана програма також написана тією ж мовою. Однак мова Java допускає тільки структурні програми, тобто графи потоку керування Java-програм завжди зводяться, що істотно обмежує діапазон застосовних перетворень графа потоку керування.

Можлива постановка завдання заплутування на більш низькому рівні, коли заплутується програма мовою асемблера або навіть об'єктна програма в машинному коді (в останньому випадку вона повинна генеруватися спеціальним компілятором, що заплутує). В ассемблерних і об'єктних програмах можна використовувати специфічні особливості роботи цільової машини, домігшись того, що відновлення програми на Delphi або Сі буде вкрай утруднене. Але з іншого боку методи заплутування, застосовні до однієї архітектури, можуть виявитися незастосовні до іншої архітектури.

Слід відзначити, що проблема низькорівневого заплутування мало досліджена, а універсального заплутувача не існує.

Отже, обфускація може ефективно застосовуватись для:

– Ускладнення дослідження коду. Декомпіляція програм Java і .NET досить проста. У цьому випадку обфускатор надає неоціненну допомогу тим, хто хоче сховати свій код від сторонніх очей. Найчастіше після обфускації декомпільований код повторно не компілюється. Обфускація HTML допомагає спамерам: на поштовому клієнті, що здатний відображати HTML, текст читається, але антиспамовий фільтр, що опрацьовує вихідний HTML-файл, пропускає небажане повідомлення, не бачачи в ньому заборонного рядка.

– Оптимізація. В мовах, що інтерпретують, обфускаційований код займає менше місця і найчастіше виконується швидше, ніж вихідний. Сучасні обфускатори також заміняють константи числами, оптимізують код ініціалізації масивів, і виконують іншу оптимізацію, що на рівні вихідного тексту провести проблематично або неможливо. Проблема зменшення розміру програм важлива, наприклад, під час програмування для стільникових телефонів на J2ME, де розмір програми серйозно обмежений. Обфускація JavaScript зменшує розмір HTML-файлів і, відповідно, прискорює процес завантаження.

У той же час слід відзначити ряд недоліків заплутування програми:

– Втрата гнучкості коду. Код після обфускації може стати більш залежним від платформи або компілятора.

– Труднощі налагодження. Обфускатор не дає можливості з'ясувати, що виконує код, але й не дає розробникові налагоджувати його. При налагодженні доводиться відключати обфускатор.

– Недостатня безпека. Хоча обфускація допомагає зробити розподілену систему більш безпечною, не варто обмежуватися тільки нею. Жоден з існуючих обфускаторів не гарантує складності декомпіляції й не забезпечує безпеки на рівні сучасних криптографічних схем; цілком імовірно, що ефективний захист неможливий.

– Помилки в обфускаторах. Сучасний обфускатор – складний програмний комплекс. Найчастіше в обфускатори, незважаючи на ретельне проектування й тестування, вкрадаються помилки. Так що є ненульова ймовірність, що код, який пройшов через обфускатор, взагалі не буде працювати. І чим складніше розроблювальна програма, тим більшою ця ймовірність.

– Виклик класу по імені. Більшість мов із проміжним кодом можуть створювати або викликати об'єкти по іменах їхніх класів. Сучасні обфускатори дозволяють зберегти зазначені класи від перейменування, однак подібні обмеження скорочують гнучкість програм.

У доповіді наводиться детальний аналіз методів обфускації, їх недоліки та переваги, а також презентується програмна реалізація системи утруднення декомпіляції коду програмного забезпечення.

Список літератури

1. Касперски К. Обфускация и ее преодоление. [Електронний ресурс] – www.delltroy.homeip.net.
2. Методы обфускации. Журнал "Хакер", Спец. вып. № 66, 2009. – С. 8-13.
3. Смірнов О.А. Основи захисту інформації: Навч. пос.. / А.О. Смірнов, Л.Г. Віхрова, С.І. Осадчий, В.Ю. Ковтун, Є.В. Мелешко. – Кіровоград: РВЛ КНТУ, 2011. – 322 с.

Огляд технології розробки програмного забезпечення Rational Unified Process (RUP)

Н.В. Підгорна, *студент,*
О.К. Конопліцька, *асистент*
Кіровоградський національний технічний університет

На сьогоднішній день у світі існує величезна кількість процесів для створення ПЗ. Тим не менш, саме технологій, що розглядають повний життєвий цикл проекту розробки ПЗ, що поєднують в собі науковий підхід, серйозну базу досліджень і мають історію реального використання та адаптації, відносно небагато. З методологій і технологій, які отримали певне визнання на даний момент, можна назвати наступні: Datarun, CMM, Microsoft Solution Framework (MSF), Oracle Method, Rational Unified Process (RUP), SADT (IDEFx).

Особливе місце в цьому списку займає технологія Rational Unified Process (RUP), яка претендує на роль світового корпоративного стандарту. RUP представляє собою, програмний продукт, розроблений компанією Rational Software, яка зараз знаходиться в складі IBM.

Основними принципами RUP технології є:

1) Ітераційний та інкрементний (нарощуваний) підхід до створення ПЗ.

2) Планування і управління проектом на основі функціональних вимог до системи - варіантів використання.

3) Побудова системи на базі архітектури ПЗ.

Перший принцип є визначальним. Відповідно до нього розробка системи виконується у вигляді декількох короткострокових міні-проектів фіксованої тривалості (від 2 до 6 тижнів), званих ітераціями. Кожна ітерація включає свої власні етапи аналізу вимог, проектування, реалізації, тестування, інтеграції і завершується створенням працюючої системи.

Ітераційний цикл грунтується на постійному розширенні та доповненні системи в процесі декількох ітерацій з періодичним зворотнім зв'язком і адаптацією модулів,