**Лекция 13. Ещё раз про что-то или «Невезучая».**

**Читаем внимательно, сначала повторяем то, что было раньше, затем разбираемся в проекте программного средства, предназначенного для зашиты ПО от несанкционированного использования. Возможно (хотя вряд ли), что кто-то из Вас НЕ будет человеком-пауком, НЕ живущим в серпентарии, НЕ домохозяйкой, «лабающей» на Jave, НЕ секретаркой, НЕ кодировщиком, НЕ «ЗАДНИМ КОНЦОМ», НЕ «ПЕРЕДНИМ КОНЦОМ», НЕ «ПОЛНЫМ СТАКАНОМ», а БУДЕТ простым разработчиком программного обеспечения, поэтому в конце этой лекции приведен список литературы, которым можно было бы воспользоваться для повышения квалификации в этой области – разработки ПО. Ни на какие вопросы отвечать не надо и задавать вопросы не надо!! Просто читайте.**

**Ключевые слова**: защита ПО, пароль, криптографическое закрытие, электронный ключ, обфускация, ассемблер, протектор, отладчик, дизассемблер, загрузчик.

**Результаты:** структурирована информация о программных методах защиты программного обеспечения от несанкционированного использования, составлен проект программного средства, предназначенного для защиты имеющегося программного обеспечения, состоящего из главного исполняемого модуля и динамически подключаемых библиотек

# ВВЕДЕНИЕ

Проблема несанкционированного копирования и использования программного обеспечения возникла еще в 70-ых годах прошлого столетия и по сей день не теряет своей актуальности. Причиной этому послужило стремительное развитие рынка персональных ЭВМ и увеличение числа домашних компьютеров. Нарушение авторских прав в области программного обеспечения называют компьютерным пиратством. Для многих людей вопрос, связанный с компьютерным пиратством не является однозначным. Часто можно услышать такие высказывания: "я просто скачал игру/программу/фильм, я же не украл что-то в магазине", "производитель сам виноват, что назначил такую высокую цену", "от того, что я скачал, производителю хуже не станет".

Все эти высказывания являются неверными. В разделе III статье 9.21 Кодекса Республики Беларусь об административных правонарушениях от 21 апреля 2003 г. № 194-З указано: "Незаконное распространение или иное незаконное использование объектов авторского права, смежных прав или объектов права промышленной собственности <...> — влекут наложение штрафа в размере от двадцати до пятидесяти базовых величин с конфискацией предмета административного правонарушения или без конфискации <...> ".

Дальнейшее развитие информационных технологий невозможно без создания новых программных средств. Экономические потери производителей программных продуктов и проблема защиты авторских прав приводят к оттоку части способных к творческой деятельности специалистов, снижению качества создаваемых информационных ресурсов.

Стив Балмер, будучи генеральным директором компании Microsoft, однажды высказался о последствиях компьютерного пиратства: "Вам может казаться, что пиратство наносит вред только тем, кто производит программное обеспечение. На самом деле ущерб от пиратства намного больше. Это - и потерянные рабочие места, и низкие зарплаты, и несобранные налоги, и непроданные товары <...>. И, чем больше новых программ появляется на рынке, тем больше нелицензионных копий используется повсеместно <...> ".

Недостаточная эффективность правовых методов защиты интересов создателей программного обеспечения приводит к необходимости создания программных и физических средств защиты.

Под системой защиты программ от несанкционированного использования понимают комплекс мер, предназначенных для усложнения или предотвращения нелегального (без согласования с правообладателем) изготовления копий программ и их использования.

Любой исполняемый файл можно открыть с помощью hex-редактора и убедиться в том, что в нем записаны определенные коды, которые ЭВМ трактует как набор команд. Следовательно, для защиты программ теоретически можно использовать те же средства, что и для защиты данных в привычном для нас смысле (аудио-файлы, изображения, текстовые документы и др.), хотя эффективность таких методов ставится под сомнение. Отличие программ от данных состоит в том, что программа заставляет ЭВМ производить некоторое действия. В связи с этим для защиты программ используются дополнительные более эффективные средства защиты, которые чаще всего не могут быть применены для защиты данных, которые сами по себе не производят никаких действий.

## Обзор существующих методов и средств защиты программ от несанкционированного использования

Средства защиты программного обеспечения можно разделить на *юридические*, *организационные*, *экономические*, *аппаратные* и *программные*. В этой работе подробно будут рассмотрены программные методы.

Авторские права охраняются специальными законами, как на национальном, так и на международном уровне. Республика Беларусь имеет современное законодательство об авторском праве. Законодательство Республики Беларусь об авторском праве состоит из положений Конституции Республики Беларусь, Гражданского кодекса Республики Беларусь, Закона «Об авторском праве и смежных правах», декретов и указов Президента и иных актов законодательства. Ответственность предусмотрена не только за использование контрафактной продукции, а также за преодоление применяемых технических средств защиты.

Организационными называют методы, связанные с грамотной организацией режима доступа к секретам и контроля за секрет-носителями (люди, переносные запоминающие устройства) на предприятии.

Экономические меры предусматривают стимулирование покупателей, легально приобретших товар. Примером можно привести получение различного рода "плюшек" — коллекционные и подарочные издания, плакаты, бонусы для компьютерной игры, и предоставление скидки на последующие покупки.

Часто аппаратные и программные средства защиты объединяют общим понятием "технические средства". Технические средства являются наиболее эффективными методами защиты. Их большим плюсом по сравнению с юридическими мерами защиты является то, что технические средства осложняют (в идеале — не допускают) несанкционированное использование, тогда как юридические вступают в силу после факта совершения преступления. Далее будут рассмотрены существующие программные методы и средства защиты программ. Аппаратные средства защиты будут упоминаться только в их связи с программными средствами.

***Паролирование.***

Дадим несколько вспомогательных определений, для толкования термина "пароль".

Субъект доступа — лицо или процесс, действия которого регламентированы правилами разграничения доступа.

Идентификатор доступа — уникальный признак субъекта или объекта доступа.

Пароль — идентификатор субъекта доступа, который является его (субъекта) секретом.

Упрощенно можно сказать, что пароль — это некий признак, предназначенный для подтверждения личности или полномочий. Часто в качестве признака выступает набор символов.

При установке или запуске программы требуется ввести серийный номер (ключ). Этот метод возник в те времена, когда программы распространялись исключительно на физических носителях. Например, на коробке с компакт-диском был напечатан серийный номер, подходящий только для данной копии программы. С распространением сетей очевидным недостатком стала проблема распространения образов дисков и серийных номеров по сети. Поэтому в настоящий момент метод используется только в совокупности с другими методами.

Примером такой "связки" можно назвать использование паролей и сетевой защиты. Многие программы для своей работы используют централизованные сервера, без которых они либо неэффективны (серверы обновлений антивирусных программ), либо лишены смысла (клиентские программы многопользовательских игр). Программа передает свой серийный номер серверу, который проверяет его, и, если номер неверный, сервер отказывает в услуге.

Пароль не должен храниться в открытом виде. Для хранения пароля в программе используют его хэш — значение хэш-функции. Хэш-функции — это частный случай односторонних функций. Одностороннюю функцию можно определить как функцию, значение которой легко вычислить для любого входного значения , но трудно найти по известному аргумент . Если битовая длина одинакова для любого , то такую одностороннюю функцию называют хэш-функцией. Механизм применения хэш-функций для хранения и проверки паролей: в определенном месте хранится хэш пароль, у пользователя запрашивается пароль, к введенному паролю применяется хэш-функция и полученный результат сравнивается с сохраненным хэшем. Для одинаковых входных параметров значения хэш-функции совпадают. Поэтому, если пароль введен верно, то сравниваемые хэши совпадут.

Как правило, защита только паролем не используется на практике. Над данными или программой выполняются некоторые криптографические преобразования. При этом пароль будет являться ключом или частью ключа для обратного преобразования.

### Криптографическое закрытие

Криптографические методы защиты основаны на возможности осуществления последовательности обратимых преобразований над данными, с использованием некоторого ключа.

К криптографическим методам относят шифрование и кодирование.

Под шифрованием понимается такой вид криптографического закрытия, при котором преобразованию подвергается каждый символ защищаемого сообщения.

Под кодированием понимается такой вид криптографического закрытия, когда некоторые элементы защищаемых данных (не обязательно отдельные символы) заменяются заранее выбранными кодами.

Суть использования криптографических методов для защиты от несанкционированного использования состоит в следующем: зашифрованные данные становятся доступными только для того, кто знает, как их расшифровать, и поэтому похищение зашифрованных данных абсолютно бессмысленно для несанкционированных пользователей.

***При шифровании программы*** поступают двумя способами: шифруется весь код программы или только некоторые критические участки кода. В обоих случаях может применяться как статическое, так и динами­ческое расшифровывание. При статическом расшифровывании весь код (фрагмент кода) один раз расшифровывается. В зашифрован­ном виде код постоянно хранится на внешнем носителе, в расшиф­рованном виде присутствует в оперативной памяти. При динамиче­ском расшифровывании последовательно расшифровываются про­цедуры программы или критического участка программы.

При выполнении зашифрованной программы всегда должен настать момент, когда произойдет расшифровывание части или всей программы. Злоумышленник при изучении программы попытается отловить этот момент создать "снимок" части оперативной памяти с расшифрованным кодом. Поэтому, если не использовать средства запутывания и методы защиты от исследования, то каким бы мощным не был алгоритм шифрования, его использование сведет защиту на нет.

Для того чтобы программе выполнить расшифровывание, ей необходимо каким-либо образом получить ключ. Это можно сделать несколькими способами: запросить его у пользователя (пароля), использовать некоторые уникальные характеристики среды, на которой она выполняется (привязка программ к оборудованию), считать его с внешнего устройства (использование физического воздействия) или совместить все эти варианты для большей надежности, что часто применяется на практике.

### Использование аппаратных средств в защите программ:

Привязка к носителю информации и использование электронных ключей.

*Привязка к носителю информации*

Идея привязки программы или данных к носителю информации, на котором они распространяются, появилась давно. До прихода интернета цифровая информация распространялась преимущественно на дискетах, затем — на компакт-дисках. Идея состоит в том, чтобы программа запускалась и работала только тогда, когда пользователь вставлял оригинальный носитель информации.

Для распространения программ с ограниченным числом инсталляций использовались перезаписываемые носители, и на каждом носителе был счетчик установок. При инсталляции программы счетчик уменьшается, пока не дойдет до ноля, тогда выполнение дополнительных инсталляций становится невозможным. При деинсталляции счетчик снова увеличивается, что позволяет установить программу на другую машину.

Обойти такую защиту можно двумя способами:

1. Перехватить в устанавливаемой программе вызов функции записи на диск. При попытке уменьшить значение счетчика, пропустить эту операцию, не производя никаких действий, и вернуть статус записи "успешно".
2. После установки программы произвести копию содержимого жесткого диска, деинсталлировать программу и восстановить образ диска с копии.

*Электронные ключи*

Электронный ключ (ЭК) — аппаратное средство, предназначенное для защиты программного обеспечения от нелегального использования.

Принцип действия ЭК состоит в следующем: ключ подсоединяется к интерфейсу компьютера, защищенная программа отправляет ему информацию, которая обрабатывается в соответствии с заданным алгоритмом в ключе и возвращается обратно. Если ответ ключа правильный, то программа продолжает свою работу. В противном случае она может выполнить определенные разработчиком действия, например, переключиться в демонстрационный режим, блокировать доступ к определенным функциям.

### Привязка программы к оборудованию

Основная идея, на которой строится защита с использованием привязки программы к оборудованию: во внешнем (по отношению к защищаемой программе) мире нужно найти объект, который обладает тремя свойствами: он должен иметься у законного пользователя, его должно быть сложно воспроизвести при распространении нелегальных копий, объект должен обладать некими уникальными свойствами. Этот объект называется "объектом привязки". После выбора объекта привязки разработчик системы защиты от копирования должен сделать так, чтобы защищаемую программу нельзя было использовать без наличия объекта привязки.

Привязка к оборудованию компьютера базируется на следующем принципе:

1. Конечный пользователь вместе со своей копией программы получает серийный номер — уникальный идентификатор, который позволяет привязать эту программу к одному или нескольким компьютерам.
2. При первом запуске программы пользователю предлагается ввести серийный номер и произвести активацию. При этом внедрённая в код защита производит следующие действия.
   1. Производится сбор информации о параметрах оборудования компьютера.
   2. Эта информация сохраняется на компьютере в виде профиля оборудования.
   3. От профиля оборудования вычисляется хэш-функция, в результате чего получается код оборудования.
3. Код оборудования вместе с серийным номером передаётся на сервер, где производится проверка серийного номера. Проверяется, что серийный номер есть в базе данных и что он ещё не использовался для активации.
4. Если проверка прошла успешно, пользователю возвращается активационный ключ, содержащий подпись профиля оборудования. Активационный ключ сохраняется на компьютере пользователя.
5. При каждом последующем запуске программы, во-первых, проверяется соответствие профиля оборудования реальному оборудованию, а во-вторых, соответствие активационного ключа профилю. Если проверки прошли, использование программы разрешается.

В качестве объектов привязки используются, в основном, серийный номер винчестера, версия BIOS, информация из реестра, MAC-адрес, информация о процессоре, объем оперативной памяти и жесткого диска и др. По отдельности эти идентификаторы не являются уникальными, но вместе дают достаточно точный слепок машины.

### "Запутывание"

Написанная человеком программа может быть понята, проанализирована и разобрана. Методы, упомянутые выше, являются мощными средствами защиты, однако их детальное изучение взломщиком может свести на нет всю их эффективность. Поэтому все они должны применяться совместно с защитой от анализа кода программы.

Обфускация или "запутывание" — это приведение исходного текста программы, машинного кода либо алгоритма к виду, сохраняющему его функциональность, но сильно затрудняющему изучению и пониманию принципов его функционирования. Является частным случаем принципа "безопасность через неясность". Алгоритмы и программы, производящие обфускацию, называются обфускаторами.

Приведем описание некоторых методов обфускации.

* *Встраивание функций и вынос группы операторов.* В первом случае вызов функции заменяется встраиванием кода функции в место вызова. Вынос операторов является обратным преобразованием, приводящим замене группы операторов функцией.
* *Добавление недостижимого кода*. Участок кода называется недостижимым, если поток управления никогда не окажется в этом участке. Эти участки заполняются похожим на действительно исполняемый код. Решение о переходе или не переходе на недостижимый код должно быть, как можно более запутанное.
* *Добавление мертвого кода.* Мертвым кодом называется код, на который будет передано управление, но его выполнение никак не влияет на результат работы. Например, команды типа *NOP*, *XCHG reg, reg*, *OR reg, reg*, *ADD reg, 0*. При внесении мёртвого кода запутыватель должен быть уверен, что вставляемый фрагмент не может влиять на код, который вычисляет значение функции. Например, команды *OR* и *ADD* не являются "совсем безобидными", так как воздействуют на флаги.
* *Добавление избыточного кода*. Так же, как и мертвый код, избыточный код выполняется, но используется в программе. Отличительной особенностью от обычного кода является то, что результат выполнение избыточного кода можно получить более простым способом или результат известен заранее.
* *Объединение функций*. Несколько функций и их параметры объединяются в одну функцию. К списку параметров добавляется еще один для определения действительно необходимой функции.
* *Клонирование функций*. Одна и та же функция копируется несколько раз, применяются различные методы обфускации. Скопированные функции могут различаться сигнатурой (например, фиктивными параметрами).
* *Развёртывание цикла*. Тело цикла дублируется с соответствующим изменением переменной, отвечающей за количество итераций.

Для автоматизации обфускации исходного кода существует ряд программных решений, различающихся своими возможностями. Например, обфускатор исходного текста программ, написанных на языке C/C++ от компании Stunnix [9] поддерживает следующие возможности:

* Замена имен переменных, классов и функций на бессмысленные.
* Замена числовых констант на выражения, вычисляемые во время компиляции.
* Замена символов в строке на их шестнадцатеричное представление.
* Удаление комментариев.
* Удаление пробелов и символов табуляции в исходном тексте.
* Размещение как можно больше исходного текста в одну строку.
* Переименование файлов, содержащихся в проекте.

Необходимо учитывать, что такая обфускация исходного текста никоим образом не отразится на машинном коде программы.

### Противодействие изучению

Для исследования и вскрытия защиты программ взломщиками применяются разнообразные программные средства:

* отладчики,
* дизассемблеры,
* различного рода вспомогательные утилиты, например, которые следят за обращением программы к жесткому диску и реестру, снимающие дамп памяти.

Дизассемблеры (средства статического исследования) используются для изучения кода программы без исходных текстов. Они преобразуют машинный код в текст на языке ассемблера. Отладчики (средства динамического исследования) используются для пошагового выполнения программы и отслеживания содержимого регистров, памяти, стека и т.д. после выполнения каждой инструкции.

Для остановки выполнения программы в определенном месте используются точки останова. Существует три вида точек останова — программные, на участок памяти и аппаратные.

Программная точка останова представляет собой байт 0xCC в коде программы. Программная точка заменяет первый байт инструкции, на которую устанавливается. Процессор, выполнив инструкцию с кодом 0xCC, генерирует отладочное прерывание INT 3. Управление передается отладчику, после чего тот заменяет байт 0xCC в коде программы на первоначальный и останавливается. После восстановления оригинальной инструкции выполнение программы может быть продолжено.

Программная точка останова на участок памяти — это страница в виртуальной памяти процесса с выставленным атрибутом PAGE\_GUARD. Любое обращение к этой странице приведет к выбросу исключения STATUS\_GUARD\_PAGE\_VIOLATION, на которое реагирует отладчик и проверяет, обращение к какому именно адресу в пределах данной страницы привело к генерации исключения.

Адреса аппаратных точек останова хранятся в отладочных регистрах DR0-DR4 процессора. Их ограниченное количество, обычно четыре. Преимуществом аппаратных точек останова перед программными является то, что не происходит модификации кода исследуемой программы и можно выбрать событие, при котором произойдет остановка: выполнение инструкции, запись по адресу, запись или чтение, но не выполнение. Процессор генерирует прерывание INT 1 и передает управление отладчику.

Защититься от статического исследования можно несколькими способами:

* Шифрование программного кода.
* Включение в тело программы переходов по динамически изменяемым адресам, скрытый переход (jmp через ret, ret и call через jmp).
* Генерация исполняемых команд впереди вычислительного процесса, производя операция над кодом или массивом данных. В Windows обычно используются три способа:
  + копирование функции и ее изменение на стеке или в кучу и выполнение оттуда;
  + модификация в секции кода, изменив уровень защиты с помощью VirtualProtect;
  + функция WriteProcessMemory.
* Обфускация.
* Ручной импорт всех необходимых внешних функций.

Это основные методы "запутывания" дизассемблеров. Большинство из них основано на том, что на самом деле во время выполнения программы команды будут отличны от тех, которые были первоначально записаны в анализируемый дизассемблером исполняемый модуль.

Защититься от исследования под отладчиком можно двумя путями:

* Тем или иным способом обнаружить отладчик и передать управление на некоторую ветку реакции на отладчик.
* Поместить в программу код, который нормально выполняется без отладчика, но под отладчиком приводит к аварийному завершению, зависанию компьютера или искажению хода выполнения программы.

Выявить отладчик можно:

* Проверкой времени выполнения отдельных участков программы.
* Проверки результата подсчета контрольной суммы, занимаемой программой в оперативной памяти. Выявит наличие программной точки останова.
* Флаг трассировки и NtGlobalFlag (находится в PEB по смещению 68h, нужно проверить, установлен ли хотя бы один из флагов: FLG\_HEAP\_ENABLE\_TAIL\_CHECK,FLG\_HEAP\_ENABLE\_FREE\_CHECK или FLG\_HEAP\_VALIDATE\_PARAMETERS).
* WinAPI функция IsDebuggerPresent. Этот прием эффективно работает при совмещении с TLS-callbacks.
* Попытаться отправить сообщение отладчику функцией OutputDebugString. Если при вызове ошибки не возникло (GetLastError() == 0), то процесс находится под отладкой.
* Межпоточное взаимодействие. Есть два потока: один выполняет полезный код и периодически извещает второй поток о том, что является активным. Второй поток следит за частотой, с которой "отзывается" первый поток.

Первый способ защиты применим, но имеет существенный недостаток: исследователь может вообще не разбираться, как программа определяет наличие отладчика. Противнику достаточно найти команду, которая отвечает за переход на ветку с реакцией на отладчик, и изменить ее. Поэтому способ номер один необходимо комбинировать со вторым, где явная проверка не производится. Некоторые из них:

* Некоторые отладчики для своих нужд используют стек изучаемой программы. Тогда, поместив вершину стека в код выполняемой программы, отладчик сам модифицирует код, что приводит к непредсказуемой работе программы. В Windows мы получим исключение в связи с запретом по умолчанию записи в секцию кода.
* Использование подсчитанной по коду контрольной суммы для расшифровывания функций.
* Установка обработчика структурного исключения (SEH) и генерация исключения INT 3. Если обработчик получил управление, то отладчик не присутствует [10, с. 352]. С некоторыми отладчиками такой трюк не пройдет, потому что они способны отличить пользовательский вызов INT 3 от установленной точки останова.

Часто аппаратные контрольные точки ставят на начало буфера или WinAPI функции. Каждая аппаратная контрольная точка контролирует область размером в 4 байта. Поэтому:

* При работе с буфером, в который помещаются "чувствительные" данные, первые четыре байта можно использовать в качестве заглушки, обращение к которой либо не происходит, либо ведет взломщика по ложному следу.
* При использовании важных WinAPI функций (открытие файла, получение пароля из поля ввода, загрузка библиотеки и т.д.) не следует их явно вызывать. Большинство из них содержат стандартный пролог push ebp/mov ebp, esp. Программист сам может выполнить эти команды, а потом совершить прыжок через три байта от после начала нужной функции. Более сложной разновидностью данного метода является полное копирование функции в буфер и вызов ее оттуда.

Заметим, что программы, защищенные только от дизассемблирования, могут легко быть изучены с помощью отладчика и наоборот. Вследствие этого для противодействия изучению алгоритмов систем защиты необходимо иметь средства, препятствующие как дизассемблированию, так и пошаговому выполнению.

Самый мощный и в тоже время самый сложный метод защиты от исследования — написание собственного эмулятора процессора, интерпретирующий код защиты или основной программы. При этом невозможна ни декомпиляция, ни отладка штатными средствами. Взломщик должен сначала изучить архитектуру виртуального процессора, написать декомпилятор (дизассемблер), и только потом может приступить к изучению программы. Защита с помощью виртуального процессора используется некоторыми промышленными протекторами: StarForce, Themida, VMProtect.

## Выводы

Не существует средства, которое гарантированно бы защитило программу от несанкционированного использования. Это можно аргументировать тем, что команды программы, находящиеся в исполняемом модуле, однозначно трактуются процессором. Борьба разработчика и взломщика защиты — это гонка вооружений. Несмотря на то, что разработчик всегда может оказаться впереди исследователя, его преимущество носит локальный, временный характер. Для каждой защиты рано или поздно отыскивается метод противодействия, а программы, использующие такую защиту, оказываются взломаны. При этом скорость взлома программы прямо пропорционально полезности разработанного программного. И если разработчиком зачастую руководят только материальные стимулы, то исследователи почти всегда движимы любопытством, интересом. А для увлеченного человека это очень сильный мотив.

Целесообразно применять такую защиту, стоимость снятия которой превышает затраты на ее установку. Защита должна быть тем сложнее, чем больший тираж имеет защищаемая продукция и чем медленнее он изменяется. Наибольший эффект можно достигнуть при использовании комплексной защиты, включающей юридические, организационные, экономические, аппаратные и программные методы.

В современном мире зачастую и не требуется "вечная защита". Перед разработчиком ставится задача разработать такую защиту, чтобы предотвратить несанкционированное использование программ в течение периода, пока они являются актуальными, например, до выхода новой версии программного обеспечения. Такая задача выполнима.

**Постановка задачи:** разработать программное средство, предназначенное для защиты от несанкционированного использования имеющееся программное обеспечение, состоящее из главного исполняемого модуля и динамически подключаемых библиотек.

## Описание предметной области

Существует множество форматов исполняемых файлов для разных операционных систем, например COM, EXE (Windows, DOS), COFF, ELF (Unix). Обычно данные, хранящиеся в исполняемом файле, состоят из нескольких частей:

* заголовки (предполагаемые исполнители инструкций, настройки исполнителя инструкций и окружения, формат инструкций). Исполнители инструкций — это аппаратно-программный комплекс, способный исполнить инструкцию (процессоры, микроконтроллеры, интерпретаторы, виртуальные машины);
* инструкции (собственно код программы). Представлены либо машинными инструкциями, либо исходным кодом (для интерпретируемого языка), либо байт-кодом для виртуальной машины;
* дополнение (данные, используемые для отладки, список библиотек и используемых функций, иконки для ярлыков, тексты, изображения, звуки, видео и др.)

В качестве защищаемых файлов пока следует выбирать наиболее распространенные в наши дни исполняемые PE-файлы для 32-разрядных версий операционной оболочки Microsoft Windows.

### Структура формата PE

Формат PE (Portable Executable) — формат исполняемых файлов, динамически подключаемых библиотек dll, драйверов sys и некоторых других, используемый в 32- и 64-разрядных версиях операционной оболочки Windows. Формат обладает большим количеством полей, поэтому далее будут описаны только те, которые необходимы для реализации поставленной задачи.

В самом общем приближении исполняемый файл в формате PE состоит из набора заголовков и секций. Для того чтобы описать, какие именно данные находятся в секции, используются директории. В одной секции могут находиться данные разных типов. Общая структура PE формата изображена на рисунке 1. Отмечены только самые важные поля.

*Рисунок 1. Общая структура PE формата*

**IMAGE\_NT\_HEADER**

Signature

**IMAGE\_FILE\_HEADER**

NumberOfSections

SizeOfOptionalHeader

Characteristics

**IMAGE\_OPTIONAL\_HEADER**

Magic

ImageBase

AddressOfEntryPoint

SectionAligment

FileAligment

SizeOfHeaders

**IMAGE\_DATA\_DIRECTORY**

VirtualAddress

Size

**IMAGE\_DOS\_HEADER**

e\_magic

e\_lfanew

**IMAGE\_SECTION\_HEADER**

Name

VirtualSize

VirtualAddress

SizeOfRawData

PointerToRawData

Characteristics

**Секция**

**Секция**

*Рисунок 1. Общая структура PE формата*

**IMAGE\_DOS\_HEADER**

e\_magic: WORD — сигнатура, находящаяся по смещению 0 и равная "MZ" (IMAGE\_DOS\_SIGNATURE);

e\_lfanew: DWORD — смещение от начала файла до заголовка PE.

Поскольку и приложения DOS, и приложения Windows имеют расширение .exe, все исполняемые файлы Windows используют схему двойной загрузки. Она состоит в том, что файл начинается с заголовка DOS, за которым следует заглушка (stub), т. е. небольшой exe-файл формата DOS. При попытке загрузить файл из DOS'а исполняется заглушка, а при загрузке файла из Windows загрузчик анализирует заголовок DOS и извлекает из него смещение до настоящего заголовка исполняемого файла. Стандартное поведение, если запустить программу на dos, то она выведет сообщения вида "This program cannot be run in DOS mode".

**IMAGE\_NT\_HEADER**

Signature: DWORD — сигнатура. Должна быть равна "PE\0\0" (IMAGE\_NT\_SIGNATURE).

PE-заголовок содержит в себе заголовок файла и необязательный заголовок. Несмотря на свое название, IMAGE\_OPTIONAL\_HEADER присутствует в PE-файле всегда (необязательным он является с точки зрения общего формата COFF, поскольку не используется в объектных файлах).

**IMAGE\_FILE\_HEADER**

NumberOfSections: DWORD — количество секций в файле. Секции следуют сразу после заголовка (PE-Header);

SizeOfOptionHeader: WORD — размер опционального заголовка (что следует сразу за текущим);

Characteristics: WORD — характеристики.

По полю Characteristics можно определить, имеем мы дело с "самостоятельным" файлом (установлен бит IMAGE\_FILE\_EXECUTABLE\_IMAGE) или с динамически подключаемой библиотекой (установлен бит IMAGE\_FILE\_DLL).

**IMAGE\_OPTIONAL\_HEADER**

Magic: WORD — сигнатура. Для PE32 равно 0x10b;

ImageBase: DWORD — предпочтительный базовый адрес загрузки в оперативную память. Если файл не может быть загружен по данному адресу, то загрузка происходит по произвольному свободному адресу, после чего применяется информация из таблицы перемещаемых элементов. Это необходимо для того, чтобы величины в коде программы и в некоторых заголовках, которые являются абсолютными адресами, продолжали указывать на верные места в памяти. Абсолютный адрес в виртуальной памяти называется виртуальным адресом (Virtual Address, VA). Относительный виртуальный адрес (Relative Virtual Address, RVA) вычисляется относительно ImageBase:

Начиная с данного поля, все поля, которые указывают на место в памяти, являются на самом деле RVA (за исключением адресов в директории TLS);

AddressOfEntryPoint: DWORD — RVA адрес точки входа;

SectionAligment: DWORD — размер выравнивания (байты) секции при выгрузке в виртуальную память. По умолчанию равно размеру страницы виртуальной памяти для данного процессора;

FileAligment: DWORD — размер выравнивания (байты) секции внутри файла;

SizeOfImage: DWORD — размер файла (в байтах) в памяти, включая все заголовки. Должен быть кратен SectionAligment;

SizeOfHeaders: DWORD — размер всех заголовков (DOS, DOS-Stub, PE, Section), выровненный на FileAligment. Задает смещение от начала файла до данных первой секции;

Также заголовок содержит массив описателей каталогов (директорий) данных. На данный момент количество описателей всегда равно 16 (IMAGE\_NUMBEROF\_DIRECTORY\_ENTRIES). Каждый из каталогов данных содержит определенную служебную информацию. Вид этой информации определяется номером каталога в массиве описателей. Например, константа IMAGE\_DIRECTORY\_ENTRY\_IMPORT равна 1. Следовательно, описатель каталога экспорта можно получить как OptionalHeader.DataDirectory[IMAGE\_DIRECTORY\_ ENTRY\_IMPORT].

**IMAGE\_DATA\_DIRECTORY**

VirtualAddress: DWORD — RVA каталога;

Size: DWORD — размер каталога.

Если каталог отсутствует, то оба поля равны 0.

**IMAGE\_SECTION\_HEADER**

Name: BYTE[IMAGE\_SIZEOF\_SHORT\_NAME] — название секции. На данный момент имеет длину в 8 символов.

VirtualSize: DWORD — размер секции в виртуальной памяти. Если больше, чем SizeOfRawData, то дополняется нулями.

VirtualAddress: DWORD — RVA адрес секции.

SizeOfRawData: DWORD — размер секции в файле. Должен быть кратен FileAligment.

PointerToRawData: DWORD — смещение до начала секции в файле. Должен быть кратен FileAligment. Если секция содержит только неинициализированные данные, то это поле равно нулю.

Characteristics: DWORD — атрибуты доступа к секции и правила для её загрузки в память. Например, атрибут для определения содержимого секции (инициализированные данные, не инициализированные данные, код). Или атрибуты доступа — чтение, запись, исполнение.

**Секция**

Сами секции располагаются в файле после всех заголовков секций. Каждая секция выровнена на границу FileAlignment.

Рассмотрим каталоги, которые должен обрабатывать загрузчик для запуска простых исполняемых файлов.

**Каталог настройки адресов (IMAGE\_DIRECTORY\_ENTRY\_BASERELOC)**

Для того чтобы исполняемый файл можно было загрузить с произвольного адреса, он должен содержать таблицу настроек адресов (таблицу перемещений). Таблица разделена на блоки. Каждый блок содержит настройки для 4 Кб данных и начинается с заголовка IMAGE\_BASE\_RELOCATION. Заголовок содержит начальный RVA и размер блока в байтах с учетом заголовка. Таблица настройки заканчивается блоком, у которого заголовок IMAGE\_BASE\_RELOCATION заполнен нулями. Каждый описатель настройки в четырех старших битах содержит тип настройки, а в 12 младших битах – смещение от начального RVA до настраиваемых данных. На сегодняшний день актуальны два типа настройки из 11:

* IMAGE\_REL\_BASED\_HIGHLOW — к данным, расположенным по настраиваемому адресу, прибавить разность между фактическим адресом, и адресом загрузки из опционального заголовка (ImageBase).
* IMAGE\_REL\_BASED\_ABSOLUTE — нет настройки. Используется для выравнивания блоков по границе двойного слова.

**Каталог экспорта (IMAGE\_DIRECTORY\_ENTRY\_EXPORT)**

Эта таблица всегда присутствует в DLL-файлах, поскольку основным назначением динамических библиотек является экспорт символов, доступных другим DLL- и EXE-файлам. Основная цель таблицы экспорта — связать имена и/или номера экспортируемых функций с их RVA, т. е. с положением в виртуальной памяти процесса.

Таблица экспорта называется IMAGE\_EXPORT\_DIRECTORY. Важными полями являются:

Name: DWORD — RVA ASCIIZ-строки, содержащей имя библиотеки;

Base: DWORD — начальный номер экспортируемых символов.

NumberOfFunctions: DWORD — количество элементов в таблице адресов.

NumberOfNames: DWORD — количество элементов в таблице имен и таблице номеров.

AddressOfFunctions: DWORD — RVA таблицы адресов.

AddressOfNames: DWORD — RVA таблицы имен. Имена отсортированы в алфавитном порядке для ускорения поиска.

AddressOfNameOrdinals: DWORD — RVA таблицы номеров (ординалов).

При поиске экспортируемого символа по его имени сначала производится бинарный поиск этого имени в таблице имен. Если имя найдено и его номер в таблице имен равен N, то извлекается N-й элемент из таблицы номеров. Если этот номер равен K, то элемент таблицы адресов с номером (K - Base) содержит RVA данного символа. Данная схема позволяет приписать одному экспортируемому символу несколько разных имен (рисунок 2).

*Рисунок 2. Таблица экспорта*

**IMAGE\_EXPORT\_DIRECTORY**

Name = "mydll.dll"

Base = 1

NumberOfNames = 3

NumberOfFunctions=2

AddressOfNames

AddressOfFunctions

AddressOfOrdinals

"func1"

"func2"

"func3"

1

1

0

0x0002B40

0x00CE42

0x00AB02

0x007E4F

0x0091FC

Из этого правила существует исключение: если RVA в таблице адресов находится в пределах таблицы экспорта, то это RVA перенаправления (Forwarder RVA); в противном случае это обычный RVA символа (Export RVA). RVA перенаправления заменяет экспорт данного символа на экспорт некоторого другого символа из другой библиотеки. RVA перенаправления указывает на ASCIIZ-строку. Строка имеет либо вид "MyDll.exportproc" (экспорт по имени), либо вид "MyDll#12" (экспорт по ординалу).

**Каталог импорта (IMAGE\_DIRECTORY\_ENTRY\_IMPORT, IMAGE\_DIRECTORY\_ ENTRY\_BOUND\_IMPORT, IMAGE\_DIRECTORY\_ENTRY\_DELAY\_IMPORT)**

Таблица импорта присутствует практически во всех PE-файлах. Данная таблица используется для разрешения ссылок из файла на динамически подключаемые библиотеки. Существует три режима импорта: стандартный, связывающий (bound import, старая и новая схема) и отложенный (delay import).

Директория импорта IMAGE\_DIRECTORY\_ENTRY\_IMPORT состоит из массива описателей IMAGE\_IMPORT\_DESCRIPTOR. Каждый описатель для одной библиотеки, но для одной библиотеки может быть несколько описателей. Конец массива определяется по описателю, у которого все поля нулевые.

Схема каталога IMAGE\_DIRECTORY\_ENTRY\_IMPORT изображена на рисунке 3.

*Рисунок 3. Структура каталога IMAGE\_DIRECTORY\_ENTRY\_IMPORT*

**IMAGE\_IMPORT\_DESCRIPTOR**

OriginalFirstThunk (INT)

TimeDateStamp

ForwarderChain

Name

FirstThunk (IAT)

**IMAGE\_THUNK\_DATA**

AddressOfData

**IMAGE\_THUNK\_DATA**

AddressOfData

**IMAGE\_IMPORT\_BY\_NAME**

Hint

Name

OriginalFirstThunk: DWORD — RVA таблицы имен импорта (Import Name Table);

TimeDateStamp: DWORD — 0 в случае стандартного. Дата и время из заголовка динамической библиотеки в случае старой схемы связывающего импорта. -1 в случае новой схемы связывающего импорта. Связывающий импорт будет описан далее;

ForwarderChain: DWORD — индекс первого перенаправленного символа. Используется для старой схемы связного импорта;

Name: DWORD — RVA имени библиотеки;

FirstThunk: DWORD — RVA таблицы адресов импорта (Import Address Table).

Описатель указывает на таблицы IAT и INT (обе являются массивом структур типа IMAGE\_THUNK\_DATA, оканчивающийся структурой, заполненной нулями). Таблица IAT может храниться в отдельном каталоге IMAGE\_DIRECTORY\_ENTRY\_IAT.

Каждая ячейка таблица INT используется для описания символа, адрес которого нужно поместить в соответствующую ячейку таблицы IAT. Во время выполнения программы адрес внешних символов будет использоваться именно из таблицы IAT. Процесс поиска в таблице экспорта загруженной динамической библиотеки необходимого символа и запись его виртуального адреса в таблицу IAT называется динамическим связыванием.

Загрузчик просматривает таблицу INT. Если старший бит поля AddressOfData установлен, то оставшиеся биты задают ординал символа, иначе это поле указывает на описатель символа IMAGE\_IMPORT\_BY\_NAME. Описатель символа состоит из индекса импортируемого символа в таблицы имен библиотеки и указателя на строку с именем. Не все сборщики заносят правильное значение в поле Hint, поэтому имя функции необходимо проверять.

Самый быстрый способ поиска адреса символа в таблице экспорта библиотеки является поиск по ординалу, самый медленный — по имени символа.

Для ускорения связывания процесса импортирования используется связывающий импорт. Этот способ также называют статическое связывание. Суть статического связывания в том, что в таблицу IAT файла прописываются адреса импортируемых символов в предположении, что все динамически загружаемые библиотеки загружаются по прописанным в них базовым адресам. В двух случаях статическое связывание невозможно и приходится прибегать к динамическому связыванию: если библиотека загружена не по ее базовому адресу ImageBase или версия библиотеки отличается от той, с которой было выполнено статическое связывание (проверяется по полю TimeDateStamp в описатели IMAGE\_IMPORT\_DESCRIPTOR и заголовке библиотеки).

Применить старую схему динамического связывания также невозможно в случае перенаправления символа, так как невозможно отследить, соответствуют ли версия библиотеки и дата связывания. Из соображений надежности такие символы всегда связываются заново в процессе загрузки. Поле ForwarderChain содержит индекс первого перенаправленного символа в IAT. По этому индексу хранится индекс следующего перенаправленного символа и т. д. В индексе последнего перенаправленного символа хранится -1.

Новая схема связывающего импорта заносит в поля TimeDateStamp и ForwarderChain значения -1. Информация о такой библиотеке содержится в каталоге IMAGE\_DIRECTORY\_ENTRY\_BOUND\_IMPORT. Каталог имеет структуру, аналогичную каталогу настройки адресов: для каждой библиотеки существует описатель IMAGE\_BOUND\_IMPORT\_DESCRIPTOR, за которым следует таблица описателей IMAGE\_BOUND\_FORWARDER\_REF для каждого перенаправленного импортируемого символа.

Описатель IMAGE\_BOUND\_IMPORT\_DESCRIPTOR содержит имя библиотеки, дату и время ее создания, и количество следом идущих структур IMAGE\_BOUND\_FORWARDER\_REF.

Структура IMAGE\_BOUND\_FORWARDER\_REF состоит из указателя на имя функции и временную отметку создания динамической библиотеки, на которую происходит перенаправление. Благодаря такому устройству, загрузчик может избежать некоторых случаев использования динамического связывания, заменив его статическим связывание, где это возможно.

Отложенная загрузка импорта означает, что библиотека присоединена к исполняемому файлу, но загружается в память не сразу, как при обычном импорте, а только при первом обращении программы к символу, импортируемому из этой библиотеки. Отложенная загрузка импорта не является свойством загрузчика ОС. Она реализуется добавлением к программе дополнительного кода и данных в процессе сборки. Поэтому каталог IMAGE\_DIRECTORY\_ENTRY\_DELAY\_IMPORT в данной работе не рассматривается.

### Исследование популярных упаковщиков

Общий алгоритм работы программы-упаковщика:

1. Упаковщик сжимает содержимое файла. Обычно изменяются секции кода, данных и таблица импорта.
2. Дописывает код распаковщика в сжатый файл.
3. Меняет точку входа (Entry Point, EP) оригинальной программы на код распаковщика. Точку входа оригинальной программы называют OEP (Original Entry Point). Адрес точки входа хранится в IMAGE\_OPTOPNAL\_HEADER PE-файла.

Благодаря сжатию файла и относительно малому размеру кода распаковщика, упакованный файл имеет меньший размер по сравнению с оригиналом. При запуске упакованного файла управление сначала получает код распаковщика. Распаковав в памяти оригинальный файл и произведя необходимые настройки, управление передается на OEP.

В качестве упаковываемого файла возьмем простую программу, написанную на C++:

#include <iostream>

int sum(int a, int b)

{

return a+b;

}

int main()

{

int a, b;

std::cout << "a = ";

std::cin >> a;

std::cout << "b = ";

std::cin >> b;

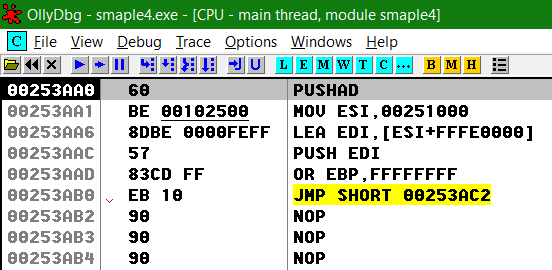
std::cout << "a + b = " << sum(a, b) << std::endl;

}

Используемые инструменты: отладчик OllyDbg, программа для работы с PE-файлами PETools, программа для восстановления импорта Import REConstructor.

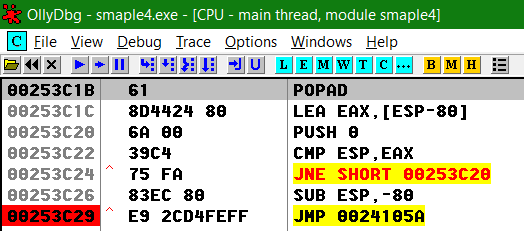
1. *UPX* [13]

Загружаем упакованную программу в отладчик OllyDbg. Попадаем на EP (Рисунок 4).



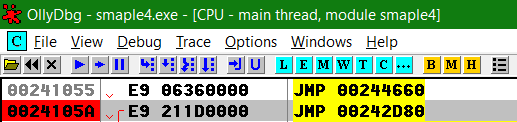
*Рисунок 4. Точка входа в файл, упакованный UPX*

Инструкция *PUSHAD* заносит значения всех регистров общего назначения в стек. Это необходимо для того, чтобы перед переходом на OEP извлечь сохраненные значения, и оригинальная программа могла работать с первоначальными значениями. Следовательно, возле команды перехода на OEP должна находиться инструкция *POPAD —* извлечение из стека значений всех регистров общего назначения. Произведем поиск этой инстуркции. Результат представлен на рисунке 5.



*Рисунок 5. Инструкция POPAD и переход на OEP*

Догадка оказалась верна. Красным цветом выделена инструкция перехода на OEP. Сама OEP имеет вид:



*Рисунок 6. OEP*

Вычислим относительный виртуальный адрес OEP. Имеем: виртуальный адрес загрузки образа в память — 0x00230000, OEP — 0x0024105A. Следовательно, относительный виртуальный адрес OEP

На этом моменте программа в памяти находится в распакованном виде. Чтобы получить исполняемый файл нужно сделать снимок памяти процесса или дамп. Сейчас процесс находится в приостановленном состоянии под отладчиком. Для снятия дампа необходимо, чтобы он находился в активном состоянии. При этом для последующей корректной работы дампа секции кода и данных не должны быть изменены, то есть программа должна находиться в исходном состоянии.

Для этого мы зациклим программу в памяти. Заменим инструкцию *JMP 0024105A* (переход на OEP) на *JMP 00253C29* (переход на саму себя). Запустим программу и отключим от нее отладчик.

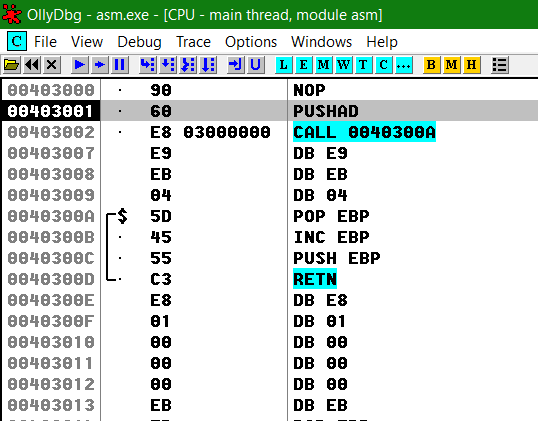
Делаем дамп памяти зацикленного процесса с помощью PETools. Полученный исполняемый файл не может быть запущен, так как в таблице импорта записаны не указатели на имена функций, а адреса функций, то есть произошла операция разрешения импорта. Загрузчик исполняемых файлов операционной системы просто не сможет найти такие функции и откажется запустить файл.

Для автоматического решения проблемы воспользуемся программой Import REConstructor. Запускаем программу и упакованный файл. В программе выбираем необходимый процесс, а в поле ввода OEP вводим вычисленный относительный виртуальный адрес OEP. Нажимаем кнопку "IAT AutoSearch". После нахождения таблицы импорта нажимаем "Get Imports" и добавляем таблицу импорта к снятому дампу с помощью "Fix Dump".

На этом этап распаковки полностью завершен.

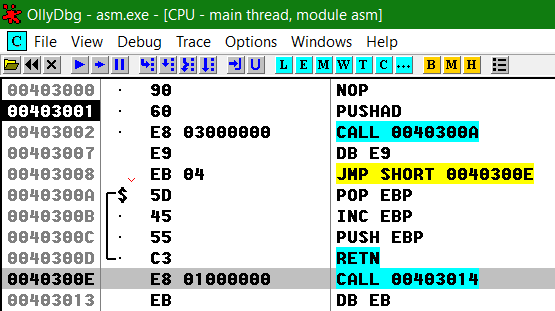
2. *ASPack* [14]

Загружаем упакованную программу в отладчик. Так же, как и в случае с UPX, видим инструкцию *PUSHAD* (Рисунок 7).



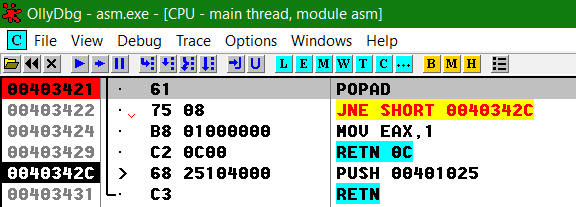
*Рисунок 7. EP программы, упакованной ASprotect*

В самом начале используется прием, осложняющий дизассемблирование программы. Инструкция *CALL* передает управление по адресу *0040300A* и помещает адрес следующей инструкции после *CALL* в стек, то есть *00403007.* В теле псевдофункции из стека извлекается адрес возврата, прибавляется к нему единица и помещается в стек. После этого по новому адресу передается управление. Укажем отладчику, что нужно дизассемблировать инструкцию по адресу *00403007+1 = 00403008.* На рисунке 8 изображена скрытая инструкция перехода и вызова функции.



*Рисунок 8. Результат дизассемблирования после указания начала инструкции*

Пролистав несколько экранов в отладчике, можно сделать вывод, что упаковщик использует переходы по динамически изменяющимся адресам и самомодифицирующийся код. Нет необходимости исследовать весь код распаковщика. Попробуем найти инструкцию *POPAD*. Поставим на найденную инструкцию точку останова и запустим программу.



*Рисунок 9. Момент перехода на OEP в ASPack*

Передача управления на OEP в случае ASPack производится отличным способом от UPX: в стек помещается адрес OEP с помощью инструкции *PUSH*, а следующей инструкцией *RET* он извлекается и передается управление на него.

Снятие дампа и восстановление таблицы импорта аналогично случаю UPX.

По такому алгоритму часто функционируют не только упаковщики, но и программы-крипторы (шифруют код программы) и протекторы. Для борьбы с ними используется метод, продемонстрированный выше:

1. Производится поиск OEP.
2. Зацикливание процесса перед переходом на OEP.
3. Снятие дампа памяти.
4. Восстановление таблицы импорта и изменение EP на OEP.
5. Опционально: очистка файла от кода упаковщика.

Вывод: программа-протектор должна быть как можно глубже связана с защищаемой программой. Для защиты от снятия дампа код защищаемой программы никогда не должен присутствовать в оперативной памяти полностью в открытом виде.

## Проект программного средства

Программное средство состоит из:

* Сервера. Взаимодействует с клиентскими программами по сети;
* Клиентской программы для отправки нового ПО на сервер (uploader);
* Клиентской программы-загрузчика (downloader, инсталлятор);
* ***Установщика защиты. Делится на четыре модуля:*** 
  + тело протектора (модуль, в который оборачивается защищаемое ПО);
  + модуль *A*, заворачивающий ПО в тело протектора и вырезающий защищенные функции, но не шифрующий протектор. Используется каждый раз для нового ПО, добавленного через uploader;
  + модуль *B*, шифрующий подготовленное тело протектора с защищенным ПО. Используется каждый раз для новой установки через инсталлятор. Ключ шифрования различный;
  + модуль *C*, шифрующий вырезанные функции. Используется каждый раз при запуске защищенного протектором ПО к серверу. Ключ шифрования различный.
* Защищенной программы.

Все сетевое взаимодействие между сервером, клиентами и защищенными программами происходит по криптографическому протоколу передачи данных TLS v1.2, что исключает вмешательство злоумышленника. Реализация протокола взята из криптографической библиотеки OpenSSL [15].

С помощью клиентской программы для отправки ПО разработчик (или поставщик) отправляет свой продукт на сервер. Для этого он:

1. Выбирает модули, которые должны быть включены в конечный продукт (exe-файлы и dll-библиотеки).
2. Отмечает модуль, с которого начинается выполнения программы (главный модуль).
3. Выбирает в модулях функции, которые должны быть усиленно защищены: они вырезаются из модуля и отправляются защищенной программе при каждом запуске. При вызове защищенных функций в протекторе используется уникальный механизм, рассматриваемый в пункте "Защищенные функции".

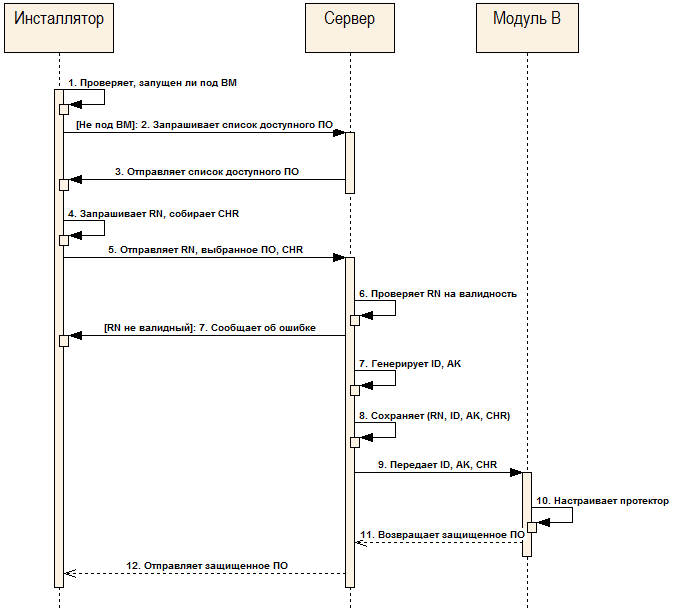
После отправки ПО сервер высылает разработчику набор регистрационных ключей, которые используются для активации и скачивания ПО через инсталлятор.

Схема этапа регистрации приведена на рисунке 10:

1. Инсталлятор проверяет, не запущен ли он под виртуальной машиной. Это необходимо в связи с тем, что в защите используются характеристики ПК. Если бы такой проверки не было, то взломщику достаточно установить одну легальную копию программы на ВМ и после этого распространять активированную программу вместе с виртуальной машиной. Методы проверки:
   * определенные поставщики в CPUID, наличие бита гипервизора,
   * особые ключи реестра,
   * количество процессоров,
   * размер ОЗУ,
   * размер жесткого диска,
   * особый MAC-адрес,

Если определили, что находимся под ВМ, то прекращаем установку.

1. Инсталлятор запрашивает у сервера список ПО, доступного для скачивания.
2. Сервер отправляет ответ.
3. Пользователь делает выбор. Инсталлятор запрашивает у пользователя приобретенный им регистрационный номер (RN) и собирает программно-аппаратные характеристики ПК (CHR). Список WinAPI функций, с помощью которых можно узнать информацию о ПК, представлен на сайте MSDN [16]. Также используется ассемблерная инструкция CPUID.

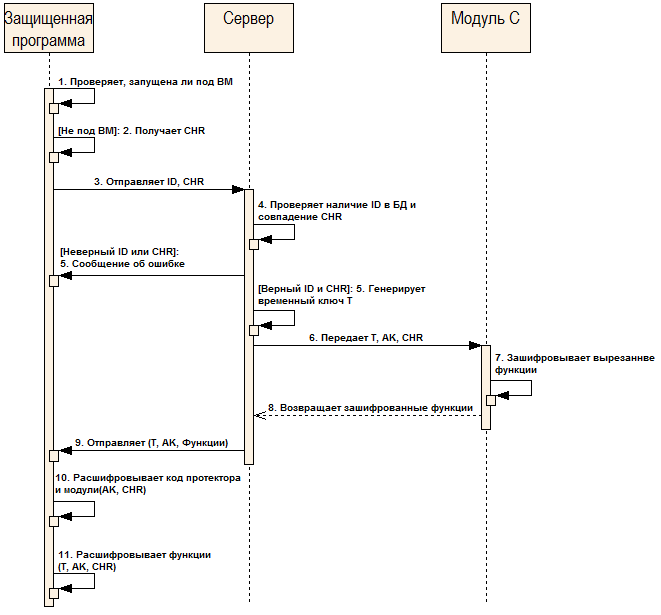


*Рисунок 10. Схема регистрации*

1. Регистрационный номер, характеристики ПК и название выбранного ПО для установки передаются серверу.
2. Сервер проверяет, зарегистрирован ли у него полученный регистрационный номер и если зарегистрирован, то была ли произведена активацию по нему. Если хотя бы один пункт не выполнен, то инсталлятору подсылается сообщение об ошибке и процесс заканчивается.
3. Сервер генерирует уникальный идентификатор регистрации (ID) и дополнительный ключ (Additional Key — AK), который также используется для защиты.
4. Сервер сохраняет соответствие между RN, ID, CHR, AK. RN помечается как использованный
5. Вызывает установщик защиты, а именно модуль *B.*
6. Модуль *B* принимает на вход незащищенное тело протектора (результат работы модуля *A*), добавляет в него ID и шифрует с помощью AK и CHR. Ключ AK на этапе регистрации не передается пользователю.
7. Сервер передает инсталлятору защищенную программу.

На этом этап регистрации закончен.

При каждом запуске защищенной программы происходят действия (Рисунок 11):

*Рисунок 11. Схема запуска*

1. Проверка на запуск в виртуальной машине.
2. Защищенная программа собирает программно-аппаратные характеристики ПК.
3. Защищенная программа отправляет свой ID, характеристики ПК.
4. Сервер проверяет, есть ли в БД присланное ID и если есть, то совпадают ли характеристики ПК с имеющимися в БД. Если хотя бы один пункт не выполнен, то защищенной программе посылается сообщение "Программа не зарегистрирована". В этом случае процесс заканчивается.
5. Сервер генерирует временный ключ T.
6. Сервер с помощью сгенерированного ключа T, имеющегося в базе ключа AK и характеристик ПК CHR шифрует помеченные функции.
7. Сервер отправляет защищенные функции, временный ключ T и ключ AK.
8. Основной код протектора и защищенные модули расшифровывается с помощью AK и CHR.
9. В процессе вызова защищенных функций, их расшифрование их инструкций происходит с помощью T, AK и CHR.

Взломщик, имея исполняемый модуль программы, не может его корректно анализировать, так как в нем отсутствует код защищенных функций, а большая часть протектора зашифрована ключом, который предается сервером только в случае успешной проверки характеристик ПК, на котором запускается программа.

# Список использованной литературы

1. Seizing Opportunity Through License Compliance [Electronic resource] / BSAGlobal Software Survey — Mode of access: [http://globalstudy.bsa.org /2016/downloads /studies/ BSA\_GSS\_US.pdf](http://globalstudy.bsa.org/2016/downloads/studies/BSA_GSS_US.pdf). — Date of access: 13.03.2017.
2. Зубович К. А. Разработка программного обеспечения для персональных электронно-вычислительный машин на языке ассемблера (краткий курс для начинающих) — Минск, 2000. — 396 с.
3. Емельянова Н. 3., Партыка Т. JI., Попов И. И. Защита информации в персональном компьютере учебное пособие. — М. ФОРУМ, 2009. — 368 с.
4. Алферов А. П., Зубов А. Ю., Кузьмин А. С., Черемушкин А. В. Основы криптографии. Учебное пособие, 2-е изд., испр. и доп. — М., Гелиос АРВ, 2002, — 480 с., ил.
5. Скляров Д. Искусство защиты и взлома информации — Санкт-Петербург, БХВ-Петербург, 2004. —271 с.
6. Электронный ключ — средство предохранения софта [Электрон. ресурс] / Компания "Актив" — Режим доступа: <http://www.guardant.ru/press-center/publication/2001-08-01.html>. — Дата доступа: 13.03.2017.
7. Методы защиты от копирования: объекты привязки [Электрон. ресурс] / Компания "Протекшн Технолоджи" — Режим доступа: <http://www.star-force.ru/press/articles/index.php?news=2958>. — Дата доступа: 13.03.2017.
8. A Taxonomy of Obfuscating Transformations [Electronic resource] / C. Collberg, C. Thomborson, D. Low. — Mode of access: [https://researchspace.auckland. ac.nz/handle/2292/3491](https://researchspace.auckland.ac.nz/handle/2292/3491). — Date of access: 13.03.2017.
9. Cxx-Obfus [Electronic resource] / Stunnix Technologies Ltd — Mode of access: <http://stunnix.com/prod/cxxo/>. — Date of access: 10.05.2017.
10. Касперски К. Техника отладки программ без исходных текстов — СПб.: БХВ-Петербург, 2005. — 832 с.
11. Microsoft Portable Executable and Common Object File Format Specification [Electronic resource] / Microsoft Corporation — Mode of access: [http://www. microsoft.com/ whdc/system/platform/firmware/PECOFF.mspx](http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx). — Date of access: 15.03.2017.
12. Структура исполняемых файлов Win32 и Win64 [Электрон. ресурс] / Лукач Ю. С. — Режим доступа: <http://cs.usu.edu.ru/docs/pe/>. — Дата доступа: 15.03.2017.
13. UPX — the Ultimate Packer for eXecutables [Electronic resource] / Markus F.X.J. Oberhumer, László Molnár & John F. Reiser — Mode of access: <https://upx.github.io/>. — Date of access: 10.05.2017.
14. ASPack Software [Электронный ресурс] / StarForce Technologies Ltd —Режим доступа: <http://www.aspack.com/>. — Дата доступа: 10.05.2017.
15. OpenSSL — Cryptography and SSL/TLS Toolkit [Electronic resource] / OpenSSL Software Foundation — Mode of access: <https://www.openssl.org/>. — Date of access: 10.05.2017.
16. System Information Functions: [Electronic resource] / Microsoft — Mode of access: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms724953 (v=vs.85).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms724953(v=vs.85).aspx). — Date of access: 10.05.2017.