УДК 681.3

Е.Г. КАЧКО, канд. техн. наук, М.В. БЛАГАЙ

ИСПОЛЬЗОВАНИЕ DLL ПРИ РАЗРАБОТКЕ ЗАЩИЩЕННЫХ ПРОГРАММ

В настоящее время широкое распространение получила технология динамического связывания библиотек (Dynamic Link Libraries), которая позволяет оптимально использовать и хранить готовые процедуры и функции, а также ускоряет разработку больших проектов. Практически любая современная операционная система имеет возможности динамического подключения библиотек. Эта статья описывает некоторые структурные особенности реализации динамических библиотек в ОС WINDOWS, снижающие защищенность проектов в целом. Предлагаются методы повышения защищенности таких проектов.

1. Динамическое связывание библиотек

Существует два основных формата файлов динамических библиотек: более старый 16 битный NE (New Executable) и относительно новый 32 битный, наиболее распространенный, PE (Portable Executable) или COFF. NE впервые появился в Windows 3.0, его внутренняя структура довольно сложна и в настоящее время такие DLL мало используется. Формат PE распространен очень широко, так как используется практически везде – в исполняемых файлах приложений под Windows, в динамических библиотеках, в объектных файлах. Структура PE или COFF описывается в документации MICROSOFT.

Рассмотрим простейший пример. Пусть нам необходимо написать программу для кодирования файлов, для входа в которую необходимо ввести персональный ключ-пароль. Имея представление об алгоритмах кодирования, мы выбираем наиболее приемлемый из них, и либо реализуем его сами в виде DLL, либо покупаем готовую динамическую библиотеку. Защиту паролем также реализуем при помощи DLL, в которой производится проверка допустимости введенного пароля. Такая структурная организация позволяет оперативно вносить изменения в систему кодирования и проверки пароля заменой или модификацией DLL. Имея несколько библиотек с различными алгоритмами кодирования можно гибко менять возможности готового приложения, заменой библиотек.

Допустим, программа написана и функционирует без отказов. Рассмотрим возможные пути взлома, которые возникают в связи с использованием динамически подключаемых библиотек.

Пускай, в этом гипотетическом приложении мы используем две динамические библиотеки CODER.DLL и PASSWORD.DLL. На самом деле, имена DLL несут довольно небольшую смысловую нагрузку. В CODER.DLL имеется две функции Coding и Decoding, а в PASSWORD – функция PasswordCheck. Использование длинных символьных имен функций и переменных, описывающих назначение функции или переменной, позволяет лучше ориентироваться в исходном тексте программы, и является хорошим тоном в программировании. Обычно, при компиляции все имена заменяются адресами, но не в случае с DLL. Если используются динамические библиотеки, существует вероятность того, что имена функций останутся в коде программы.

Рассмотрим сам процесс динамического связывания. При компиляции программы, использующей функции из DLL, компилятор заменяет вызов функции из DLL пустышкой следующего вида:

```
Call Dword Ptr [ Function_Stub_Address ] ,
```

либо второй вариант

```
Call near Function_Stub
.....
Function_Stub: Jmp Dword Ptr [Function_Stub_Address]
```

Этот фрагмент кода в мнемониках ассемблера означает *Переход по адресу, находящемуся по адресу Fuction_Stub_Address* . Таким образом, в коде программы не указывается адрес функции, а только

адрес, по которому хранится указатель на функцию. Во время выполнения программы, процессор перейдет по адресу, который указан в двойном слове Function_Stub_Adderss. Вся работа по установке правильных адресов во все такие заглушки выполняется операционной системой на этапе загрузки программы или библиотеки в память. Загрузчик ОС получает полную информацию, необходимую для связывания, из файла программы, причем существует два способа связывания – импортирование по имени функции, либо по номеру функции. Рассмотрим формат PE файла.

2. Формат PE файла

Файлы формата PE [1]состоят из секций, каждая из которых имеет свое назначение. Из основных можно назвать – Text, Data, BSS, Relocs, TLS, idata, edata. В первой собран весь код программы, во второй все инициализированные данные, в третьей – не инициализированные данные. Секция Relocs предназначена для хранения информации о необходимых привязках, она используется если образ программы (Program Image) загружается по другому адресу, отличному от стандартных 4Mb для Windows 95 или 1Mb для Windows NT. В секции TLS описаны все нити или цепочки (Threads) программы. В последних двух секциях хранится информация об экспорте – импорте. На основе именно этой информации и происходит связывание динамических библиотек.

Object table:

#	Name	VirtSize	RVA	PhysSize	Phys off	Flags
01	.text	00026000	00001000	00025800	00000600	[CER]
02	.data	0001C000	00027000	00005400	00025E00	[IRW]
03	.tls	00001000	00043000	00000200	0002B200	[IRW]
04	.idata	00001000	00044000	00000A00	0002B400	[IR]
05	.edata	00001000	00045000	00000200	0002BE00	[IR]
06	.relocs	00002000	00047000	00001A00	0002CC00	[ISR]

Key to section flags:
C - contains code
E - executable
I - contains initialized data
R - readable
S - shareable
W - writeable

Exports from Coder.dll
11 exported name(s), 11 export addresse(s). Ordinal base is 1.

Ordinal	RVA	Name
0000	00028ea8	_DebugHook
.....	.....	.....
0003	000035b8	_gostCoding
0004	00003646	_gostDecoding
0005	000013dc	_readCodeKeys
.....	.....	.....
0011	00001468	_shifrFile

Рис. 1

[1] Питрек Мэтт. СЕКРЕТЫ системного программирования в Windows 95; Киев: Диалектика 1996

Существует множество программ, позволяющих получить информацию о PE файле и его секциях Практически все языки, позволяющие писать под Windows, имеют утилиты для получения подобной информации. Рассмотрим нашу гипотетическую программу при помощи утилиты TDUMP из Турбо Ассемблера пятой версии.

Файл, полученный при помощи TDUMP, содержит много информации о программе, но нас прежде всего, интересует таблица секций и секции импорта-экспорта. В таблице секций подробно описано, где в файле и в памяти располагаются секции программы (рисунок). Рассмотрим таблицу секций на примере простейшей DLL:

Как видно из рисунка, таблица секций содержит полное описание всех секций файла. Нас наиболее интересует импорт, в главный исполняемый файл, функций из DLL. В EXE файле эти функции будут находиться в секции idata, в DLL они попадут в секцию edata.

Просмотрев секцию экспорта в DLL, можно найти символьное имя функции, определить язык, на котором написана эта функция, и получить точку входа в функцию. Например, функция gostCoding скорее всего занимается кодированием, написана на языке C (признаком применения C может служить символ подчеркивания '_', который автоматически вставляется компилятором) и применяет правила передачи параметров C.

Термин RVA, применяемый в таблицах, расшифровывается как относительный виртуальный адрес (Relative Virtual Address). Это смещение в памяти по отношению к адресу, с которого начинается отображение файла в памяти. Phys off это смещение в файле программы Для вычисления точки входа в функцию необходимо вычислить смещение точки входа по отношению к началу секции кода и скорректировать полученное значение с учетом смещения секции кода в файле. Для этого возьмем значение RVA функции из таблицы экспорта, вычтем из этого значения RVA секции .text из таблицы секций и прибавим значение из поля Phys off для секции .text. В файле CODER.DLL по адресу 0x2BB8 (0x35B8-0x1000+0x600) находится следующий код:

```
00002BB8:    55        Push      Ebp
00002BB9:    8BEC      Mov       Ebp,esp
00002BBB:    83C4F0    Add       Esp,FFFFFFF0
```

Это стандартный код пролога, создаваемый компилятором для всех функций. Данный фрагмент выделяет 16 байт в стеке для локальных переменных функции. Замена первого байта 0x55 на 0xC3, код команды возврата из подпрограммы, даст очень интересный результат:

```
00002BB8:    C3        ret
00002BB9:    8BEC      mov       Ebp,esp
00002BBB:    83C4F0    add       esp,FFFFFFF0
```

Теперь процедура не выполнится ни разу – процессор сразу же выйдет из функции. Таким образом, можно очень легко и быстро отключить любую функцию из любой DLL. Получив точку входа в функцию можно внести любые изменения, получить код функции, а так же перехватить передаваемые функции параметры.

Для перехвата параметров используется немного другой механизм, основанный на тех же принципах. Пишется фрагмент кода, сохраняющий параметры в буфере и выполняющий переход на точку входа в функцию. В этом случае корректируется таблица экспорта DLL, либо таблица импорта EXE файла – в зависимости от того к какому файлу приписывается этот фрагмент. При вызове функции, сначала выполняется этот фрагмент затем сама функция. При желании, можно перехватить и результат – модифицировав адрес возврата на точку входа своего фрагмента кода. Учитывая, что обычно PE файлы имеют файловое выравнивание по 512 байтам (размер сектора на диске), добавление небольшого фрагмента даже не изменит длины файла. При этом автоматически решается проблема раздельных адресных пространств Windows NT. При желании, можно добавить новые секции к PE файлу так как обычно после нее в файле имеется около 400 байт свободного пространства. При длине описания сек-

ции 40 байт получаем место для 10 дополнительных секций. В данный момент существуют вирусы, которые используют свободное пространство в конце секций для хранения своего кода, при этом длина инфицированного файла не изменяется. Одним из таких вирусов является CIH.

В нашем случае функция PasswordCheck прерывается таким образом чтобы любой пароль считался корректным, функция Coding просто прерывается. Вся процедура внесения изменений не займет много времени и может выполнятся вручную, без применения утилит типа TDUMP с помощью простого редактора.

3. Анализ возможной угрозы и варианты её предотвращения

Рассмотренный метод взлома основан на использовании динамических библиотек с импортированием по имени. Нетрудно догадаться, что использование импортирования по номеру существенно затруднит взлом. Именно так поступают программисты MICROSOFT, если необходимо скрыть какую либо функцию. Найти нужную функцию, имея только точки входа теоретически реально, но практически почти невозможно. Однако использование импортирования по номеру функции не рекомендовано, так как при совпадении номеров могут возникнуть проблемы совместимости, по - этому большинство приложений используют импортирование по имени функции И конечно этот метод взлома не работает в случае, когда функция не вынесена в DLL.

Существует множество методов противодействия, опишем вкратце основные из них.

Избежать всего выше приведенного можно очень простым и старым методом - просто не выносить особо критичные секции кода из основной программы. Однако при этом теряется гибкость, о которой говорилось в начале статьи.

Другая возможность состоит в использовании импортирования по номеру функции. Однако этот метод имеет следующий недостаток – если кто-либо напишет DLL с функцией, импортируемой по такому же номеру, как и у вашей функции, то могут возникнуть проблемы.

Третий вариант состоит в использовании оверлеев. Можно попробовать использовать и стандартные DLL, но загружать их в процессе работы программы, а не на этапе загрузки. При этом перед загрузкой проверяется целостность файла.

Последний, наиболее простой и результативный метод состоит в том, чтобы скрыть имена функций в DLL и проверить их сигнатуру перед вызовом. Этот метод устраняет возможность определения предназначения функции по её имени и дополнительно проконтролировать код функции на точке входа. Сокрытие имен может производиться либо на этапе разработки, либо по окончании отладки всего приложения.

В первом случае функции, выносимые из основной программы в DLL, называют абсолютно нейтральными именами – например буквенно-цифровым кодом. В главном файле на каждую внешнюю функцию пишется либо функция-пустышка, либо макрос с «интуитивно» понятным именем. В этом случае в секциях импорта/экспорта останется только цифровой код, как имя внешней функции. Если код будет достаточно длинен (5 – 10 символов) возможность совпадения с именем уже существующей функции будет исключена.

Во втором случае сокрытие имен производится в уже готовом приложении, после окончания отладки. Процесс сокрытия состоит в замене имен функций как в DLL так и в EXE файлах буквенно-цифровым кодом. Такой подход позволяет вести разработку и отладку приложения без дополнительных трудностей с именами, а после её окончания при помощи небольшой утилиты провести сокрытие имен.

Обычно приложения под Windows имеют в составе кроме DLL, написанных программистом, несколько стандартных DLL, поставляемых с языком программирования. Файлы DLL также имеют имена, по которым можно определить их принадлежность. Сокрытие может выполняться и с именами DLL. Для этого файлы переименовывают и поправляют их имена в секции импорта EXE файла.

Для контроля сигнатуры функции необходимо сохранить в основном файле приложения небольшой фрагмент начала кода функции, а перед её вызовом сравнить её с кодом вызываемой функции.

Адрес функции может быть легко получен в любом языке программирования, однако необходимо уточнить для конкретного компилятора, как создаются заглушки импортируемых функций. В качестве адреса функции может передаваться адрес двойного слова Function_Stub_Address, либо адрес заглушки Function_Stub, в этом случае реальный адрес функции будет храниться по этому адресу. Данный метод позволяет проконтролировать точку входа, обнаружить вышеописанную модификацию кода и попытку перехвата параметров функции.

Описанный метод позволяет эффективно предотвратить взлом приложения, использующего динамическое связывание.

Харьковский государственный технический университет радиоэлектроники

Поступила в редколлегию 15.03.2000