## Методы защиты программного обеспечения от внедрения на этапе его эксплуатации и сопровождения программных закладок.

###### 3.2.1. Классификация средств исследования программ

В этом подразделе мы будем исходить из предположения, что на этапе разработки программная закладка была обнаружена и устранена, либо ее вообще не было. Для привнесения программных закладок в этом случае необходимо взять готовый исполняемый модуль, дизассемблировать его и после внесения закладки подвергнуть повторной компиляции. Другой способ заключается в незаконном получении текстов исходных программ, их анализе, внесении программных дефектов и дальнейшей замене оригинальных программ на программы с приобретенными закладками. И, наконец, может осуществляться полная замена прикладной исполняемой программы на исполняемую программу нарушителя, что впрочем, требует от последнего необходимость иметь точные и полные знания целевого назначения и конечных результатов прикладной программы.

Все средства исследования ПО можно разбить на 2 класса: статические и динамические. Первые оперируют исходным кодом программы как данными и строят ее алгоритм без исполнения, вторые же изучают программу, интерпретируя ее в реальной или виртуальной вычислительной среде. Отсюда следует, что первые являются более универсальными в том смысле, что теоретически могут получить алгоритм всей программы, в том числе и тех блоков, которые никогда не получат управления. Динамические средства могут строить алгоритм программы только на основании конкретной ее трассы, полученной при определенных входных данных. Поэтому задача получения полного алгоритма программы в этом случае эквивалентна построению исчерпывающего набора текстов для подтверждения правильности программы, что практически невозможно, и вообще при динамическом исследовании можно говорить только о построении некоторой части алгоритма.

Два наиболее известных типа программ, предназначенных для исследования ПО, как раз и относятся к разным классам: это отладчик (динамическое средство) и дизассемблер (средство статистического исследования). Если первый широко применяется пользователем для отладки собственных программ и задачи построения алгоритма для него вторичны и реализуются самим пользователем, то второй предназначен исключительно для их решения и формирует на выходе ассемблерный текст алгоритма.

Помимо этих двух основных инструментов исследования, можно использовать:

* "дискомпиляторы", программы, генерирующие из исполняемого кода программу на языке высокого уровня;
* "трассировщики", сначала запоминающие каждую инструкцию, проходящую через процессор, а затем переводящие набор инструкций в форму, удобную для статического исследования, автоматически выделяя циклы, подпрограммы и т.п.;
* "следящие системы", запоминающие и анализирующие трассу уже не инструкции, а других характеристик, например вызванных программой прерывания.

###### 3.2.2. Методы защиты программ от исследования

Для защиты программ от исследования необходимо применять методы защиты от исследования файла с ее исполняемым кодом, хранящемся на внешнем носителе, а также методы защиты исполняемого кода, загружаемого в оперативную память для выполнения этой программы.

В первом случае защита может быть основана на шифровании секрет-ной части программы, а во втором - на блокировании доступа к исполняемому коду программы в оперативной памяти со стороны отладчиков. Кроме того, перед завершением работы защищаемой программы должен обнуляться весь ее код в оперативной памяти. Это предотвратит возможность несанкционированного копирования из оперативной памяти дешифрованного исполняемого кода после выполнения защищаемой программы.

Таким образом, защищаемая от исследования программа должна включать следующие компоненты:

* инициализатор;
* зашифрованную секретную часть;
* деструктор (деициниализатор).

Инициализатор должен обеспечивать выполнение следующих функций:

* сохранение параметров операционной среды функционирования (векторов прерываний, содержимого регистров процессора и т.д.);
* запрет всех внутренних и внешних прерываний, обработка которых не может быть запротоколирована в защищаемой программе;
* загрузка в оперативную память и дешифрование кода секретной части программы;
* передача управления секретной части программы.

Секретная часть программы предназначена для выполнения основных целевых функций программы и защищается шифрованием для предупреждения внесения в нее программной закладки.

Деструктор после выполнения секретной части программы должен выполнить следующие действия:

* обнуление секретного кода программы в оперативной памяти;
* восстановление параметров операционной системы (векторов прерываний, содержимого регистров процессора и т.д.), которые были установлены до запрета неконтролируемых прерываний;
* выполнение операций, которые невозможно было выполнить при запрете неконтролируемых прерываний;
* освобождение всех незадействованных ресурсов компьютера и завершение работы программы.

Для большей надежности инициализатор может быть частично зашифрован и по мере выполнения может дешифровать сам себя. Дешифроваться по мере выполнения может и секретная часть программы. Такое дешифрование называется динамическим дешифрованием исполняемого кода. В этом случае очередные участки программ перед непосредственным исполнением расшифровываются, а после исполнения сразу уничтожаются.

Для повышения эффективности защиты программ от исследования необходимо внесение в программу дополнительных функций безопасности, направленных на защиту от трассировки. К таким функциям можно отнести:

* периодический подсчет контрольной суммы области оперативной памяти, занимаемой защищаемым исходным кодом; сравнение текущей контрольной суммы с предварительно сформированной эталонной и принятие необходимых мер в случае несовпадения;
* проверку количества занимаемой защищаемой программой оперативной памяти; сравнение с объемом, к которому программа адаптирована, и принятие необходимых мер в случае несоответствия;
* контроль времени выполнения отдельных частей программы;
* блокировку клавиатуры на время отработки особо секретных алгоритмов.

Для защиты программ от исследования с помощью дизассемблеров можно использовать и такой способ, как усложнение структуры самой программы с целью запутывания злоумышленника, который дизассемблирует эту программу. Например, можно использовать разные сегменты адреса для обращения к одной и той же области памяти. В этом случае злоумышленнику будет трудно догадаться, что на самом деле программа работает с одной и той же областью памяти.

###### 3.2.3. Анализ программ на этапе их эксплуатации

В данном разделе будут рассмотрены методы поиска и нейтрализации РПС с помощью дизассемблеров и отладчиков на этапе эксплуатации программ. То есть задача защиты в отличии задач защиты в предыдущих разделах здесь решается "с точностью до наоборот".

Основная схема анализа исполняемого кода, в данном случае, может выглядеть следующим образом

* выделение чистого кода, то есть удаление кода, отвечающего за защиту этой программы от несанкционированного запуска, копирования и т.п. и преобразования остального кода в стандартный правильно интерпретируемый дизассемблером;
* лексический анализ;
* дизассемблирование;
* семантический анализ;
* перевод в форму, удобную для следующего этапа (в том числе и перевод на язык высокого уровня);
* синтаксический анализ.

После снятия защиты осуществляется поиск сигнатур (лексем) РПС. Примеры сигнатур РПС приведены в работе . Окончание этапа дизассемблирования предшествует синтаксическому анализу, то есть процессу отождествлению лексем, найденных во входной цепочке, одной из языковых конструкций, задаваемых грамматикой языка, то есть синтаксический анализ исполняемого кода программ состоит в отождествлении сигнатур, найденных на этапе лексического анализа, одному из видов РПС.

При синтаксическом анализе могут встретиться следующие трудности:

* могут быть не распознаны некоторые лексемы. Это следует из того, что макроассемблерные конструкции могут быть представлены бесконечным числом регулярных ассемблерных выражений;
* порядок следования лексем может быть известен с некоторой вероятностью или вообще не известен;
* грамматика языка может пополняться, так как могут возникать новые типы РПС или механизмы их работы.

Таким образом, окончательное заключение об отсутствии или наличии РПС можно дать только на этапе семантического анализа, а задачу этого этапа можно конкретизировать как свертку терминальных символов в нетерминалы как можно более высокого уровня там, где входная цепочка задана строго.

Так как семантический анализ удобнее вести на языке высокого уровня далее проводится этап перевода ассемблерного текста в текст на языке более высокого уровня, например, на специализированном языке макроассемблера, который нацелен на выделение макроконструкций, используемых в РПС.

На этапе семантического анализа дается окончательный ответ на вопрос о том, содержит ли входной исполняемый код РПС, и если да, то какого типа. При этом используется вся информация, полученная на всех предыдущих этапах. Кроме того, необходимо учитывать, что эта информация может считаться правильной лишь с некоторой вероятностью, причем не исключены вообще ложные факты, или умозаключения исследователей. В целом, задача семантического анализа является сложной и ресурсоемкой и скорее не может быть полностью автоматизирована.