# Контрольная работа

Методы взлома защиты программ

Введение

Цель: Научиться использовать средства статического и динамического анализа программ. Разобраться в основных принципах работы компилятора при генерации кода на примере MS Visual Studio 2003 (C++).

Предисловие

Данная работа основывается на том, что студент 4 курса специальности ФБ прослушал следующие дисциплины:

* «Программирование на языке С++»
* «Системное программирование»
* «Архитектура ЭВМ»
* «Основы защиты информации».

Подразумевается, что студент, прослушавший данные курсы, обладает следующими навыками:

* Ознакомлен с языком программирования С++, т.е. умеет программировать на уровне «черного С» (без использования расширенных средств языка С++) (курс «Программирование на языке С++»).
* Ознакомлен c языком программирования Assembler, т.е. умеет различать между собой команды языка, и знает их назначение (курс «Архитектура ЭВМ»)
* Ознакомлен со структурой исполняемых файлов формата PE (для Windows), т.е. знает, что такое заголовок исполняемого файла, имеет представление о внутренней структуре файла (курсы «Архитектура ЭВМ», «Системное программирование»)
* Ознакомлен с методами защиты программ от взлома, и методами взлома (курс «Основы защиты информации»)

## Теоретические сведения

### HIEW 6.81

Данная программа является одним из лучших и наиболее удобных шестнадцатеричных редакторов и пользуется заслуженной популярностью на территории стран СНГ. Она обладает приятным и понятным интерфейсом.

Для нормальной работы с данной программой необходимо знать некоторые управляющие комбинации и режимы работы программы.

При старте программы без параметров перед пользователем открывается окно выбора нужного файла (рис. 1) Для определенности выберем файл Hiew.hlp из родного каталога программы (рис. 2).

Как можно видеть, по умолчанию данный редактор работает в режиме обычного просмотра текста. Теперь нажмем клавишу ENTER (аналогично можно нажать клавишу F4 и выбрать необходимый режим просмотра информации), таким образом совершив переход от обычного просмотра в режиме текста в шестнадцатеричное отображение информации (рис. 3). Теперь окно редактора разбито на 3 колонки. Крайняя левая представляет собой смещение байта относительно начала файла, центральная область – шестнадцатеричное представление информации, крайняя правая – текстовое отображение информации.

Для того чтобы перейти в режим редактирования текста, необходимо нажать клавишу F3. После этого можно менять все байты, находящиеся в центральной области окна редактора, на необходимые пользователю. Чтобы сохранить изменения, нужно нажать клавишу F9.

При работе с редактором нужно помнить, что он открывает файлы с возможностью чтения другими программами, но не на запись. Это значит, что если вы в данный момент редактируете при помощи HIEW какой-то файл, то другая программа, пытающаяся внести изменения в этот же файл, не сможет обратиться к этому файлу. К этому относится также попытка запуска приложения, если вы его в данный момент редактируете.

Теперь рассмотрим практический пример – откроем файл приложения Windows Notepad. Перед этим рекомендуется сделать его резервную копию.

Нажмем клавишу F9 и перейдем в режим выбора файлов. После того, как файл был выбран, перейдем в режим дизассемблера (клавиша F4, либо два раза ENTER).

Рис. 1. Окно HIEW при запуске без параметров.

Рис. 2 Текстовый файл, открытый в HIEW

Рис. 3 Шестнадцатеричный режим отображения информации

Рис. 4. Отображение кода программы Блокнот в HIEW

А теперь внесем в программу необратимые изменения, после которых она не сможет запуститься. Для этого нажмем F8 и посмотрим, где находится точка входа в программу (рис. 5).

Рис. 5. Информация о заголовке файла Блокнот

Как видим, точка входа находится по адресу 739d. Перейдем в нее. Для этого выйдем из режима просмотра заголовка, нажав клавишу ESC, а затем F5. Наберем «.739D». НЕ ЗАБУДЬТЕ ПРО ТОЧКУ ПЕРЕД ШЕСТНАДЦАТИРИЧНЫМ ЧИСЛОМ!!!!! Теперь на экране должно быть нечто похожее на то, что изображено на рис. 6.

Рис. 6. Точка входа Блокнота

Теперь «поиздеваемся» над Блокнотом, заменив вызов одной процедуры на пустые операции. Для этого нажмем F3, и затем последовательно заменим байты со значением 6A70 на 9090 (0х90 = nop – отсутствие какой-либо операции). После этого нажимаем F9 для того, чтобы сохранить изменения и нажимам ESC для выхода из HIEW.

Отметим, что зачастую nop'ами приходится «забивать» не один байт, а целый кусок программы; для этого удобнее отметить блок байт (клавиша «\*»), а затем выполнить команду FillBlk (ALT-F2).

Рисунок 7 Измененная точка входа Блокнота

Запустим полученную версию Блокнота. Как видим, она не запускается. Таким образом, мы научились уничтожать полезный код программы путем замены нужных байтов на nop. Помните, что в данной лабораторной работе львиную долю защиты можно снять таким методом, главное знать, что и где заменять. Отсюда следует вывод: ПРИ НЕОБХОДИМОСТИ ОБЕЗВРЕДИТЬ УЧАСТОК КОДА ПУТЕМ УДАЛЕНИЯ ЕГО ФУНКЦИОНАЛЬНОСТИ ИЗ ПРОГРАММЫ, НЕОБХОДИМО ЗАМЕНИТЬ НУЖНУЮ ОПЕРАЦИЮ ЭКВИВАЛЕНТНЫМ КОЛИЧЕСТВОМ БАЙТ 0х90.

Для поиска в редакторе нужной строки достаточно нажать клавишу F7 и ввести то, что вам нужно найти либо в ASCII, либо в шестнадцатеричных кодах. Если вхождений нужной строки более чем одно, то для поиска оставшихся вхождений необходимо использовать комбинацию клавиш CTRL+F7.

За более детальным описанием команд обращаться к функции помощи данной программы F1. Информация файлов помощи контекстно-зависима.

### IDA Pro v4.51

Для выполнения данной лабораторной работы нам понадобится возможность IDA показывать нам таблицу экспорта и импорта функций программы. Собственно, любая программа под Windows не является вещью в себе, а хоть как-то должна взаимодействовать с окружающей средой, в нашем случае – операционной системой. Поэтому как минимум одну функцию любая программа должна экспортировать: точку входа. Можно сказать, что в нашем случае она может послужить отправной точкой в исследовании программы. Кроме этого, большинство программ импортируют функции WinAPI, и их имена представлены в таблице импорта. В нашем случае нам понадобятся функции, ответственные за вывод информации на экран и взятие информации из элементов окна. Дабы облегчить поиск студента, ниже приводятся функции, задействованные в данной лабораторной работе:

MessageBoxA();

GetDlgItemTextA();

Из этого следует, что в работе необходимо будет искать места вызова данных функций. На этом теоретическом отступление заканчивается.

### Практическое использование IDA.

Рис. 8. Основное окно IDA

При запуске IDA можно наблюдать окошко, представленное на рис. 8. Дабы почувствовать, как работает система, выберем нашу программу для анализа, и получим результат, изображенный на рис. 9.

Рис. 9. Окно запроса о методе анализа файла.

Нажимаем на ОК и получаем следующее (рис. 10).

Рис. 10 .Завершенный анализ исполняемого файла

Данное окошко получается после окончательного анализа программы. После этого выбираем вкладку Imports, чтобы посмотреть список функций, используемых программой (рис. 11).

Рис. 11. Открытое окно таблицы импорта

Щелкнув двойным щелчком на функции GetSystemTimeAsFileTime, мы увидим картину, изображенную на рис. 12.

Рис. 12. Информация об импортированной функции GetSystemTimeAsFileTime

Как можно видеть из рис. 12, после совершенной операции перед нами предстало описание импортируемой функции с ее параметрами и местом вызова в программе (; DATA XREF: .text:00401D75). Теперь, если нам необходимо узнать, где в программе вызывается данная функция, то достаточно сделать двойной щелчок по надписи .text:00401D75 и IDA автоматически переведет Вас в нужное место вызова (рис. 13). Таким образом, можно просматривать любую цепочку вызовов функций.

Рис. 13. Место вызова функции GetSystemTimeAsFileTime

Стоит отметить очень мощную функциональность IDA в плане навигации по тексту. Чтобы вернуться в прежнее место (то есть момент до перехода по ссылке), достаточно всего лишь нажать клавишу ESC. Кроме этого, очень удобным является графическое отображение переходов в программе стрелочками в правой стороне окна дизассемблера.

Собственно, данной информации вполне достаточно для выполнения лабораторной работы.

### Работа с отладчиками Visual Studio и OllyDbg

Даже для решения самых тривиальных задач в области реверсинга необходим отладчик: он позволяет динамически отслеживать ход работы программы, вносить изменения «на лету» в ее образ в памяти, следить за состоянием регистров и оперативной памяти и многое другое. Среди современных отладчиков наиболее популярными являются SoftIce из комплекта Compuware Driver Studio и OllyDbg. SoftIce работает в режиме ядра и поэтому позволяет исследовать драйверы и компоненты Windows, работающие в ring0; однако в большинстве случаев его мощность является избыточной и целесообразнее использовать более “дружественный» OllyDbg. К тому же, OllyDbg является freeware и доступен для бесплатного скачивания в Интернете. Кроме того, мы также рассмотрим встроенный отладчик Microsoft Visual Studio.

Отладчик Visual Studio

На основе данного отладчика дается представления о коде программы, который необходимо будет искать. Итак, приступим. Во что превращается код программы, написанной на Си, после компиляции?

Возьмем простой пример:

Создадим простое приложение средства MS Visual Studio типа Win32 с графическим интерфейсом.

Переведем режим проекта в стадию Release (для тех, кто к 4 курсу еще не научился этому, необходимо выбрать свойства проекта Properties->Configuration Manager->Configuration и перевести из режима Debug в режим Release).

Теперь все должно быть готово к применению.

Для начала посмотрим, во что превращается обычный вызов точки входа программы. Для этого поставим точку останова на функцию \_tWinMain (нажмем F9) в строке с названием функции.

Запустим приложение.

Выполнение программы должно прекратиться в поставленной нами точке. Теперь перейдем от отображения в виде Си к ассемблеру путем перехода к вкладке Disassembly (рис. 14 и 15).

Рис. 14. Исходный код программы

Рис. 15. Дизассемблированный вид программы

Попробуем подойти с практической точки зрения и посмотрим, во что превращаются некоторые стандартные конструкции языка Си.

Рассмотрим комплекс инструкций:

ULONG A, B;

A = GetTickCount();

B = 10;

if (A > B)

{

A--;

}

for (A = 0; A < 100; A++)

{

B++;

}

switch(A)

{

case 10:

B+=100;

break;

case 20:

A+=200;

break;

}

Примечание: если Вы захотите просмотреть это на реальном примере, то либо перейдите в режим DEBUG, либо отключите оптимизацию компилятора, так как в полученном вами исполняемом файле данная конструкция, скорее всего, не встретится, потому что компилятор посчитает ее неиспользуемой и такой, что не стоит включения в конечный исполняемый файл.

Итак

ULONG A, B;

A = GetTickCount();

00411B9E 8B F4 mov esi,esp

00411BA0 FF 15 1C C2 42 00 call dword ptr [\_\_imp\_\_GetTickCount@0 (42C21Ch)]

00411BA6 3B F4 cmp esi,esp

00411BA8 E8 72 F8 FF FF call @ILT+1050(\_\_RTC\_CheckEsp) (41141Fh)

00411BAD 89 45 F8 mov dword ptr [A],eax

 B = 10;

00411BB0 C7 45 EC 0A 00 00 00 mov dword ptr [B],0Ah

 if (A > B)

00411BB7 8B 45 F8 mov eax,dword ptr [A]

00411BBA 3B 45 EC cmp eax,dword ptr [B]

00411BBD 76 09 jbe WinMain+48h (411BC8h)

 {

 A--;

00411BBF 8B 45 F8 mov eax,dword ptr [A]

00411BC2 83 E8 01 sub eax,1

00411BC5 89 45 F8 mov dword ptr [A],eax

 }

 for (A = 0; A < 100; A++)

00411BC8 C7 45 F8 00 00 00 00 mov dword ptr [A],0

00411BCF EB 09 jmp WinMain+5Ah (411BDAh)

00411BD1 8B 45 F8 mov eax,dword ptr [A]

00411BD4 83 C0 01 add eax,1

00411BD7 89 45 F8 mov dword ptr [A],eax

00411BDA 83 7D F8 64 cmp dword ptr [A],64h

00411BDE 73 0B jae WinMain+6Bh (411BEBh)

 {

 B++;

00411BE0 8B 45 EC mov eax,dword ptr [B]

00411BE3 83 C0 01 add eax,1

00411BE6 89 45 EC mov dword ptr [B],eax

 }

00411BE9 EB E6 jmp WinMain+51h (411BD1h)

 switch(A)

00411BEB 8B 45 F8 mov eax,dword ptr [A]

00411BEE 89 85 F4 FE FF FF mov dword ptr [ebp-10Ch],eax

00411BF4 83 BD F4 FE FF FF 0A cmp dword ptr [ebp-10Ch],0Ah

00411BFB 74 0B je WinMain+88h (411C08h)

00411BFD 83 BD F4 FE FF FF 14 cmp dword ptr [ebp-10Ch],14h

00411C04 74 0D je WinMain+93h (411C13h)

00411C06 EB 16 jmp WinMain+9Eh (411C1Eh)

 {

 case 10:

 B+=100;

00411C08 8B 45 EC mov eax,dword ptr [B]

00411C0B 83 C0 64 add eax,64h

00411C0E 89 45 EC mov dword ptr [B],eax

 break;

00411C11 EB 0B jmp WinMain+9Eh (411C1Eh)

 case 20:

 A+=200;

00411C13 8B 45 F8 mov eax,dword ptr [A]

00411C16 05 C8 00 00 00 add eax,0C8h

00411C1B 89 45 F8 mov dword ptr [A],eax

 break;

 }

Итак, мы видим следующее: под конструкциями языка высокого уровня Си находится то, во что превращает компилятор исходный код программы.

НАСТОЯТЕЛЬНО РЕКОМЕНДУЕТСЯ ПРОВЕСТИ ОЗНАКОМЛЕНИЕ С ДАННЫМ ПРИМЕРОМ И УЯСНИТЬ КОНСТРУКЦИИ АССЕМБЛЕРА, ЭКВИВАЛЕНТНЫЕ КОМАНДАМ ЯЗЫКА ВЫСОКОГО УРОВНЯ. ПОЛУЧЕННЫЕ ЗНАНИЯ БУДУТ НЕОБХОДИМЫ ДЛЯ ВОССТАНОВЛЕНИЯ АЛГОРИТМА СОЗДАНИЯ КЛЮЧА ПОЛЬЗОВАТЕЛЯ.

Работа с OllyDbg

Рассмотрим работу с этим отладчиком на примере версии 1.1 без дополнительных плагинов.

Для этого запустим программу и откроем в ней notepad.exe:

#

Рис. 15. Общий вид OllyDbg

Слева перед нами содержится окно, в которое выводится ассемблерный код программы, справа – панель с состоянием регистров (по правому щелчку можно выбирать, какие дополнительные регистры показывать – MMX, 3dNow! и так далее), внизу – дамп стека и оперативной памяти.

В окне с ассемблерным листингом указано смещение, hex-код, непосредственно ассемблерная команда, ему соответствующая, и расширенный комментарий. В качестве комментария часто выводится такая полезная информация, как например, имя вызываемой API-функции.

Дважды кликнув по команде, можно редактировать ее и заменять своей. Обратите внимание на меню, которое появляется при правом клике мышью. Меню Breakpoint позволяет управлять точками останова: Toggle (F2) ставит брейкпоинт на выбранной команде, Conditional Breakpoint – останов при выполнении некоторого условия, Run To Selection – выполнить программу и остановиться на выбранной команде.

Меню Search for содержит такие незаменимые опции, как All Referenced Text Strings и All Intermodular Calls. Открыв пункт Search for All Referenced Text Strings, можно увидеть все строковые данные, используемые в программе, и быстро перейти в то место, где они упоминаются. Приведем практический пример: программа проверяет серийный номер и в случае неудачи сообщает «Your serial number is incorrect». Делаем поиск этой строки и осуществляем переход к тому месту, где вызывается соответствующий MessageBox; наверняка где-то недалеко идет проверка валидности серийного номера, которую можно либо пропатчить (например, забить теми же nop'aми) или попытаться разобраться в алгоритме и написать кейген. В меню пункт Search for All Intemodular Calls можно найти список всех API-функций, используемых в программе, и поставить на нужных из них точку останова. В предыдущем примере можно было бы поставить breakpoint на MessageBoxA и ввести неправильный серийный номер – программа бы остановилась как раз на том месте, где нам нужно.

Кроме этих пунктов, есть еще множество других не менее полезных. Предлагаем студенту ознакомиться с ними самостоятельно.

Ну и, конечно, важным является меню Debug: надеемся, что стандартные команды вроде Trace Into или Step Over ни у кого проблем не вызовут.

Прочитав до этого места, студент должен считается достаточно подготовленным для выполнения данной лабораторной работы.

## Практическая часть

## Задание. Исследовать предоставленную программу, обладающую защитой от своего несанкционированного использования и взломать защиту разными способами

В процессе выполнения лабораторной работы необходимо:

1. Выделить в исследуемой программе участок кода, выполняющий функцию принятия решения о корректности введенного пароля. Определить файл(ы), в котором(ых) хранится зашифрованный пароль.
2. Осуществить взлом установленной защиты, реализовав отключение защитного механизма, путем модификации функции принятия решения о корректности введенного пароля.
3. Выделить в программе участок кода, ответственный за формирование корректного пароля, соответствующего введенному имени пользователя. Исследовать данный код и формально записать алгоритм формирования корректного пароля. Используя код программы, ответственный за формирование правильного пароля, создать генератор паролей.
4. Осуществить взлом установленной защиты, используя некоторый пользовательский идентификатор (имя пользователя) и соответствующий ему корректный пароль, сформированный по найденному в п.3 алгоритму.
5. Выделить в исследуемой программе участок кода, выполняющий функцию принятия решения о превышении установленного предела запусков. Определить ключи реестра, в которых хранится счетчик запусков.
6. Осуществить взлом установленной защиты - либо подменой функции принятия решения о превышении установленного предела, либо путем изменения ключей реестра.

Для решения данных задач рекомендуется использовать следующие средства динамического и статического исследования ПО:

1. Interactive DisAssembler Pro v 4.51 и выше.
2. Hiew 6.81 и выше.
3. Встроенный отладчик IDE MS Visual Studio 2003/2005, OllyDbg или же SoftIce

# Рекомендуемая литература

1. Крис Касперски. Техника и философия хакерских атак. Записки мыщ’а. 2-е издание, переработанное и дополненное.
2. То же самое, только первое издание.
3. В.И. Юров. Assembler. Учебник для ВУЗов. (любое из изданий).
4. Крис Касперски. Образ мышления – дизассемблер IDA.
5. Р.А. Айрапетян. Отладчик SoftIce. Подробный справочник.