**Московский физико-технический институт**
**(государственный университет)**

Эссе по курсу
«Основы защиты информации»
студента 016 группы
Ястребова С.В.
на тему

# Приемы защиты программного обеспечения, облегчающие взлом

**Долгопрудный 2004**

# *Введение*

В процессе разработки защитных механизмов нельзя не учитывать то, что противник будет стремиться использовать все доступные средства для того, чтобы устранить защиту. Тем не менее, часто защищенная программа содержит внутри себя такое количество информации, благоприятствующей взлому, что было бы странно, если противник не воспользовался бы ею.

Целью данной работы является обзор некоторых способов выявления противником слабых мест программных продуктов, которые он может использовать в своих корыстных целях.

## Транслируемые языки

Некоторые распространенные языки программирования при компиляции преобразуют исходный код в так называемый псевдокод - промежуточное представление кода программы, не являющееся машинным. Это, например, Clipper, C#, FoxPro, сценарии инсталляции InstallShield, Java, Maplnfo Map Basic, MicroStation MDL. Python. Visual Basic и другие. В процессе выполнения программы виртуальная машина интерпретирует псевдокод и выполняет его на виртуальном процессоре.

Теоретически использование виртуальной машины может являться эффективным способом противодействия исследованию программы, т.к. до начала анализа необходимо понять, как устроена виртуальная машина. Но это верно только для ситуации, когда система команд, которую применяет машина, никем не была описана.

Понятно, что для популярных языков программирования это совсем не так. Для некоторых языков (таких, как C#, Java и Python) в Интернете легко отыскать подробности о том, как кодируется определенная операция в виртуальной машине. Интерпретатор языка Python вообще распространяется в исходных текстах, что не позволяет сохранять устройство виртуальной машины в тайне.

Если для какого-то языка нет описания кодов операций, через какое-то время кто-то всё равно разберется в тонкостях псевдокода и разработает весь нужный инструментарий.

Есть несколько причин, почему разобраться с системой команд виртуальной машины для транслируемого языка программирования чаще всего бывает не так уж трудно. Сначала исследователь может компилировать любые примеры и наблюдать за псевдокодом, в который превращаются команды. Это очень важно, т. к. при внесении малых изменений в исходный код и анализе разницы в оттранслированном псевдокоде, гораздо проще находить закономерности, чем при изменении в псевдокоде и контроле изменений в поведении виртуальной машины.

Кроме того, виртуальная машина обычно проектируется, исходя из требований максимизации производительности. Следовательно, знание базовых принципов построения эффективных виртуальных машин часто позволяет быстро разобраться с особенностями конкретной реализации. Вдобавок система команд виртуальной машины

редко бывает очень сложной. Это на реальном процессоре обнулить регистр eax можно несколькими способами, например: sub eax, eax; xor eax, eax; and eax, eax; mov eax, 0.

В виртуальной машине это не имеет смысла. И, наконец, виртуальную машину, которая является частью языка программирования, не разрабатывают как запутанное логическое устройство. Наоборот, чем проще организация виртуальной машины, тем легче отлаживать и оптимизировать ее. Для большинства известных транслируемых языков программирования разработаны декомпиляторы, позволяющий получить если не точную копию исходного текста, то эквивалентный код, который можно скомпилировать, и который будет работать так же, как и исходная программа. В разных языках программирования объем информации, сохраняемой в оттранслированном тексте, может отличаться. В некоторых случаях сохраняются имена всех функций и переменных. Иногда могут быть извлечены даже номера строк исходного текста, где располагался тот или иной оператор, и имя исходного файла. Но может быть и так, что сохраняется только последовательность вызовов функций и употребления операторов, в то время как вся символическая информация об именах оказывается утраченной. Но, даже имея всего лишь эквивалентный текст, в котором имена переменных и функций заменены на произвольные, анализировать защитные механизмы гораздо проще, чем, если бы они были написаны на языке, компилируемом в команды реального процессора. Для предотвращения применения декомпиляторов иногда разработчики программ идут на модификацию виртуальной машины.

## Осмысленные имена для функций

Хорошим тоном при разработке сложных проектов является разбиение проекта на части, минимально связанные между собой. Для того чтобы программистам было проще обращаться к частям кода, написанным другими людьми, функциям дают легко запоминающиеся и говорящие имена. Например, легко сообразить, что функция DeleteFile используется для удаления файла. При компиляции программы некоторые части кода могут выноситься во внешние библиотеки и импортироваться по именам. Иногда по имени функции можно восстановить даже количество и тип аргументов функции. Многие интерфейсные среды программирования (например, Borland Delphi и C++ Builder) сохраняют внутри исполняемых файлов строки с именами функций, которые являются обработчиками событий, возникающих в интерфейсе.

Если функция называется CheckPassword, то противнику имеет смысл в первую очередь исследовать именно такую функцию, т.к. скорее всего в ней выполняются действия, относящиеся к защите. Стоит избегать попадания говорящих имен функций, относящихся к защитным механизмам, в исполняемые файлы и библиотеки, являющиеся частью готовой программы. Если в исходном тексте программы функции должны фигурировать под осмысленными именами (для удобочитаемости), то для сокрытия истинных имен функций можно использовать макроподстановки (рис. 1).

**Рис.1**

```
#define CheckPassword fn13

void CheckPassword (char *Psw) {
        /* текст функции */
void main (void) {
        CheckPassword ("Password String");
}
```

При компиляции такого кода препроцессор вместо CheckPassword везде подставит fn13, а значит, имя, полезное для противника, просто не попадется ему на глаза. Библиотечные функции могут импортироваться и экспортироваться не только по имени (by name), но и по номеру (by ordinal). Это позволяет вообще исключить соответствующие имена из программы и библиотек. Однако никогда не помешает в готовых исполняемых файлах выполнить поиск текстовых строк, содержащих названия важных для защиты функций — никогда нельзя быть уверенным, что компилятор или редактор связей нигде не оставил важных имен.

## Demo-версии

Для того чтобы потенциальные пользователи смогли лучше оценить возможности программы, разработчики часто распространяют демонстрационные версии своих продуктов. Такие версии, как правило, имеют ограниченный набор функций и/или ограничение на время использования или количество запусков программы. Условно бесплатные продукты обычно являются ограниченными версиями, которые предоставляют пользователю возможность ввода регистрационного кода, после чего все ограничения снимаются.

## Ограничение функциональности

Если автор демо-версии программы желает сделать недоступным, например, пункт меню Save, то он может выбрать один их двух способов:

1) при инициализации программы сделать этот пункт недоступным;
2) удалить из программы весь код, относящийся к сохранению данных на диске, и при инициализации программы сделать пункт меню Save недоступным.

Понятно, что для реализации первого способа нужно приложить меньше усилий. Однако существуют специальные инструменты, позволяющие менять свойства элементов диалога в процессе выполнения программы. С помощью подобных инструментов можно каждый элемент любого диалогового окна сделать доступным, превратив демонстрационную версию в полноценную по функциональности программу. А некоторые инструменты можно даже "обучить" автоматически делать доступными нужные кнопки и пункты меню при открытии соответствующего диалога.

Поэтому необходимо исключить из кода демонстрационной программы те функции, которые должны присутствовать только в полной версии. Достичь желаемого результата, не создавая две очень похожих программы, можно, например, путем использования директив условной компиляции, поддерживаемых препроцессором языка Си. К полезным директивам относятся, например, #define, #ifndef, #ifdef, #else и #endif. С их помощью можно добиться того, что, изменяя в настройках проекта всего одно определение препроцессора (аналог #define), можно будет из одного набора исходных текстов получить совершенно разные по набору функций программы.

## Ограничение периода использования

Для того чтобы ограничить период возможного использования продукта, необходимо в некоторой области компьютера сохранить дату установки или количество запусков. Обычно для этого используются произвольные файлы или реестр (Registry Database). Многие считают, что, спрятав такой счетчик в самый дальний угол операционной системы, они сделают невозможным обнаружение его местоположения, а, следовательно,

и сброс счетчика. Однако существует два семейства инструментов, позволяющих определить, где именно располагается счетчик. К первому семейству относятся программы-мониторы. Они отслеживают все обращения к файлам пли реестру и протоколируют те из них, которые попросил запомнить пользователь. Мониторы обычно состоят из двух частей: драйвера, устанавливаемого в ядро операционной системы, и интерфейсной части, посредством которой пользователь имеет возможность управлять работой монитора и получать результаты протоколирования. Наиболее известными являются, наверное, программы File System Monitor и Registry Monitor, разработанные компанией Sysinternals. Однако программы могут противодействовать мониторам. Очень важен тот факт, что монитор является активным инструментом — для того чтобы монитор выполнял свои функции, он должен находиться в памяти во время работы исследуемой программы. Следовательно, защищенная программа может обнаружить присутствие монитора и скорректировать свое выполнение разными способами. Например, она может просто отказаться работать, если в памяти присутствует монитор. Другой способ заключается в посылке интерфейсной части монитора сообщения о необходимости завершения работы. При этом монитор оказывается выгруженным из памяти, программа продолжает функционирование в чистом окружении. Красиво выглядит и следующий способ: защищенная программа посылает драйверу монитора команду временно приостановить регистрацию событий, выполняет важные обращения, а затем снова разрешает драйверу работать. При этом интерфейсная часть монитора никак не отражает тот факт, что работа драйвера была откорректирована. И пользователь находится в полной уверенности, что программа не производила никаких обращений, в то время как они имели место, но монитор в это время просто был отключен.

Противодействие мониторам работает только в том случае, если программа знает, как определить наличие монитора в памяти и как его обезвредить. Поэтому часто противнику достаточно изменить логическое имя драйвера монитора, чтобы программа оказалась не в состоянии его обнаружить.

Кроме программ-мониторов, принадлежащих к активным инструментам, есть и пассивные инструменты. Это программы, которые позволяют отслеживать произошедшие изменения, не находясь всё время в памяти. Просто до первого запуска защищенной программы необходимо сохранить текущее состояние реестра или определенных файлов, а после запуска сравнить новое состояние с предыдущим. Те записи, которые были изменены, сразу окажутся на подозрении у противника. Использование пассивных инструментов не может быть обнаружено защитой, и единственный способ противостоять им — внести очень большое количество фиктивных изменений, чтобы затруднить противнику определение действительно важных записей. Но такой подход неминуемо приведет к снижению производительности.

## Программы с возможностью регистрации

Программы, в которых предусмотрена возможность регистрации, выглядит привлекательно с точки зрения маркетинга. Действительно, пользователей должно радовать, что сразу после оплаты стоимости лицензии они получат регистрационный код и могут моментально превратить ограниченную версию в полноценную. Не потребуется ни выкачивание другого инсталляционного пакета, ми повторная инсталляция.

Очевидно, что такая схема может быть уязвлена различными способами.

Очень часто регистрация подразумевает ввод имени пользователя и одного или нескольких регистрационных кодов. Если регистрация не имеет машинозависимой

составляющей, т. е. на любом компьютере определенному имени пользователя соответствуют всегда одни и те же регистрационные коды, то, зарегистрировав программу один раз, ее можно будет использовать на любом количестве компьютеров. Неудивительно, что для очень многих программ в Интернете можно без труда найти регистрационные коды.

Проверка правильности введенного регистрационного кода тоже может выполняться по-разному. Нередко встречаются реализации, когда программа вычисляет для введенного имени пользователя правильный регистрационный код, который просто сравнивается с тем кодом, который ввел пользователь. В подобной ситуации для нелегальной регистрации программы на любое имя достаточно ввести это имя и произвольный код, найти точку, где правильный регистрационный код уже вычислен, и прочитать его из памяти. При следующей попытке регистрации достаточно ввести то же самое имя и прочитанный из памяти код, чтобы программа решила, что она корректно зарегистрирована.

И, конечно же, если регистрационные данные просто проверяются на корректность, ничто не мешает противнику исправить тело процедуры проверки таким образом, что любые введенные регистрационные коды будут считаться правильными. Лучше сделать так, чтобы регистрационная информация использовалась в жизненно важных функциях программы.

Это особенно полезно в свете того, что программа после регистрации должна превратиться в полноценную версию без использования каких-либо дополнительных модулей. Следовательно, ограниченная версия должна в какой-либо форме содержать весь код, доступный в полной версии. Одно из возможных решений — зашифровать фрагменты программы, относящиеся к полной версии, а ключ шифрования каким-то образом связать с регистрационным кодом. При этом без знания правильной регистрационной информации не удастся расшифровать защищенные фрагменты, даже если отмочить вес проверки правильности регистрационного кода.

## Распределенные проверки

После того, как программа получила от пользователя регистрационную информацию, не обязательно сразу же определять ее корректность и оповещать об этом пользователя. Если все проверки сосредоточены в одном месте, противнику намного легче определить, откуда следует начинать анализ.

Лучше всего выбирать такой подход. При вводе регистрационной информации производится начальная проверка, направленная скорее на исключение ошибок набора с клавиатуры, чем для защиты. Если введенная информация окажется правильной, она сохраняется на диске (в файле или реестре) и пользователю выдается сообщение с благодарностью за регистрацию и предложением перезапустить программу, т. к. только после перезапуска будут активированы все функции, недоступные в бесплатной оценочной версии.

После перезапуска программа считывает с диска регистрационную информацию и копирует ее в памяти в нескольких местах. Это делается для того, чтобы противнику сложнее было найти место, где программа интерпретирует регистрационные данные.

В разных местах программы следует вставить разные функции, которые будут проверять разные фрагменты регистрационной информации или анализировать целостность всей программы. Лучше всего иметь несколько функций, которые будут проверять одно и то

же, чем одну функцию, которая проверяет сразу все. Противнику нужно будет разбираться со всеми этими функциями проверки.

Если в результате одной из этих проверок обнаружилось, что регистрационные данные неверны, совсем не стоит сообщать об этом пользователю сразу. Лучше установить специальный флаг, который указывает на то, что программа была некорректно зарегистрирована. А в другой функции, относящейся к совершенно другой части программы, проводить анализ над одним или несколькими флагами и предпринимать соответствующие действия.

Могут иметь место самые разные действия. Так, например, один из DOS-овских симуляторов Formula-1 в процессе запуска проверял наличие документации: пользователя просили ввести слово, которое записано на заданной странице. Исправив один байт, можно было принудить программу запускаться после ввода любого слова, но тогда автомобиль переставал быть управляемым буквально через несколько минут.

Другой пример особого поведения программы. Программа ReGet Deluxe, которая предназначалась для управления выкачиванием файлов по протоколам FTP и HTTP, иногда заменяла загруженные файлы, если устанавливала, что программный код был модифицирован. Поэтому у взломщика ReGet Deluxe была возможность обнаружить в архиве файл readme.txt, в котором содержится текстовая строка "This file downloaded with cracked version of ReGet Deluxe", или получить сообщение с тем же текстом в процессе запуска закачанного файла.

### Инсталляторы с защитой

Некоторые производители программного обеспечения позволяют всем желающем скачать инсталлятор программы с официального интернет-сайта. При этом инсталлятор может быть защищен таким образом, что для установки программы необходимо знать некоторую информацию, которую производитель сообщит только после получения оплаты.

Однако далеко не все разработчики знают, что некоторые способы защиты инсталляторов совсем не такие безопасные, как хотелось бы.

## *Вывод*

Мы видим, что у любого программного продукта, пусть даже принадлежащего известным компаниям и корпорациям, есть слабые места, которые порой являются значительными. Компании теряют свои деньги из-за невнимательности разработчиков и из-за нежелания открыто обсуждать алгоритмы защиты с другими производителями программного обеспечения. Приходится разрабатывать новые, более надёжные схемы защиты программ от взлома. И никогда не знаешь, как противник будет атаковать завтра.

## *Список литературы*

1. Скляров Д., «Искусство защиты и взлома информации», изд. «БХВ-Петербург», 2004
2. Масленников М., «Практическая криптография», изд. «БХВ-Петербург», 2002
3. Шнайер Б., «Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си», изд. «ТРИУМФ», 2002