УДК 004.056

***А.А. Бутин*[1]**

[1] *Иркутский государственный университет путей сообщения, г. Иркутск, Российская Федерация*

ТЕХНОЛОГИИ ЗАЩИТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Аннотация. *В статье рассматриваются вопросы защиты программного обеспечения от неавторизованного использования, модификации и копирования, что является важнейшей задачей современных информационно-вычислительных систем. Учитывая широкое распространение технологий виртуализации и облачных вычислений, в рамках которых прикладное программное обеспечение часто исполняется в недоверенной вычислительной среде, особую актуальность приобретают исследования и разработки, направленные на создание новых методов защиты программ от незаконного использования и обратного проектирования алгоритмов. Приведен обзор механизмов зашиты, применяемых к программному обеспечению с целью усложнения процесса его модификации.*

Ключевые слова: *технологии взлома и защиты программного обеспечения, исследование исполняемого кода, статический анализ, динамический анализ.*

***A.A. Butin*[1]**

[1] *Irkutsk State University of railway engineering, Irkutsk, Russian Federation*

PROTECTION TECHNOLOGIES SOFTWARE

Abstract. *The article discusses the issues of software protection from unauthorized use, modification and copying, which is the most important task of modern information and computing systems. Given the widespread use of virtualization and cloud computing, in which application software is often executed in an untrusted computing environment, research and development aimed at creating new methods to protect programs from illegal use and reverse engineering of the algorithms used in them are of particular relevance. A review of the protection mechanisms applied to the software to complicate the process of its modification is given.*

Keywords: *technology hacking and software protection, the study of executable code, static analysis, dynamic analysis.*

Написанный программистами код может быть как открытым (исходный код, написанный на языках низкого или высокого уровней, находиться в свободном доступе, а значит, любой желающий может изучить устройство программы), так и закрытым (разработчики таких программ скрывают их структуру, исходных код является конфиденциальной информацией и защищён).

Как правило, программы с закрытым исходным кодом распространяются на платной основе. Разработчики тратят ресурсы, следовательно, ожидают покрытия расходов на разработку и дохода с производимой продукции. Но без использования средств защиты исполняемого кода можно лишиться дохода полностью, т. к. единожды продав лицензионную копию программы, купивший начнёт делиться ею, а особо предприимчивые продавать по цене дешевле, чем у разработчика, тем самым уменьшая его доход.

На практике ситуация ещё сложнее: исполняемый код программы может быть модифицирован злоумышленниками не только с целью её бесплатного распространения, но и по просьбе конкурентов в программу может быть внедрено вредоносное программное обеспечение (далее ПО), что заставит ее работать неправильно, и тем самым подорвет репутацию компании-разработчика. Исполняемый код может быть изучен злоумышленником, возможно, по просьбе конкурентов. Результат изучения: общие сведения о программе, используемые в программе инновационные алгоритмы и т. д. вплоть до восстановления исходного кода программы.

Изучение методов защиты исполняемого кода является перспективным направлением, т. к. количество программных продуктов неуклонно возрастает, а вместе с ними и количество случаев пиратства, в результате которого разработчики несут колоссальные убытки. По-

этому вопрос безопасности собственности разработчиков ПО от его копирования, модификации и изучения становится одним из важнейших [1] – [10].

Анализ современных технологий взлома и защиты ПО

Необходимость внедрения систем защиты исполняемого кода возникает ввиду определённых ограничений, накладываемых разработчиками на распространение своей продукции. Поводом для введения таких мер могут служить уникальность отдельных частей кода или программы в целом, которые приносят разработчикам прибыль от продажи данной продукции, сокрытие программных закладок или люков, которые разработчик не хотел бы оглашать или документировать и в случае изучения кода компания может понести материальные убытки, а также серьёзно подорвать свою репутацию.

Классификация воздействий на исполняемый код включает два направления:

- анализ;
- модификация.

Оба этих направления – это целый класс методов и утилит, используемых не только злоумышленниками после коммерческого релиза приложения, но и разработчиком на этапах программирования, сборки и отладки.

При разработке систем защиты исполняемого кода необходимо учитывать особенности реализации таких воздействий и поэтому их следует рассмотреть более детально.

Исследование исполняемого кода

Следует отметить, что термины: исследование, изучение и анализ исполняемого кода являются синонимами, и рассматриваются в контексте статьи именно так.

Анализ ПО – методики исследования исполняемого кода приложения; полученная информация в ходе их применения используется в различных целях: поиск уязвимых мест в коде (бесконечные циклы, исключения и т.п.), ошибок нарушения доступа к памяти, тестирование на предмет ошибок синхронизации многопоточных приложений, отчёты о производительности и т.д.

Согласно определению, область применения анализа ПО широка. Им могут воспользоваться как разработчики, так и злоумышленники, следовательно, анализ ПО нельзя отнести к однозначно негативным воздействием на код.

Выделяют два типа анализа ПО: статический и динамический анализ.

Статический анализ

Статический анализ кода – анализ ПО, производимый без выполнения исследуемой программы, т.е. без её запуска.

Этому типу анализа могут быть подвергнуты как исходный код программы, так и объектный код или исполняемый машинный код.

В зависимости от реализации статический анализ может быть проведён в ручном или автоматизированном режиме (в виде утилит или надстроек для систем IDE). Утилиты и надстройки применяются разработчиками для тестирования, выявления и устранения потенциальных ошибок ПО.

Злоумышленниками могут использоваться обе реализации для большего сбора сведений о программе, но, как правило, используют трансляторы, преобразующие исполняемый код в текст программы на языке ассемблера. Произведя дизассемблирование незащищённого приложения, становится довольно легко изучить принцип его работы и функционирования.

Динамический анализ

Динамический анализ кода – анализ ПО, производимый в результате выполнения исследуемой программы на реальном или виртуальном процессоре.

В отличие от статического анализа, здесь все манипуляции над исполняемым кодом происходят, когда программа загружена в оперативное запоминающее устройство (ОЗУ).

Злоумышленник, используя динамический анализ, способен исследовать исполняемый код пошагово, трассируя программу, а также используя точки останова и отслеживая изменение состояний регистров процессора, стека и других сегментов программы.

Доступные средства: отладчики, виртуальные машины.

Модификация исполняемого кода

Бывают ситуации, когда разработчик выпускает патч (от англ. patch – заплата) для приложения, позволяющий изменить несколько известных ему значений в исполняемом файле. К примеру, исполняемый файл программы размером 50МБ и патч – размером 11КБ. Из соображений экономии интернет-трафика пользователей или по каким-либо другим причинам можно выпустить патч размером 11КБ, передать пользователям по сети и изменить значения исполняемого файла в 50МБ.

Возможно, такие ситуации бывали и ранее, но в современные дни передача 50МБ данных не является проблемой, вдобавок данные можно подвергнуть сжатию.

Приняв это во внимание, можно отметить, что модификация исполняемого кода различными патчами, активаторами и crack-программами – прерогатива злоумышленников, и если в лицензионном соглашении прописан пункт о запрете модификации программы, то это приводит к его нарушению.

Модификация исполняемого кода может быть произведена в файле или даже во время выполнения, т.е. динамически. Удачным попыткам модификации исполняемого кода во многом способствует слабая защита или полное отсутствие защиты от анализа ПО. Действительно, если злоумышленнику удастся изучить критические части программы, в которых содержатся сведения о проверке лицензионных ключей или каких-либо других атрибутов безопасности, то вся остальная защита становится преодолённой.

Переходя к аспектам разработки безопасности исполняемого кода, необходимо определить два немаловажных аспекта: от чего и чем следует выполнять защиту.

Безопасность исполняемого кода направлена на противодействие:

- анализу ПО;
- модификации исполняемого кода.

По типу реализации защиты следует выделить:

- статический режим;
- динамическая режим.

В статическом режиме обеспечивается безопасность исполняемого кода, без его запуска и выполнения. Этот тип защиты реализуется разработчиком непосредственно в процессе или после сборки приложения. Как правило, такая защита является разделённой: часть защиты производится над готовым кодом, другая часть защиты является динамической.

Динамический режим подразумевает интерактивную защиту кода во время его выполнения и нахождения в ОЗУ компьютера. Реализуется разработчиком при программировании приложения и является встроенным в него кодом или вынесен в отдельное приложение.

Средства защиты программ от анализа и модификации принципиально отличаются от обычных средств защиты, которые применяются в защитах от несанкционированного использования и копирования ПО.

Специфика защиты исполняемого кода заключается в сокрытии логики алгоритмов и частей программы, содержащих уникальный код. Добиться этого весьма непросто, т.к. если приложение достаточно сложное, то и логика его защиты будет сложной.

Обычные действия злоумышленника сводятся к дизассемблированию машинного кода приложения и попыткам восстановить алгоритм программы. Возможны методы трассировки кода, что приводит к более быстрому исследованию программы, а как следствие, к взлому встроенной защиты от несанкционированного использования и копирования.

Методы защиты исполняемого кода

Широкое применение на практике нашли следующие методы защиты кода:

- сжатие/криптографические методы;
- обфускация (запутывание);
- виртуализация процессора;
- использование упаковщиков/изменение заголовков;
- полиморфизм и мутации;
- доступ к драйверам и оборудованию;
- нестандартные методы.

Каждый метод уникален по-своему, им присущи определённые достоинства и недостатки, которые нуждаются в более детальном изучении. Дальнейшее рассмотрение методов защиты будет происходить с учётом их специфики применимости в статическом и динамическом режимах.

Сжатие/Криптографические методы

Криптографические методы направлены на видоизменение открытых данных путём математических действий по заданным алгоритмам. Такими методами являются:

- шифрование;
- стеганография;
- хеширование;
- сжатие;
- кодирование;
- распределение ключевой информации и др.

В зависимости от разрабатываемой системы защиты исполняемого кода, могут применяться любые из возможных криптографических методов, т.к. к построению таких систем следует подойти творчески. Действительно, как было отмечено, специфика защиты исполняемого кода от изучения кардинально отличается от любой другой защиты (несанкционированное использование и т.п.), хотя порой используются одни и те же средства защиты.

Шифрование является универсальным средством защиты не только в данном контексте, оно применяется повсеместно: протоколы безопасного обмена данными, конфиденциальная информация в базах данных (БД), цифровых подписях и др.

Для обеспечения защиты исполняемого кода шифрование применяется для противодействия дизассемблированию и модификации. Очевидно, что реинжиниринг зашифрованного кода не представляется возможным и, как следствие, модифицировать определённые значения в исполняемом файле так же затруднительно.

Шифрование перестаёт быть эффективным методом, когда применяется трассировка программы под отладчиком. Тогда, чтобы программа работала корректно, ей необходимо будет расшифровать код. Возможно, у нарушителя уйдёт немало времени на трассировку, но отследив момент дешифрования, защита будет считаться преодолённой.

Принимая это во внимание, следует учитывать некоторые особенности в реализации шифрования исполняемого кода:

- внедряемый алгоритм шифрования может быть как симметричным, так и ассиметричным. Бытуют различные мнения, какая из систем лучше подходит для защиты исполняемого кода. При этом использование двухключевых криптосистем в случае успешной попытки анализа логики работы защитного механизма не позволит модифицировать код, поскольку изменения необходимо внедрить в зашифрованном виде в код, а закрытый ключ недоступен нарушителю.

  В таком случае злоумышленник попытается отыскать открытый ключ, используемый при дешифровании кода; сгенерирует новую пару ключей и заменит код и открытый ключ в приложении. Выполнив эти действия, система защиты основанная на криптографии, будет преодолена. Стоит принять во внимание производительность и сложность реализации одно- и двухключевых алгоритмов, т.к. они не равнозначны;

- внедряемый алгоритм шифрования можно комбинировать с методом обфускации (запутывания) для повышения затраченного времени при использовании нарушителем трассировки.

Усилить защиту способны:

- многопроходная расшифровка кода; возможна реализация с двумя и более ключами для дешифрования или использование мастер-ключа;
- динамическое шифрование, являясь частью динамической защиты, способно выполнять шифрование программы в ОЗУ непрерывно, а также модифицировать исполняемый файл.

Недостатки метода:

- существует проблема хранения ключей;
- способ малоэффективен при трассировке приложения.

В защите исполняемого кода можно применять методы стеганографии. К примеру, для сокрытия ключевой (или иной) информации, необходимой для дешифрования кода, замаскированный под исполняемый код. Очевидно, если прописывать ключи в ресурсах или сегменте данных программы, где расположены константы и другая информация, то найти ключ довольно легко. Но отыскать сокрытый ключ в исполняемом файле уже непросто.

Можно также скрывать части защищаемого кода, но следует помнить, что контейнер должен быть во много раз больше скрываемых данных.

Недостаток метода состоит в том, что он позволяет скрывать лишь небольшие фрагменты информации.

Сжатие участков исполняемого кода или целиком всего кода - довольно распространённый метод защиты. Функция сжатия преобразовывает исходные данные к иному виду, дизассемблировать которые не имеет смысла.

Хорошую степень защиты приносит комбинирование методов сжатия с различными упаковщиками при условии, что упаковщик не является одним из популярных (к примеру, UPX или ASPack) и хорошо известных взломщикам, а написан разработчиком самостоятельно. Такая защита затруднит анализ и уменьшит размер приложения.

Недостаток метода - популярные функции сжатия хорошо известны многим взломщикам;

Основная задача использования хэш-функций в системе защиты кода – защита от модификации. Хеширование работает в динамическом режиме. Принцип работы состоит в следующем. Путём вычисления контрольных сумм (КС) и выполнения сравнения их с эталонами, прошитыми в программе, обеспечивается контроль целостности (КЦ) защищаемых участков кода или его целиком. КЦ производится не только при перезапуске приложения, но и при непрерывном контроле защищённых участков памяти в ОЗУ.

Стоит отметить, что обойти защиту с использованием простых функций (CRC или FNV и др.) достаточно просто, поэтому стоит использовать MD!, SHA-!, SIMD или написать собственную хэш-функцию.

Недостатки метода:

- проблема хранения КС;
- популярные хэш-функции хорошо известны многим взломщикам.

В заключении стоит отметить, что все криптографические методы не защищают приложение от динамического анализа, несмотря на свою эффективность в борьбе с дизассемблированием. Для этого систему защиты приложения необходимо дополнить методами защиты от отладки.

Обфускация (запутывание)

Запутывание кода состоит в видоизменении исполняемого кода программы с сохранением её функциональности, но затрудняющим анализ и модификацию.

На уровне запутывания исполняемого кода применяются технологии:

- перемешивания кода;

- передача лишних параметров в процедуры;
- внедрение «балластного» кода;
- выполнение ложных процедур;
- выполнение взаимоисключающих действий и пр.

Метод в основном не эффективен, поэтому его следует комбинировать с другими средствами защиты: с виртуальными машинами, для создания мощного байт-кода с обфускацией или полиморфными методами.

Недостатки метода:

- современные дизассемблеры упрощают и оптимизируют результат;
- происходит потеря производительности при сильной обфускации.

Виртуализация процессора

В контексте защиты исполняемого кода виртуальная машина (ВМ) – это ПО, эмулирующее вычислительную среду аппаратного обеспечения системы, но с переопределённым набором команд (инструкций).

Виртуальные машины базируются на операционных системах (ОС) или сами являются ОС. Целью их является абстрагирование от аппаратной или программной платформы, обеспечение независимости. Система команд ВМ может во многом отличаться от команд аппаратной части. Регламентированное в ВМ выполнение последовательности команд должно соответствовать тем же действиям, которые выполняются машинными инструкциями. В этом случае такие последовательности команд называются виртуальными программами, а исполняемый в ВМ код называется байт-кодом.

Таким образом, исходный код программы интерпретируется в байт-код ВМ с последующим его выполнением в программной среде ВМ, минуя трансляцию приложения в машинный код. Для проведения реинжиниринга виртуальных программ злоумышленник должен изучить архитектуру ВМ, а затем написать дизассемблер, распознающий байт-код данной ВМ.

Усложнить задачу взлома способны комбинации с полиморфизмом или программные генераторы, создающие при запуске программы новую ВМ со своими наборами инструкциями, регистрами и т.д. Тогда злоумышленнику придётся начинать свою работу сначала.

Важным условием в разработке ВМ является варьирование ее параметров таким образом, чтобы они в значительной мере отличались друг от друга.

Наряду с полиморфизмом, виртуализация процессора является одним из лучших методов защиты исполняемого кода от реинжиниринга.

Недостаток метода состоит в длительности и трудоёмкости разработки.

Упаковщики/Изменение заголовков

Упаковка исполняемых файлов – это процесс формирования отличного от исходного исполняемого файла (контейнера), который содержит сжатый исходный файл программы и добавленный к нему код, необходимый для распаковки и выполнения содержимого контейнера.

Упаковка исполняемых файлов может быть совмещена с шифрованием исходного файла программы для предотвращения дизассемблирования и модификации при условии, если использовался нестандартный упаковщик. Популярные платные и бесплатные упаковщики известны злоумышленникам (такие как UPX, ASPack, exPressor и др.), и не являются полноценной защитой.

Изменение заголовков исполняемых файлов программы – это ещё один из способов для затруднения дизассемблирования. Для примера, исполняемые файлы .exe в ОС Windows содержат следующие записи (язык pascal):

```
HeadExeType = record
Sign Word;      {ПризнакEXE-файла}
…
```

```
ReloCntWord;  {Количество элементов в табл. перемещения}
HdrSizeWord;  {Длина заголовка в параграфах}
…
ReloSSWord;   {Начальное значение сегмента стека SS}
ExeSPWord;    {Начальное значение указателя стека SP}
…
ExelPWord;    {Смещение точки запуска программы}
ReloCSWord;   {Начальное значение сегмента кода CS}
TablOffWord;  {Смещение первого элемента табл. перемещения}
  …
end;
```

Модифицируя эти записи, можно управлять запуском приложения в системе, в значительной мере затруднив его реинжиниринг.

Нестандартная сегментация программы

Современные программы состоят как минимум из трёх сегментов: сегмент данных и сегмент кода создаются непосредственно при трансляции проекта, сегмент стека создаётся при запуске приложения в ОС. Добавляя в приложение дополнительные сегменты, есть возможность использовать сегмент данных для расположения в нём исполняемого кода. Таким образом, дизассемблеру не удастся распознать такую сегментацию программы.

Недостатки метода:

- упаковка программ стандартными упаковщиками неэффективна;
- трудоёмкость написания нестандартных упаковщиков и сегментов.

Полиморфизм и мутации

Самогенерирующийся и самомодифицирующийся код – одно из самых мощнейших и лучших решений. Защита, базирующаяся на этом методе, позволяет предотвратить как модификацию, так и анализ ПО. Достигается это за счёт видоизменения программного кода с сохранением его изначальной функциональности. Изменения могут происходить несколько раз, начиная с запуска в ОС, в процессе работы или завершения приложения. Идеальным вариантом разработки полиморфных программ является уникальность каждого нового запуска, т.е. предыдущий результат запуска кода отличается от текущего. Распространёнными приёмами являются:

- перемещение участков кода; основывается на случайном изменении адресов процедур при каждом запуске и выгрузке приложения;
- смена сегментов; размещение кода в нестандартных сегментах;
- генерация кода; может быть запись «мусорного» кода, не имеющего никакого значения;
- генерация кода с обратной связью; видоизменение стандартного кода в некоторой зависимости от его предыдущего состояния, это может быть подмена инструкций процессора, т.е. синонимы к текущим операциям;
- изменение порядка следований инструкций и др.

Недостатки метода заключаются в длительности и трудоёмкости разработки.

Мутации – один из вариантов видоизменения исполняемого кода без изменения его функциональности. Суть метода заключается в составлении таблицы синонимов для операндов, участвующих в хранении данных и пр., и их замене при запуске программы по схеме или случайным образом.

Драйверы и доступ к оборудованию

Использование методов этой категории направлено в целом на защиту программ от динамического анализа, т.е. отладки.

Общая концепция работы приложений под отладчиками заключается в следующем: существуют два основных отладочных механизма:

- точки останова;
- трассировка программы.

Точки останова – это специальный однобайтовый код (CCh), который вносится в исполняемый код. Во время выполнения программы, как только достигается точка останова, генерируется исключительная ситуация и сразу вызывается подпрограмма обработки прерывания INT 3h.

Процессор приостанавливает выполнение программы, сохраняет в стеке регистры флагов, регистр сегмента кода CS и регистр IP – указатель на следующую команду после прерывания.

Для анализа программы, используя точки останова, следует их внести в исполняемый код, а значит, модифицировать приложение. Для защиты от этого распространены следующие методы:

- контроль целостности кода и в случае ее нарушения прекращение дальнейшей работы приложения;
- многопроходное шифрование, видоизменяющее контрольные точки и делающее отладку трудоёмкой из-за возникающих ошибок и коллизий;
- написание функций поиска позиций контрольных точек останова с последующим их удалением;
- контроль времени выполнения защищаемых участков кода и сравнение с эталонным временем и др.

Наиболее эффективными являются методы:

- перехвата прерывания INT 3h. Метод основывается на модификации обработчика прерывания, заполняя его мусором или подменяя на подпрограмму, выполняющую какое-либо действие, направленное на прекращение выполнения приложения (к примеру, шифрование кода),
- манипуляции со стеком программы. Как было отмечено ранее, точки останова генерируют исключительную ситуацию и процессор сохраняет текущее состояние программы в стек. При выполнении защищённого участка кода значение регистра указателя стека SP обращают в ноль. В таком случае стек считается полным и ОС завершает работу приложения. Ещё одно использование стека для защиты: хранить важные для работы программы данные в стеке. Отладчик перезапишет находящиеся в стеке данные и программу ожидает аварийное завершение.

Трассировка программы происходит путём установки флага TF – флага трассировки, что приводит к генерированию процессором исключительных ситуаций INT 1h после каждой выполненной команды. Прерывание INT 1h называется трассировочным. Аналогично точкам останова, в стек программы сохраняются отладочные данные, а значит, применимы методы защиты от точек останова.

Основным способом защиты является контроль и подмена прерываний. Выглядит он следующим образом: содержимое вектора прерывания, который нужно вызвать, копируется в новый вектор, а в исходный вектор помещается случайное значение. Для обработки вызывается новый вектор прерывания вместо известного взломщику исходного. При этом отслеживание известных исходных векторов не даст нарушителю ничего, кроме случайных значений, а постоянное применение команд разрешения и запрещения прерываний незаметно при нормальной работе программы, но очень сильно затрудняет отладку.

Недостатки метода:

- современные ОС и отладчики позволяют устанавливать аппаратные точки останова и эти методы неэффективны против них;
- некоторые из способов хорошо изучены и опытные нарушители довольно легко преодолевают такую защиту;

- эмуляция аппаратного обеспечения позволяет создавать мощнейшие отладчики, которые могут сохранять стек до действия и после, полностью или частично интерпретировать команды процессора с отслеживанием всех производимых манипуляций, что способствует более эффективному анализу и сводит подобные методы защиты практически на нет.

Нестандартные методы

Использование электронной подписи (ЭП).

Существует не так много вариантов применения ЭП при защите исполняемого кода. Одним из них является использование ЭП для проверки целостности кода, как усиленный вариант хэш-функций.

В таком случае проверка подписи осуществляется на открытом ключе, а для подмены подписи нарушителем необходимо вычислить закрытый ключ. Стоит отметить, есть вероятность, что когда нарушитель отыщет в коде сигнатуры и изучит алгоритм защиты, то ему не придётся подбирать закрытый ключ. Он сгенерирует новую пару ключей, подпишет защищаемые участки и заменит старые сигнатуры на новые. Исправить ситуацию может контроль целостности файла или участка памяти, где хранятся сигнатуры.

Недостатки метода:

- ресурсоёмкость асимметричной криптографии;
- сложность разработки.

Разработка многопоточных приложений.

Отлаживать многопоточное приложение является непростой задачей. В отдельных нитях программы можно реализовывать частичные средства защиты, что значительно усложнит анализ программы. Для примера, в отдельном потоке организуется таймер, который по прошествии заданного периода времени проверяет метки времени и счётчик команд процессора. Если за минуту были выполнены несколько десятков команд, то приложение можно модифицировать, шифровать, путать код или просто завершить.

Ещё одним из нестандартных решений может быть выпуск обновлений для приложений. К примеру, можно внедрить в приложение некритическую для его работоспособности ошибку, а затем выпустить обновление, которое исправит ошибку и вернёт исполняемый файл к исходному виду в случае его взлома.

Недостаток метода - возможно, пользователь не будет обновлять приложение.

Эффективность реализации методов

Как было отмечено, не все из представленных методов одинаково эффективны. Простые методы защиты, такие, как обфускация, не являются достаточно эффективными сами по себе, но это не означает, что такой вариант не годится для защиты. К примеру, в комбинации с виртуальной машиной или реализация динамической обфускации – одни из сильнейших методов статической и динамической защиты. Существуют методы, которые крайне эффективны, однако их реализация может быть неоправданной, т.к. затраты на разработку защиты могут превысить стоимость защищаемого приложения.

При проектировании систем защиты исполняемого кода, как и при проектировании любой другой системы защиты, необходимо придерживаться принципа разумной достаточности, а также учитывать производительность приложения, т.е. уровень ее снижения от использования системы защиты.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Казарин О.В. Безопасность программного обеспечения компьютерных систем. Монография. – М.: МГУЛ, 2003. – 212 с.

2. Бутин А.А., Носков С.И., Торопов В.Д. Спецификация статистической модели деятельности предприятия малого бизнеса//Информационные технологии и проблемы математического моделирования сложных систем. – Иркутск:ИрГУПС, 2006. –Вып. 4. – С. 59-62.

3. Иванченко А.А. Бутин А.А. Использование DLP-систем при расследовании инцидентов информационной безопасности// Информационные технологии и проблемы математического моделирования сложных систем. – Иркутск:ИрГУПС, 2017. –Вып. 18. – С. 15-22.

4. Макаренко С. И., Чукляев И. И. Терминологический базис в области информационного противоборства //Вопросы кибербезопасности, 2014. – № 1. – С. 13-21.

5. Чукляев И. И., Морозов А. В., Болотин И. Б. Теоретические основы построения адаптивных систем комплексной защиты информационных ресурсов распределенных информационно-вычислительных систем. Монография // Смоленск: ВА ВПВО ВС РФ, 2011. – 227 с.

6. Гецци, Карло Основы инженерии программного обеспечения // Карло Гецци , Мехди Джазайери , Дино Мандриоли. – М.: БХВ-Петербург, 2017. – 832 с.

7. Гроувер, Д. Защита программного обеспечения // Д. Гроувер, Р. Сатер, и др.. - М.: Мир, 2017. - 283 с.

8. Гэртнер, Маркус ATDD. Разработка программного обеспечения через приемочные тесты // Маркус Гэртнер. - Москва: Мир, 2014. - 232 с.

9. Джим, Маккарти Правила разработки программного обеспечения (+ CD-ROM) / Маккарти Джим. - М.: Русская Редакция, 2017. - 825 с.

10. Дюваль Непрерывная интеграция. Улучшение качества программного обеспечения и снижение риска // Дюваль, М. Поль. - М.: Вильямс, 2017. - 240 с.

REFERENCES

1. Kazarin O. Computer software security. Monograph. - M .: MGUL, 2003. - 212 p.

2. Butin A.A., Noskov S.I., Toropov V.D. Specification of a statistical model of a small business enterprise // Information technologies and problems of mathematical modeling of complex systems. - Irkutsk: IrGUPS, 2006. –Rel. 4. - p. 59-62.

3. Ivanchenko A.A. Butin A.A. Use of DLP-systems in the investigation of information security incidents // Information technologies and problems of mathematical modeling of complex systems. - Irkutsk: IrGUPS, 2017. –Rel. 18. - p. 15-22.

4. Makarenko S.I., Chuklyaev I.I. Terminological basis in the field of information confrontation // Cybersecurity Issues, 2014. - № 1. - P. 13-21.

5. Chuklyaev I. I., Morozov A. V., Bolotin I. B. Theoretical foundations of the construction of adaptive systems for the integrated protection of information resources of distributed information and computing systems. Monograph // Smolensk: BA VPVO RF Armed Forces, 2011. - 227 p.

6. Gezzi, Carlo Basics of software engineering // Carlo Gezzi, Mehdi Jazayeri, Dino Mandrioli. - M .: BHV-Petersburg, 2017. - 832 c.

7. Grover, D. Software Protection // D. Grover, R. Suter, et al. - M .: Mir, 2017. - 283 p.

8. Gärtner, Markus ATDD. Software development through acceptance tests // Markus Gärtner. - Moscow: World, 2014. - 232 c.

9. Jim, McCarthy Software Development Rules (+ CD-ROM) / McCarthy Jim. - M .: Russian Edition, 2017. - 825 c.

10. Duval Continuous Integration. Improving the quality of software and reducing risk // Duval, M. Pol. - M .: Williams, 2017. - 240 p.

Информация об авторе

Бутин Александр Алексеевич – к.ф.-м.н, доцент кафедры «Информационные системы и защита информации», Иркутский государственный университет путей сообщения, г.Иркутск, e-mail: baa@irgups.ru

Author

Butin Alexander Alekseevich - Candidate of Physical and Mathematical Sciences, associate professor of the department "Information Systems and Information Security", Irkutsk State Transport University, Irkutsk, e-mail: baa@irgups.ru

Для цитирования

Бутин А.А. Технологии защиты программного обеспечения// «Информационные технологии и математическое моделирование в управлении сложными системами»: электрон. науч. журн. – 2019. – №2. – С. 53-63 – Режим доступа: http://ismm-irgups.ru/toma/23-2019, свободный. – Загл. с экрана. – Яз. рус., англ. (дата обращения: 19.06.2019)

For citation

Butin A.A. *Technologii zashiti programmnogo obespechenia* [Protection technologies software] // *Informacionnye tehnologii i matematicheskoe modelirovanie v upravlenii slozhnymi sistemami: ehlektronnyj nauchnyj zhurnal* [Information technology and mathematical modeling in the management of complex systems: electronic scientific journal], 2019. No. 2. P. 53-63 – Access mode: http://ismm-irgups.ru/toma/23-2019, free. – Title from the screen. – Language Russian, English. [Accessed 19/06/19]