МЕТОДИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ СИСТЕМ ЗАЩИТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Бутин А.А. Email: Butin652@scientifictext.ru

Бутин Александр Алексеевич – кандидат физико-математических наук, доцент,
направление: информационная безопасность,
кафедра информационных систем и защиты информации,
Иркутский государственный университет путей сообщения, г. Иркутск

Аннотация: *в статье рассматриваются вопросы защиты программного обеспечения от неавторизованного использования, модификации и копирования, что является важнейшей задачей современных информационно-вычислительных систем. Компьютерное пиратство и незаконное использование программ наносят большой вред экономике страны, особенно ее высокотехнологичному сектору. Согласно оценкам специалистов совокупные потери от нелицензионного использования программ постоянно растут, что свидетельствует о необходимости дальнейшего повышения эффективности методов их защиты. 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 применяемых в них 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.*

METHODICAL ASPECTS OF DEVELOPMENT OF PROTECTION SYSTEMS SOFTWARE

Butin A.A.

Butin Alexander Alekseevich – Candidate of Physical and Mathematical Sciences, Associate Professor,
DIRECTION: INFORMATION SECURITY,
DEPARTMENT OF INFORMATION SYSTEMS AND INFORMATION SECURITY,
IRKUTSK STATE UNIVERSITY OF COMMUNICATIONS, IRKUTSK

Abstract: *the article discusses the issues of protecting software from unauthorized use, modification and copying, which is the most important task of modern information and computing systems. Computer piracy and illegal use of programs do great harm to the country's economy, especially its high-tech sector. According to expert estimates, the cumulative losses from the unlicensed use of programs are constantly increasing, which indicates the need to further improve the effectiveness of their protection methods. Given the widespread use of virtualization and cloud computing technologies, within which application software is often executed in an untrusted computing environment, research and development aimed at creating new methods for protecting programs from illegal use and reverse engineering of the algorithms used in them are of particular relevance. An overview of the protection mechanisms applied to the software in order to complicate the process of its modification is given.*

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

УДК 004.056

Образцом для построения современных цифровых компьютеров стала работа английского математика Чарльза Бэббиджа по созданию аналитической машины, идея которой появилась ещё в XIX веке. В том же веке, английский математик, графиня Ада Лавлейс создала первую в мире программу для другой машины Бэббиджа, которая носит название разностной машины. Графиня Лавлейс считается первым программистом в истории.

Современные задачи, в решении которых нуждается мир, стали намного сложнее, чем в XX веке. Некоторые из них не представляется возможным решить, даже если объединить вычислительные мощности устройств на всей планете. Стоит заметить, что некоторые программы для отыскания решения существуют, однако недостаточная мощность вычислительных устройств не позволяют реализовать подобные задачи за приемлемое время.

Разрабатываемые сегодня программы упрощают работу не только учёным и научным сотрудникам, но и организациям, деятельность которых с наукой не связана, а торгово-рыночные отношения наделяют программные продукты стоимостью. Чем выше качество, возможности программы, богаче функционал, тем больше её стоимость.

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

Как правило, программы с закрытым исходным кодом распространяются на платной основе. Разработчики тратят ресурсы, следовательно, ждут покрытия расходов на разработку и доход с производимой продукции. Но без использования средств защиты исполняемого кода, можно лишиться дохода совсем, 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 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 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. Поэтому 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 нарушение 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 устранение потенциальных ошибок ПО.

Злоумышленниками могут использовать 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 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 данных 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;
- динамический 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. Является встроенным в приложение 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 следствию 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 защита исполняемого кода от изучения кардинально отличается от любой другой защиты (несанкционированное использование и т.п.), порой используя 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, ведь 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 можно 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 выполняя сравнение их с эталонами, прошитыми в программе, обеспечивается контроль целостности (КЦ) защищаемых участков кода или его целиком. КЦ производится не только 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.

Список литературы / References

1. *Казарин О.В*. Безопасность программного обеспечения компьютерных систем. Монография. М.: МГУЛ, 2003. 212 с.
2. *Аветисян А.И., Белеванцев А.А., Бородин А.Е., Несов В.В.* Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ // Труды ИСП РАН. Том 21, 2011. С. 23-38.
3. *Макаренко С.И., Чукляев И.И.* Терминологический базис в области информации информационного противоборства // Вопросы кибербезопасности, 2014. № 1. С. 13-21.
4. *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.
5. Provos Niels, Holz Thorsten. Virtual Honeypots: From Botnet Tracking to Intrusion Detection//AddisonWesleyProfessional, 2007. 480 p.
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. *Маккарти Джим*. 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.
11. *Фаронов В.В.* Turbo Pascal 7.0 Практика программирования. Учебное пособие. М.: Нолидж, 2001. 416 с.