Д. С. Варганов
ОАО КБ «ЮНИСТРИМ» (г. Москва)

СОСТОЯНИЕ И ПЕРСПЕКТИВЫ РАЗВИТИЯ ТЕХНОЛОГИИ ЗАЩИТЫ ПРОГРАММНЫХ ПРОДУКТОВ

В настоящее время средства защиты (СЗ) программного обеспечения (ПО) широко распространены и находятся в постоянном развитии благодаря расширению рынка ПО. Необходимость использования средств защиты ПО обусловлена следующим рядом проблем:

· несанкционированное использование ПО (копирование);

· незаконное использование алгоритмов, являющихся интеллектуальной собственностью автора, при написании аналогов продукта (промышленный шпионаж);

· несанкционированная модификация ПО (внедрение вредоносного для конечного пользователя кода);

· незаконный сбыт ПО (пиратство).

Россия, где, по данным независимого исследования IDC — ведущей мировой исследовательской и консалтинговой компании в сфере информационных технологий, уровень использования нелицензионного ПО составляет 83 % [1], а убытки от компьютерного пиратства в 2005 г. достигли 1,63 млрд долларов, является одним из приоритетных регионов для BSA (Business Software Alliance). Для сравнения, в среднем доля пиратского ПО в мире составляет 35 %. Исследование также показало, что образовательные программы, юридические прецеденты и в целом политика по борьбе с компьютерным пиратством, проводимая в странах с развивающейся экономикой, приносят практические результаты. Россия, несмотря на то что уровень использования нелицензионного ПО здесь по-прежнему остается одним из самых высоких в мире, также вошла в число стран, где отмечен существенный прогресс с точки зрения сокращения уровня компьютерного пиратства. По данным IDC, в 2005 г. в нашей стране уровень использования нелицензионного ПО сократился на 4 % — с 87 % до 83 %.

По прогнозам IDC, уровень компьютерного пиратства в России будет снижаться на несколько процентов каждый год. В первую очередь, этому будут способствовать усилия производителей программного обеспечения по разработке более гибких ценовых политик и по повышению качества поддержки конечных пользователей. С другой стороны, важную роль будут играть меры, направленные на снижение уровня пиратства в стране, принимаемые государством в лице законодателей и правоохранительных органов. Но, поскольку продолжается развитие технологий широкополосного доступа, а рынок информационных технологий по-прежнему растет, приток новых пользователей и увеличивающаяся доступность пиратских программ означают, что придется приложить еще большие усилия для того, чтобы уровень пиратства продолжал снижаться.

Пока российские компании уделяют недостаточное внимание вопросам безопасности. В большинстве случаев подход к защите информации сводится к тезису «чем дешевле, тем лучше». Наши компании выражают серьезную озабоченность проблемой, осознают таящиеся в ней опасности, однако не предпринимают ровным счетом никаких практических шагов к ее решению. Исследование InfoWatch «Внутренние ИТ-угрозы в России 2005» показало, что всего лишь 2 % организаций используют специализированные технические средства защиты. Вместе с тем 83 % респондентов подтвердили, что планируют их внедрение в течение ближайших трех лет, хотя в целом наблюдается увеличение расходов на информационную безопасность во всех развитых регионах [2].

Степень защищенности программного продукта критично влияет на прибыль по его реализации. Если продукт плохо защищен, после выхода пиратской версии прибыль резко падает. Продукту не удается завоевать свою долю рынка.

1. Решение проблемы защиты ПО

На рис. 1 приведена классификация средств защиты программного обеспечения. Средства защиты ПО можно классифицировать по ряду следующих признаков:

· по используемому механизму защиты (юридический, экономический, технический);

· по методу установки (устанавливаемые на скомпилированные модули, внедряемые в исходный код, комбинированные);

· по используемым методам при реализации систем защиты (использующие шифрование, использующие сложные логические механизмы, комбинированные);

· по методу реализации (программные, аппаратные);

· по типу вредоносного воздействия (от обратной инженерии, от несанкционированного доступа (НСД), от несанкционированного копирования);

· по типу средства анализа злоумышленника (средства защиты от статического анализа, средства защиты от динамического анализа, средства защиты от снятия дампа памяти в оперативной памяти ОП).

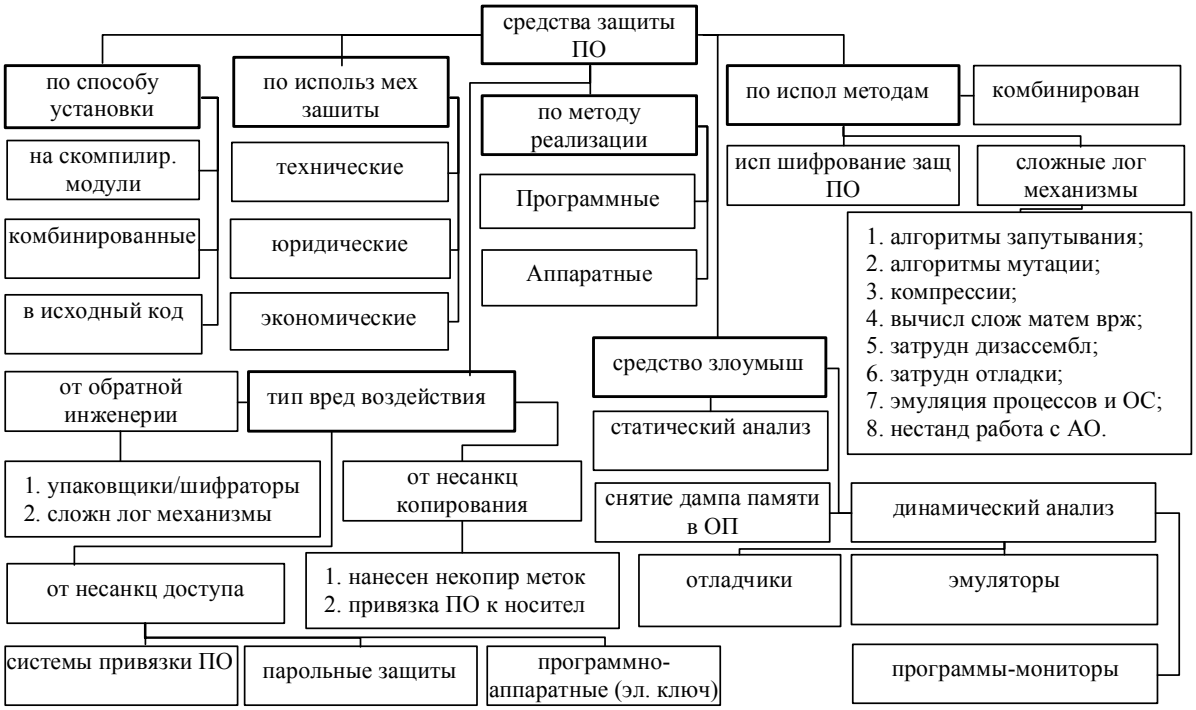

Рис. 1. Классификация средств защиты программного обеспечения

Стоит отметить, что зачастую защита программного продукта достигается за счет конечного пользователя. Системы защиты наносят вред компьютеру: ломаются диски в дисководах, устанавливаются драйверы с высоким приоритетом на выполнение (опасность заражения вирусами) и т. д. Большинство отрицательных факторов средств и методов защиты испытывает на себе как раз пользователь ПО.

2. Технический, юридический и экономический подходы

Возможны три подхода к решению проблемы защиты ПО. Один из них — решение проблемы с помощью технических средств защиты, включаемых в состав ПО. Такой подход направлен на то, чтобы техническими средствами затруднить или сделать невозможным использование программного продукта незаконным пользователем. Техническими средствами защиты программных продуктов можно реализовать несколько подходов к данной проблеме: противодействие копированию программы, противодействие запуску нелицензионной копии, снабжение программы скрытой информацией об управлении правами, что в случае необходимости дает возможность доказать авторство и незаконное использование.

Другой — использование юридической защиты, когда ПО не содержит технические СЗ, а сопровождается информацией об управлении правами. Корпорация использует предупреждающие сообщения о том, что продукт защищен авторским правом и международными соглашениями (голографические изображения, сертификат подлинности). Юридическая защита основана на том, что создана достаточная правовая база для охраны прав авторов интеллектуальной собственности, в том числе и компьютерных программ. Юридическая защита предусматривает гражданскую, уголовную и административную ответственность за нарушение авторских прав создателей программных продуктов. Такая защита является наиболее действенной и, по идее, должна быть основной, так как она гарантирует возмещение ущерба автору, права которого были нарушены. К сожалению, в нашем государстве такая защита оказывается неэффективной.

Третий подход — экономический. Сделать цену продукта настолько низкой, что она может сравниться с пиратской. В большинстве случаев, если цена будет приблизительно одинаковой, покупатель предпочтет лицензионный продукт пиратскому. Тем не менее экономическая конкуренция с пиратством — дело очень тяжелое и подходит далеко не всем производителям ПО.

Возможна ситуация, когда ПО практически невозможно использовать без фирменной документации и технической поддержки, а данные услуги предоставляются только легальным пользователям.

3. Внедрение системы защиты

Для производителей ПО удобнее всего использовать защиту, устанавливаемую на скомпилированные модули. В этом случае можно быстро защитить уже готовое и отлаженное ПО. Установка защиты максимально автоматизирована, сводится в большинстве случаев к заданию нужных параметров и нажатию кнопки «Enter». Такая защита, однако, наименее стойка к атакам. К сожалению, многие компании идут именно по данному пути наименьшего сопротивления, получая в итоге сломанный в кратчайшие сроки пиратами программный продукт. В результате не только ломается продукт, но и дискредитируется компания, которая производит подобную защиту.

Системы с внедрением СЗ в исходный код неудобны для производителей, так как им приходится обучать персонал работе с СЗ, возможно появление новых ошибок в уже отлаженном ПО и ряд других неудобств. Но такие системы наиболее стойки к атакам, так как размывается граница между самим ПО и системой защиты. Компания — поставщик СЗ в обычном случае предоставляет так называемый SDK (Software Developer Kit, комплект разработчика ПО), который позволяет детально ознакомиться с продуктом. Для этого в состав SDK включены полная техническая документация, описание утилит и средств разработки. В комплект разработчика входит все необходимое для начала использования представляемой технологии в собственных программных продуктах — детальные примеры, фрагменты кода, поддержка различных ОС, средств разработки. При использовании SDK защита интегрируется в ПО наиболее эффективным образом (размазанная защита). Традиционно комплект разработчика предлагает многоуровневый API (Application Programming Interface, программный интерфейс приложения) доступа к СЗ. При работе на нижнем уровне существует возможность обращения к любым функциям СЗ, используется наиболее полный и гибкий интерфейс, что увеличивает стойкость защиты, но при этом усложняется процесс разработки. API верхнего уровня предоставляют возможность работать с СЗ на уровне ее абстрактной модели и функциональности. Эти API скрывают от разработчика детали реализации протоколов коммуникации, вопросы разделения времени и различия в реализациях разных моделей и версий СЗ. Разработчик, таким образом, освобождается от большей части рутинной работы и может сконцентрироваться на разработке приложения. Естественно, эффективность написанного кода снижается из-за появления лишнего уровня абстракции, а также ограничений, специфичных для различных API. Также существует возможность использовать одно и то же приложение API различного уровня. Системы с внедрением СЗ в исходный код требуют некоторых дополнительных издержек на обучение специалистов и написание дополнительного кода в тело имеющегося ПО. Стоит отметить, что существенно затрудняется процесс отладки, так как система

уже содержит множество антиотладочных средств. Защитный механизм лучше всего реализовать в виде модулей, готовых к интеграции в программу в любой момент. Рабочая версия вместо защитных функций вызывает процедуры-пустышки, заменяемые в финальной версии реальным кодом.

Лучшим решением является использование СЗ *комбинированного* типа. Сохраняя достоинства первого типа, они делают защиту максимально эффективной.

4. Программные и аппаратные СЗ

Относительно программной защиты необходимо принять во внимание следующее: любая программная защита может быть раскрыта в конечное время. Верность этого утверждения следует из того, что команды программы, выполняющей защиту, достоверно распознаются компьютером и в момент исполнения присутствуют в открытом виде как машинные команды. Следовательно, чтобы понять работу защиты, достаточно восстановить последовательность этих команд, а таких инструкций, очевидно, может быть лишь конечное число [3].

К аппаратным относятся методы, использующие специальное оборудование, например электронные ключи, подключаемые к портам компьютера; нанесение меток на CD и т. д., или физические особенности носителей информации (компакт-дисков, дискет).

Электронный ключ HASP (Hardware Against Software Piracy) или Sentinel — это аппаратная часть системы защиты, представляющая собой плату с памятью и, в некоторых случаях, микропроцессором, помещенную в корпус и предназначенную для установки в один из стандартных портов ПК (COM, LPT, USB) или слот расширения материнской платы (ISA или PCI). Также в качестве такого устройства могут использоваться СМАРТ-карты.

Технология HASP — продукт компании Aladdin Knowledge Systems Ltd, Sentinel — Rainbow Technologies, которая на данный момент является лидером на рынке аппаратных ключей для защиты ПО.

Наличие энергонезависимой памяти дает возможность программировать HASP, размещая внутри модуля различные процедуры, либо хранить дополнительные ключи, а также:

· управлять доступом к различным программным модулям и пакетам программ;

· назначать каждому пользователю защищенных программ уникальный номер;

· сдавать программы в аренду и распространять их демо-версии с ограничением количества запусков;

· хранить в ключе пароли, фрагменты кода программы, значения переменных и другую важную информацию.

Электронные ключи по архитектуре можно подразделить на ключи с памятью (без микропроцессора) и ключи с микропроцессором (и памятью).

Наименее стойкими, в зависимости от типа программной части, являются системы с аппаратной частью первого типа. В таких системах критическая информация (ключ дешифрации, таблица переходов) хранится в памяти электронного ключа. Для дезактивации таких защит в большинстве случаев необходимо наличие у злоумышленника аппаратной части системы защиты. Основная методика взлома такой системы: перехват диалога между программной и аппаратной частями для доступа к критической информации.

Самыми стойкими являются системы с аппаратной частью второго типа. Такие комплексы содержат в аппаратной части не только ключ дешифрации, но и блоки шифрации/дешифрации данных, таким образом при работе защиты в электронный ключ передаются блоки зашифрованной информации, а принимаются оттуда расшифрованные данные. В системах этого типа достаточно сложно перехватить ключ дешифрации, так как все процедуры выполняются аппаратной частью, но остается возможность принудительного сохранения защищенной программы в открытом виде после отработки системы защиты. Кроме того, к ним применимы методы криптоанализа.

У каждого ключа HASP имеется уникальный опознавательный номер, или идентификатор (ID-number), доступный для считывания защищенными программами. Идентификатор присваивается электронному ключу в процессе изготовления, что делает невозможным его замену и гарантирует надежную защиту от повтора. С помощью идентификатора можно шифровать содержимое памяти и

использовать возможность ее дистанционного перепрограммирования. Также идентификаторы позволяют различать пользователей программы. Программа по данному идентификатору может определить, какие привилегии могут быть предоставлены пользователю (разграничение прав).

Система HASP позволяет защищать программное обеспечение двумя различными способами: автоматически (стандартно) и вручную (через специальный API).

СЗ, основанные на привязке ПО к дистрибутиву носителя, также можно отнести к аппаратным методам защиты. На физическом уровне создается дистрибутивный носитель, обладающий уникальными характеристиками, обычно это достигается при помощи нестандартной разметки носителя информации или/и записи на него дополнительной информации — пароля или метки, а на программном — создается модуль, который проводит идентификацию и аутентификацию носителя по его уникальным свойствам (guard module). Стоит отметить, что любой диск (CD/DVD/R/RW) имеет неповторимые свойства, которые теряются при копировании, при этом возможно применение приемов, используемых упаковщиками/шифраторами. Создать защиту, которая безошибочно распознавалась бы любым приводом, но в то же время не копировалась ни одним известным копировщиком, практически невозможно. Наилучший результат дает привязка к геометрии спиральной дорожки (защиты типа CD-COPS, Star-Force). Ее невозможно скопировать, но легко проэмулировать или подобрать диск с похожими характеристиками. Для предотвращения эмуляции разработчикам Star-Force пришлось очень глубоко зарыться в систему, в результате чего они получили большой набор ошибок и множество нападок пользователей. Установка очередного пакета обновлений на Windows требует параллельной установки соответствующего обновления для Star-Force, что очень неудобно. Привязка к дистрибутиву носителя на практике показала свою неэффективность.

Защиты, основанные на привязке ПО к носителю, и защита с HASP-ключами занимают разные сегменты рынка. Если первые СЗ используются главным образом для относительно дешевых мелких и средних продуктов, таких как игры, shareware-программы и т. п., то защита с ключами HASP обходится дороже и рассчитана на рынок серьезных и дорогих систем, обычно для корпоративных пользователей.

5. Шифрование, виртуальные процессоры и другие перспективные методы защиты

Для противодействия обратной инженерии используются упаковщики/шифраторы и сложные логические механизмы. Обратная инженерия в узком понимании — это процесс восстановления спецификации из кода, а в более широком смысле — восстановление всех знаний о системе: анализ функционирования системы и восстановление спецификации. Шифровка — достаточно мощное средство. Она бывает двух видов: статическая и динамическая. При статической шифровке зашифрованный код расшифровывается один-единственный раз на самой ранней стадии инициализации программы, после чего расшифрованному коду передается управление. Это наиболее простой способ реализации и очень ненадежный, так как существует возможность снятия дампа памяти уже после расшифровки. Динамические шифровщики расшифровывают код по мере возникновения в нем необходимости и после возвращения управления тут же зашифровывают его вновь. При этом целесообразно расшифровывать за раз как можно меньшие порции кода, в идеале следует расшифровывать по одной машинной команде или байту данных. Таким образом, при динамической шифровке в каждый момент времени в памяти присутствуют только крохотные куски расшифрованного кода и снятие дампа дает немного пользы. Широкому внедрению динамической шифровки препятствуют следующие проблемы: во-первых, сложность разработки и трудоемкость отладки, во-вторых, очень низкая производительность и, в-третьих, потенциальная возможность снять дамп руками самого расшифровщика, которому последовательно передаются адреса всех зашифрованных регионов, или его тело исправляется так, чтобы он только расшифровывал, но ничего не зашифровывал. Конечно, этому можно противостоять, например, используя перекрывающие шифроблоки, которые могут быть расшифрованы только по очереди, но не все сразу, или многослойную шифровку типа «луковицы», когда один шифровщик плюс немного полезного кода вложен в другой, а тот — в третий и т. д. Шифровщики как бы перемешаны с кодом и «отодрать»

чистый дамп невозможно. Это очень надежно, но очень сложно реализуемо [4]. Бесспорных лидеров среди алгоритмов шифровки нет. Выбор предпочтительного метода защиты не столь однозначен, и статическая шифровка при всех своих недостатках продолжает удерживать прочные позиции.

Вместе с шифрованием часто используется упаковка. Упаковщик добавляет в программу множество антиотладочных приемов, что затрудняет пошаговую трассировку или даже делает ее невозможной. Чтобы успешно противостоять взлому, компоновщик должен быть «размыт» по коду. Если злоумышленнику удается найти точку окончания распаковки, защита легко снимается. То, что программа упакована, можно легко определить по «мусору», который выдает дизассемблирование, или отказу редактора ресурсов работать. Также это можно определить программно, путем анализа степени сжатости кода.

Эмуляция процессоров и операционных систем — создается виртуальный процессор и/или операционная система (необязательно реально существующие) и программа — переводчик из системы команд IBM в систему команд созданного процессора или операционной системы ОС (p-код), после такого перевода ПО может выполняться только при помощи эмулятора, что резко затрудняет исследование алгоритма ПО. Является достаточно эффективным средством защиты, но значительно (в десятки раз) замедляет скорость выполнения программы. Рекомендуется использовать только для защиты критических участков, хотя в этом случае защитные участки можно обойти и защита в итоге получается недостаточно эффективной. Непосредственное дизассемблирование в этом случае становится невозможно и приходится прибегать к декомпиляции, а для этого необходимо проанализировать алгоритм работы интерпретатора, написать и отладить декомпилятор [5]. Разработка и отладка интерпретатора также процесс достаточно трудоемкий. К тому же разработка языка тянет за собой множество вспомогательного инструментария. В итоге получается большой проект, который может быть использован в одной-единственной программе, да и то после выхода нескольких версий ядро интерпретатора желательно слегка изменять, чтобы написанный злоумышленником декомпилятор перестал работать. На практике все критичные ко времени выполнения функции программируются на обычном языке, вся логика и вызовы реализованы на p-коде. Есть вариант использовать Pascal- или Basic- подобные языки, но в этом случае программу легко декомпилировать. С другой стороны, можно использовать примитивные виртуальные машины: Машину Тьюринга, Сети Петри, Стрелку Пирса и т. д., которые реализуют фундаментальные логические операции, в результате чего даже примитивные конструкции распадаются на сотни микрокоманд. Производительность при этом стремится к нулю. Форт-машина в этом случае неплохая альтернатива. Ее главные преимущества: простота реализации, компактность p-кода, довольно высокая производительность, сложность декомпиляции и ни на что непохожесть. Основные концепции Форт сложны для человеческого восприятия. Другая хорошая идея — использовать готовые интерпретаторы малораспространенных языков (Пролог, Хаскель, Оберон), под которые еще не написаны декомпиляторы. Многие из них разработаны в исследовательских институтах и поставляются в исходных текстах, что упрощает модификацию ядра интерпретатора и позволяет легко адаптировать чужой язык к нашим целям. Еще одно возможное решение — эмулятор малоизвестной ЭВМ, под которую есть компилятор с языка высокого уровня. В этом случае злоумышленнику придется разбирать неизвестный машинный язык и писать дизассемблер. А в следующей версии программы можно использовать эмулятор от другой ЭВМ, перекомпилировав весь код без адаптации и каких-либо значительных изменений.

Защита должна быть неявной, без явных проверок и использования оператора разрыва «break», проверки целостности, сигнатуры, счетчики — все это легко ломается. Например, один из вариантов неявной защиты — наложение локальных переменных на код программы [4]. В этом случае программа после взлома продолжает работать, но со сбоями. Плавающая защита не всегда себя проявляет. Защитный участок функционирует при некоторых, трудно определяемых параметрах. Также удачным приемом считается использование вероятностного вызова СЗ. Пусть вызовы защитных функций следуют из разных мест с той или иной вероятностью, определяемой либо оператором типа «random», либо действиями

пользователя. Например, если сумма ASCII-кодов (American Standard Code for Information Interchange, американский стандартный код для обмена информацией) последних шести символов, набранных пользователем, в шестнадцатеричном представлении равна 88h — происходит «внеплановый» вызов защитного кода. Если программа при каждом прогоне ведет себя несколько по-разному, восстановление ее алгоритма чрезвычайно усложняется. Основная ошибка большинства разработчиков защит заключается в том, что они дают злоумышленнику понять, что защита распознала попытку взлома. Значит, возможно вместо немедленного выполнения какого-либо действия создать список отложенных процедур (массив указателей на функции) и проверять его в цикле выборки сообщений или во время простоя системы из отдельного потока. Один поток проверяет регистрационные данные и кладет сюда указатель на функцию, которую нужно выполнить вместе с другими функциями, выполняемыми программой.

6. Борьба с отладчиками, мониторами, дизассемблерами и снятием дампа

Антиотладочные приемы стоит использовать с большой осторожностью или совсем не использовать. Работоспособность данных средств сильно зависит от версии ОС. Недопустимо отказываться от работы в присутствии пассивного отладчика. Необходимо противодействовать лишь активной отладке. Нейтрализации точек останова обычно бывает вполне достаточно. Ставится задача не помешать отладке, а сделать этот процесс максимально неэффективным.

Некоторые способы борьбы с файловыми мониторами и мониторами реестра [4]: самый простой — это найти и закрыть главное окно монитора, другой способ — это хранить флаг регистрации вместе с настройками в одном ключе реестра в двоичном виде, тогда его мониторинг ничего не даст. Злоумышленник видит, из такой ветви читается набор байтов, но вот какой из них за что отвечает, выяснить можно только путем отладки/дизассемблирования защищенной программы.

Методы защиты от статического исследования программ: затруднение дизассемблирования, шифрование, компрессия и т. д. Статические средства защиты оперируют исходным кодом как данными и строят ее алгоритм без исполнения.

Статические средства анализа являются более универсальными в том смысле, что теоретически могут получить алгоритм всей программы, в том числе и тех блоков, которые никогда не получат управления, динамические же могут строить этот алгоритм только на основании конкретной трассы (trace) программы, полученной при определенных входных данных. Поэтому получение полного алгоритма программы в этом случае практически невозможно, так как нужно перебрать все возможные варианты данных, и вообще, при динамическом исследовании можно говорить только о построении некоторой части алгоритма. Отладчику противостоять гораздо проще, чем дизассемблеру.

Снятие дампа работающего приложения в ОП, представляется самым коварным способом преодоления защиты. Все механизмы защиты уже отработали, и мы имеем «чистое» приложение в ОП. Хотя полученный образ exe-файла зачастую оказывается некорректным и работающим нестабильно (вместо реальных идентификаторов строк в секции ресурсов окажутся указатели на память), для дизассемблирования он вполне пригоден, особенно если его использовать в связке с отладчиком, помогающим «подсмотреть» значения некоторых переменных на разных стадиях инициализации.

Средства борьбы со снятием дампа: постоянное перемещение программы в ОП, затирание, шифрование, динамическая распаковка, пометка физической страницы памяти как недоступной, перехват API функций дампинга с использованием специального драйвера. В целом надежно противостоять снятию дампа на прикладном уровне нельзя, а спускаться на уровень драйверов нецелесообразно. Некоторые защиты искажают PE-заголовок, искажают таблицу импорта и используют другие приемы, опасные для стабильной работы системы, затрудняющие дампинг, но не предотвращающие его в принципе. Возможно пойти другим путем – не препятствовать снятию дампа, но сделать полученный образ бесполезным. Для этого достаточно использовать глобальные инициализированные переменные, «перебивая» их новыми значениями. Единственная надежда злоумышленника – отловить момент

завершения распаковки и тут же сбросить дамп, пока защита еще не успела ничего записать в глобальные переменные. Пошаговая трассировка исключается, поскольку ей легко противостоять.

7. Система защиты, реализованная в виде двух взаимодействующих программ

Одним из возможных решений рассматриваемой задачи является использование СЗ, реализованной в виде двух взаимодействующих программ, защищаемой и защищающей, которые взаимодействуют по определенному протоколу. Защищающая программа требуется для увеличения эффективности стандартных механизмов защиты. Они выполняют следующие основные функции: контроль за соблюдением авторских прав (например, парольная защита, система сертификационных ключей), защита от копирования, борьба с отладчиками, дизассемблерами, мониторами, снятием дампа и т. д. Главной задачей защищающей программы является контроль целостности контролируемого приложения. Так, если злоумышленник попытается устранить/обезвредить (модифицировать код) один из стандартных механизмов защиты, контролирующая программа не позволит программному продукту продолжать корректно выполняться. При попытке модификации контролирующей программы система также перестает корректно работать. Таким образом, существенно увеличивается эффективность стандартных механизмов защиты, в обычном же виде они достаточно малоэффективны. Часть защитных механизмов возможно реализовать на защищаемой программе.

У злоумышленника есть три возможных направления атаки: защищаемая программа, информационный канал и защищающая программа. Требуется противодействие по всем трем направлениям:
- защищаемая программа (использовать наиболее эффективные средства защиты: динамическое шифрование, косвенная передача управления, использование вероятностной/неявной защиты, список отложенных процедур);
- канал связи (шифрование данных — использование стойкого криптоалгоритма, осуществление взаимодействия в случайные моменты времени);
- защищающая программа (должна быть реализована таким образом, чтобы достигался максимальный контроль целостности защищаемой программы и при этом по возможности не критично замедлялась ее работа).

Целостность защищаемой программы контролируется одним из следующих образов:
- в защищаемой программе должны быть установлены контрольные точки на критических участках, например секция проверки пароля, где осуществляются диалог и контроль целостности. При этом от защищающей программы должна поступать какая-то информация, необходимая для продолжения своей корректной работы, например адрес следующей команды, ключ для расшифрования участка кода;
- в случайные моменты времени (не так часто) устраивается «отложенный» диалог.

Систему защиты необходимо реализовать таким образом, чтобы обеспечить максимально простой механизм внедрения СЗ, при этом должна обеспечиваться возможность обновлений, а контроль не должен сказаться на производительности.

Заключение

Защиты, гарантирующей стопроцентную надежность, не существует и в принципе быть не может. Аппаратные СЗ более стойкие, но также обладают конечным временем жизни, сюда можно добавить трудность их реализации и высокую стоимость.

В основе любой программной защиты лежат машинной код, шифрование и виртуальные процессоры. Зачастую защита выходит из строя из-за элементарных ошибок. Следует в первую очередь следить за «чистотой» машинного кода (не хранить тестовые ASCII или UNICODE строки в обычном виде), удалять отладочную информацию и все символьные имена из экспорта, не использовать RTTI (Runtime Type Identification — динамическую идентификацию типа данных) в защитных механизмах, не давать формам и обработчикам осмысленные имена. Шифрование (возможно, вместе с компоновкой), как и использование p-кода, является одним из самых эффективных и перспективных методов защиты.

К его недостаткам стоит отнести потерю в производительности. Решение этой проблемы является одной из ключевых при реализации СЗ.

При разработке СЗ требуется учитывать множество факторов: влияние на производительность, системонезависимость, возможность легко обновлять ПО и исправлять ошибки, защита должна быть максимально простой и отлаженной, полученное ПО — удобным в тиражировании и распространении. Система безопасности под конкретный проект должна разрабатываться заранее и работать на прикладном уровне.

СПИСОК ЛИТЕРАТУРЫ

1. BSA и IDC объявляют итоги исследования уровня использования пиратского программного обеспечения на российском корпоративном рынке // http://www.bsa.org/russia/press/newsreleases/russiapressrelease01June2006.cfm.
2. Внутренние ИТ-угрозы в России 2006 // http://www.infowatch.ru/threats?chapter=147151396&id=207732752.
3. *Домашев А. В., Грунтович М. М., Попов В. О., Правиков Д. И., Щербаков А. Ю.* Программирование алгоритмов защиты информации. Учебное пособие. М., 2000. — 288 с.
4. *Касперски Крис*. Техника и философия хакерских атак. М., 2004. Серия: Кодокопатель. — 272 с.
5. *Касперски Крис*. Фундаментальные основы хакерства. Искусство дизассемблирования. М., 2002.