ОБФУСКАЦИЯ КАК ВСПОМОГАТЕЛЬНЫЙ ИНСТРУМЕНТ ПРИ ОБУЧЕНИИ СТУДЕНТОВ IT-СПЕЦИАЛЬНОСТЕЙ

Быцкевич Ю.И., Куликов С.С.

Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Беларусь, yuliya.bytskevich@gmail.com, kulikov@bsuir.by

Abstract. Demonstration is an important part of any learning process. A clearly presented example allows student to solve a task faster and better. Obfuscation is an excellent instrument for providing an example without giving a student the possibility to plagiarize code. Obfuscation can be applied to source or intermediate code or even for both of them.

В учебном процессе студента любой специальности важнейшую роль играют примеры. Имея наглядный образец, гораздо проще выполнить поставленную задачу так, как следует.

Особенно остро нехватку практических примеров ощущают студенты дистанционной формы обучения. Это провоцирует отсутствие у студента чёткого понимания о том, что и как именно следует реализовать, и, как следствие, замедление учебного процесса, вызываемое многократным переделыванием и проверкой заданий преподавателем.

Хорошим шагом для предотвращения подобных ситуаций было бы предоставление студентам наравне с заданием и демонстрационной программы, прототипа, на основании которого студент мог мы построить своё решение и проверить корректность его работы. Однако в современном мире не нужно быть опытным реверс-инженером для того, чтобы получить доступ к исходному коду большого числа программ. Рынок предлагает широкий выбор декомпилирующего ПО, в том числе бесплатного, что делает получение исходного кода демо-программ простой задачей. Очевидно, что это совершенно неприемлемо в учебном процессе, так как может спровоцировать студентов к плагиату кода. Так что перед предоставлением демонстрационных программ необходимо принять меры для защиты их исходного кода.

Возможные способы защиты исходного кода:

– выполнение программы полностью или частично на стороне сервера;

– шифрование кода;

– использование нативного кода;

– обфускация кода [1].

Первый вариант иначе можно описать как предоставление не программ целиком, а в виде сервисов, способных по запросу обращаться к узлу с работающей программой (через Интернет). Плюс в том, что у студента нет доступа к исполняемым файлам, а значит декомпиляцию провести нельзя. Минус заключаются в том, что из-за пропускной способности и задержек Интернет-канала приложение будет иметь худшую производительность чем если бы оно работало на стороне пользователя, к тому же это обязывает студентов иметь постоянный доступ в Интернет, что не всегда возможно.

Метод шифрования кода работает только если процесс шифрования/выполнения целиком поддерживается аппаратной платформой. Если код выполняется интерпретатором виртуальной машины, у студента будет возможность перехватить и декомпилировать расшифрованный код. А если расшифровка выполняется «на лету», выполнение программы потребует дополнительных ресурсов, то есть будет работать более медленно и затратно по памяти.

Под третьим вариантом подразумевается предоставление линейки программ, каждая их которых представлена нативным кодом той или иной архитектуры. Этот вариант можно назвать слишком трудоёмким с точки зрения разработчика демо-программы.

Четвёртое решение заключается в том, чтобы пропустить демо-приложение через обфускатор – программу, которая трансформирует исходную программу в такую, которая имела бы идентичную оригинальной функциональность, но была гораздо более сложной для понимания.

По одному из определений, сущность обфускации заключается в том, чтобы различными способами изменить код так, чтобы при сохранении своей функциональности он имел максимально сложный для понимания вид [2].

Минусом обфускации является то, что в общем случае запутанная программа имеет более или менее значительные накладные расходы – например, дополнительное время выполнения и/или затраты памяти. Фактически объём дополнительных ресурсов, необходимых обфусциованной программе, сильно зависит от того, какие именно запутывающие преобразования предоставляет обфускатор.

Все преобразования могут быть оценены по таким параметрам, как эффективность, эластичность и стоимость [3].

Мера эффективности показывает, насколько обфусцированный код стал более сложным для понимания человеком в сравнении с оригинальным [4]. Преобразование называют эффективным, если значение этого показателя больше 0.

Мера эластичности характеризует то, насколько хорошо преобразование противостоит атакам автоматического деобфускатора [4]. По эластичности преобразование может варьироваться от незначительного до необратимого.

Стоимость преобразования – это дополнительное время выполнения/расход памяти, которое преобразование вносит в обфусцированную программу. Можно классифицировать стоимость по шкале из четырёх измерений: бесплатно, дёшево, ощутимо, дорого [3].

Обфускацию исходных кодов обычно производят тогда, когда необходимо защитить некомпилируемый код. Например, JavaScript-код веб-приложений, к которому легко получить доступ из браузера.

В случае, когда используются компилируемые языки, их исходные коды не отправляют в открытый доступ, распространению подлежат уже скомпилированные исполняемые модули. Но и в этом случае присутствует уязвимость: если исполняемые модули представлены промежуточным кодом (например, платформ .NET и Java), с помощью специальных средств можно без проблем восстановить из промежуточного кода исходный и использовать его по своему усмотрению. Обычно в отношении программ, компилируемых в промежуточный, производится запутывание именно его (то есть исполняемых модулей, а не файлов исходного кода). Из такого кода извлечь с помощью декомпиляции оригинальные исходные коды не удастся [5].

Причина такого подхода в том, что промежуточный код – финальный продукт, который и будет подлежать распространению, в отличие от исходного кода, которому ещё предстоит пройти компиляцию. Компилятор производит над кодом ряд оптимизаций, поэтому если запутывать исходный код, некоторые из преобразований могут быть ликвидированы (например, недостижимый/мёртвый код).

Но обфускация промежуточного кода имеет и минусы. Самый главный из них – то что чем объёмнее и сложнее программа, тем больше шанс того, что преобразования обфускатора внесут ошибки в её работу. Причём эти ошибки будет сложно обнаружить, так как речь идёт о готовом исполняемом модуле (.exe). Также их сложно будет исправить, так как далеко не все разработчики столь же хорошо знакомы с промежуточным языком, как с исходным (например, далеко не все C#-разработчики смогут отладить IL-код). Может случиться и так, что программа после обработки обфускатором не запустится вовсе. Все зависит от качества написания используемого обфускатора, а также от набора и сложности преобразований, которые он производит.

В этом отношении запутывание исходных кодов имеет преимущества: можно производить более изощрённые преобразования (например, структур данных), оперируя более сложными абстракциями, и производить отладку запутанного кода, пользуясь средствами среды разработки. Неработающий код просто не скомпилируется, а среда разработки даст подсказки относительно мест и причин ошибок.

Проведя анализ плюсов и минусов подходов обфускации исходного и промежуточного кода, можно сделать вывод, что подходы не противоречат, а дополняют друг друга. В сочетании они дают возможность использовать достоинства обоих решений.

Таким образом, в качестве актуального и эффективного обфускатора может рассматриваться программа, реализующая двухступенчатое запутывание: сначала исходного кода, затем – промежуточного. Первый этап позволяет производить семантически более сложные преобразования, второй – преобразования по внесению дополнительной информационной нагрузки, которые нельзя выполнить в первом этапе из-за оптимизаций компилятора.

Примерный список запутывающих преобразований, который можно выделить для реализации в двухступенчатом обфускаторе:

1. Преобразования для исходного кода:

– удаление всех комментариев из кода;

– удаление отступов, переносов строки, пробелов (где это возможно);

– вставку дополнительных фигурных и круглых скобок (без нарушения хода выполнения программы);

– изменение названий пространств имён, классов, переменных, свойств, методов;

– вставка кодов функций в место их вызова;

– развёртка циклов;

– переплетение (слияние) функций;

– внесение кода, который может быть упрощён или удалён вовсе (избыточного кода);

– шифрование константных строк;

– в рамках одного класса объединение переменных одного типа в массив;

– изменение иерархии классов;

– преобразование сводимого графа потока управления к несводимому;

– удаление идиом языка;

– изменение времени жизни переменных;

– перевод статических данных в экземплярные;

– внесение фиктивных классов в проект и создание их экземпляров в исходном коде проекта, а также в других фиктивных классах [2,3].

2. Преобразования для промежуточного кода:

– реструктуризация графа потока управления (разбиение кода на блоки, перемешивание блоков, управление посредством элемента-диспетчера);

– нарушение порядка выполнения программы с использованием инструкций безусловного перехода;

– внесение кода, который никогда не будет выполняться, а только увеличивает информационную нагрузку (недостижимого кода);

– внесение кода, который может быть упрощён или удалён вовсе (избыточного кода) [3, 4].

В отличие от выполнения программ на стороне сервера, обфускация не может обеспечить абсолютной защиты от попыток обратной разработки. Поэтому в отношении ценного коммерческого ПО, возможно, лучше применить другие методы. Но в целях обучения, в отношении программ-примеров, этой меры защиты будет вполне достаточно.

Литература

1. Методы защиты от исследования программ // GRSU [Электронный ресурс]. – 2014. – Режим доступа: http://mf.grsu.by/UchProc/livak/arxiv_22102010/kursi/zaschita/antiisl.htm.

2. Wikipedia [Электронный ресурс] – Режим доступа: https://goo.gl/h5rB6V.

3. Christian Collberg, Clark Thomborson, Douglas Low. A Taxonomy of Obfuscating Transformations. – Department of Computer Science, The University of Auckland, 1997. – 36 с.

4. Чернов А. В. Анализ запутывающих преобразований программ. – Институт Системного программирования РАН, 2003. – 34 с.

5. Реверсинг и обфускация // Geekbrains [Электронный ресурс]. – 2016. – Режим доступа: https://geekbrains.ru/posts/Reversing_and_obfuscation.