[Linefeed]

Для текстовых файлов задает символ переноса строки. Из-за того что в разных системах он разный, могут возникнуть некоторые проблемы с его интерпретацией. Большинство просмотрщиков автоматически определяют используемый формат. Точно так же поступает и HIEW (Auto). Однако если нужно, то можно перевести его и на ручной режим управления. Он понимает следующие значения — LF (OxA), CR (OxD), CRLF (OxODOA), но, к сожалению, неправильно интерпретирует (OxA OxD) и (OxA OxA). Впрочем, последнее достаточно редко встречается, чтобы вызвать какие-то проблемы. Отметим, что эта возможность впервые появилась только в версии 5.10.

[AutoCodeSize]

Автоматически определяет разрядность кода (16\32) в LX-файлах (префикс 'a' в строке статуса говорит, что активирован автоматический режим определения). Нет никакого смысла выключать эту возможность, хотя это включение по каким-то загадочным причинам все же предусмотрено автором.

[KbdFlush]

Управляет очисткой клавиатурного буфера перед вводом. По умолчанию включено. В противном случае в окнах ввода информации частенько бы появлялся мусор, оставшийся от предыдущих нажатий клавиш. Поэтому никак невозможно представить себе ситуацию, в которой это было бы полезно. Словом, отключать эту возможность можно разве что ради эксперимента.

[XlatTableIndex]

Задает индекс таблицы перекодировки в файле hiew.xit, выбираемый по умолчанию. При этом '0' трактуется как отсутствие таблицы перекодировки, или, в терминологии HIEW-a, 'AS IS'.

tFlistSort] Задает критерий сортировки файлов в Навигаторе по умолчанию.

[FlistSortReverse]

Инвертировать критерий сортировки. Т.е. по умолчанию он задается по возрастанию параметра сортировки. Если это кажется неудобным, то процесс можно и обратить.

[FlistShowHidden]

Показывать или нет скрытые файлы в Навигаторе. По умолчанию такие файлы не отображаются. Странно, однако. Рекомендую установить этот параметр в 'On'.

[NextFileSaveState]

Сохранять текущее состояние при переключении между файлами. По умолчанию выключено, что мне категорически не нравится. Рекомендую активировать сей механизм — это сбережет немало времени и нервов.

[SaveFileAtExit]

Записывать состояние файла по выходу. По умолчанию выключено (?!), что мне абсолютно непонятно, — если учесть, что выход происходит без всякого предупреждения пользователя и на Esc и на FIO, так что ложные нажатия происходят довольно часто. Непременно включить этот механизм.

[ScanStep]

Шаг по умолчанию при поиске ссылки (F6 Reference) или команд (F7,F7). По умолчанию имеет значение 'Command', но лучше проиграть в скорости, чем в надежности, и рекомендуется устанавливать шаг поиска в байт ('Byte').

[Savefile]

Задает имя и путь к файлу сохранения состояния. По умолчанию HIEW создает файл 'hiew.sav' в текущей директории, но это можно изменить. Это бывает полезно, например, когда на текущий диск записать нельзя (ключевая дискета, защищенная от записи, CD-ROM)...

Цвета

Наконец-то HIEW стал поддерживать цветовую раскладку! Теперь каждый может настроить ее под свой вкус. Допустим, некто большой поклонник "зеленым по черному" и все используемые программы настраивает именно так.

Подробно описывать кодировку цветов не имеет смысла, она тривиальна. К тому же это гораздо удобнее делать специальной утилитой, (например, похожей на QVIEW) чем вручную.

Структура этой секции hiew.ini ясна из приведенного фрагмента:

ColorMain = Ox1B ; основной цвет ColorCurrent = 0x71 ; текущий байт

HIEW.XLT

Этот файл служит для перекодировки символов. Может содержать до 15 кодировок, но в распространяемом автором варианте содержит только две — Win 1251 и K018R. Имеет довольно витиеватую структуру, и для его создания/просмотра/редактирования не помешает написать хорошую утилиту. Автор в сопутствующей документации описывает структуру как:

```
// "HiewXlaf',0
// 0x05 // 0x40
typedef  struct( BYTE  sign[ 9 ], unused [ 5 ], versionMajor, versionMinor; }XLAT HEADER;typedef struct(
BYTE title [ 16 ],     // заголовок
tableOutI 256 ],          // для вывода
tableln[ 256 ], // для ввода
tableUpper[ 256 ];      // для игнорирования регистров в поиске
}XLAT;
```

Вряд ли это вызовет какие-нибудь вопросы. Структура полностью понятна и удобна в обращении. Ввод-вывод разделены, что приятно. Аналогично обстоит дело и с переводом регистра. Разумеется, HIEW не может знать, как расположены символы в произвольной кодировке, поэтому регистр автоматом не переводит.

Жаль только, что в комплекте с HIEW-OM нет утилиты для работы с этим файлом. Откомпилировать его можно — положим, препроцессором, но вот декомпилировать... для декомпиляции потребуется написать специальную программу. Если вы, допустим, решитесь добавить поддержку ISO, то потребуется сначала декомпилировать существующий файл, внести исправления и повторить

опять.

Заметим, что hiew.xit может отсутствовать. В этом случае поддерживается единственная кодировка по умолчанию DOS, или, в терминологии HIEW-a, 'AS IS'.

BoundsCheck OverView

"Покажите мне пример совершенно гладко идущей рутины и я найду вам того, кто прикрывает ошибки." Ф. Херберт. "Дом Глав Дюны".

BoundsChecker фирмы NuMega (далее просто BC) — это инструмент, прежде всего нацеленный на поиск ошибок в вашем собственном программном обеспечении. Однако он подходит и для исследования взаимодействия приложений с операционной системой или другими приложениями, поступившими в ваше распоряжение без исходных кодов.

Что представляет собой BC? Отладчик или дизассемблер? Ни то ни другое. Это \Уш32-шпион. Обычные шпионы (типа Spyxx из MS VC) перехватывают только сообщения, которыми приложение обменивается с окном. Учитывая, что архитектура Windows фактически полностью построена на сообщениях, перехват последних несет достаточно полную информацию о происходящих событиях. Полную, но увы, не исчерпывающую. Множество функций вообще не генерируют сообщений (например, выделение памяти, чтение из файла или реестра).

Между тем именно эта информация имеет первостепенную важность при анализе приложения. MicroSoft, однако, отделывается молчанием по этому поводу. Написав блестящего шпиона сообщений, она вряд ли даже помышляет об API-шпионе. Это наводит многих на мысль, что API-шпионы штука не простая и их написание доступно не каждому.

На самом деле это не такое сложное занятие, и Мэтт Питрек в своей книге подробно описывает этот процесс во всех тонкостях, прилагая на дискете в качестве иллюстрации довольно неплохого шпиона, занимающего в упакованном виде чуть больше 7 килобайт.

BC же — это десятки мегабайт сложного кода, который ненамного хитрее созданного Питтреком; он разве что покрывает больший круг задач. По крайней мере, поработав с версией 5.0, можно прийти к выводу, что нет ничего лучше собственных шпионов, специализированных применительно к интересующему вас кругу задач. И все же BC не совсем то, что нужно хакеру. Очень много в нем недостает: в частности, нет никакой системы навигации по его протоколам. Нет и никакого фильтра функций, так что вам предстоит тяжелая археологическая работа: отыскивать нужные имена среди бесконечных вызовов GetMessage/Tran-slateMessage/DispathMessage и им подобных. Так же обстоит дело и с сообщениями — километрами листингов, в основном не представляющих для хакера в данный момент ничего интересного. Spyxx имеет гибкую систему фильтров, которая сокращает протоколы и заметно ускоряет анализирующий выходные файлы BC. Разумеется, написание его целиком ляжет на ваши плечи и может оказаться не слишком простой задачей (особенно если требуется не просто фильтр, а очень хороший фильтр — с гибкой системой поиска). Однако время, затраченное на разработку внешнего фильтра с лихвой окупится ускорением анализа программ.

К недостаткам BC можно отнести также невозможность анализа самозагружаемых модулей, крайнюю неустойчивость к приложениям, пытающимся "прибить" BC, а так же отсутствие поддержки 16-разрядных приложений и DLL. Последнее особенно неприятно. Но ничего не поделаешь. Пишите собственных шпионов или довольствуйтесь тем, что есть сегодня на рынке.

Вышесказанное не означает, что я призываю отказаться от BC. Вовсе нет! Это действительно очень мощный инструмент, в настоящее время не имеющий аналогов. Но нужно четко представлять, для решения каких задач он полезен. Часто более быстрый путь — использование дизассемблера или отладчика.

Не стоит пытаться "перетянуть" одеяло и решать BC-ом те задачи, для которых он, мягко выражаясь, не совсем предназначен. Строго говоря, BC даже и не может работать в одиночку. Его назначение — показать вызовы API-функций и сообщений в изучаемом приложении. Для чего может понадобиться эта информация?

Рассмотрим случай, когда программа, считывающая строку из окна редактирования, не вызывает ни GetWindowText, ни GetDIgIternText, и даже не посылает сообщения WM_GETTEXT. Чтобы понять, как текст все же попадает в буфер, надо проанализировать манипулирующий с ним код. Т.е. пройтись дизассемблером по нужному участку кода. Но тут вся проблема в том, что мы не знаем, какой код "нужный", и в поисках последнего рискуем потерять уйму времени.

В этом случае BC, показывая все вызовы API и связанных DLL, наверняка покажет и те, которыми приложение считывает текст. Это может быть посимвольное чтение клавиш или еще что-нибудь. Конечно, будет большой проблемой найти эти функции и в протоколе (так как он наверняка будет иметь немалый размер, сравнимый, может быть, даже с листингом дизассемблера) — при изучении последнего придется порядком попотеть.

BC поможет также при исследовании приложений на предмет недокументированных функций и возможностей. Или документированных, но плохо понятных. Например, если работа Менеджера Задач вам кажется сплошной загадкой, а дизассемблировать его, естественно, не хватает времени и терпения, можно запустить API-шпиона и посмотреть, какие функции приложение вызывает.

Заметим, что те же функции можно узнать простым просмотром таблицы импорта, но при этом останется загадкой порядок их вызова и взаимодействие друг с другом. Вот тут и следует использовать шпиона. В противном случае более эффективными окажутся отладчик + дизассемблер. Разумеется, BC также незаменим и при исправлении ошибок в чужих приложениях без исходных текстов (он, собственного говоря, для этого и разрабатывался).

Быстрый старт

"— *Существуют во Вселенной попросы, на которые нет ответов."* Ф. Херберт. "Мессия Дюны".

BC необычайно прост в управлении. NuMega специально отмечает это в прилагаемой документации. Иначе говоря, вас сильно ограничили в гибкости и возможностях контроля над приложением. Все за вас делает BC!

Воистину непостижим такой подход на фоне Soft-Ice, который содержит весьма привлекательный для профессионала интерфейс. Но почему-то применительно к BC фирма-разработчик пошла другим путем, чем явно огорчила многих своих поклонников.

Остается использовать то, что дают. Запустим BC и попробуем открыть какой-нибудь файл для анализа. Напомню, что поддерживаются только 32-разрядные \Уш32-файлы. Даже сегодня во время победного шествия Win32 API это ограничение все еще существенно, и существует немало 16-разрядных приложений даже среди новых разработок.

Чтобы согласовать наши действия, предлагаю вам остановить свой выбор на примере BUG 1, находящемся в комплекте поставки BC. При этом последний автоматически откроет окно "Program Transcript". В нем на протяжении всего сеанса работы будут отображаться такие события, как загрузка различных DLL, а также выводиться все отладочные сообщения. Это нас будет интересовать в последнюю очередь.

Если открыть руководство, то можно прочесть рекомендацию кликнуть по кнопке "run" (эдакая соблазнительная стрелочка, направленная вниз) и запустить программу. Но не будем спешить и полагаться только на руководство. В противном случае BC будет мирно крутиться в фоне, ожидая ошибок или аварийных ситуаций. Естественно, что в изучаемом приложении таковых скорее всего не окажется и финальный рапорт будет пустым.

Это кстати, самая распространенная ошибка начинающих пользователей BC. За неудачной попыткой следует разочарование или вопрос — а где же обещанные вызовы API?

На самом деле, как уже отмечалось, это и есть нормальная работа BC, который в первую очередь предназначен для предотвращения ошибок и их локализации. Зададимся вопросом — а как это он делает? Полистав документацию, мы узнаем, что BC перехватывает практически все ключевые функции и проверяет корректность передаваемых им параметров. В противном случае он выводит в окно сообщение об ошибке.

Обратите внимание: BC перехватывает функции, но не отмечает этот факт в протоколе, если вызов функции с его точки зрения корректен. Быть может, есть какой-то способ отображать все события вне зависимости от аварийности ситуации? Действительно, NuMega предусмотрела такую возможность. Конечно, среднестатистическому пользователю она не нужна и поэтому по умолчанию выключена. Чтобы исправить это, заглянем в установки программы (Program/Setting). Выберем закладку "Event Reporting". Появится выбор из следующих пунктов:

Collect and report program event data

Report messages

Report pointer data for API

Report Hooks

Но последние три пункта будут недоступны, пока не установлена галочка "Collect and report program event data". Выбор остальных зависит от выбранной вами цели и исследуемой программы. Для перехвата функций API нужно установить "Report pointer data for API", а также остальные — для перехвата сообщений или хуков.

Если теперь снова перейти на первую закладку "Error Detection", то будет возможность сохранить текущие настройки значениями по умолчанию. Это и в самом деле приятная возможность, особенно если планируется использовать BC в основном для анализа чужих программ, а не по прямому назначению.

Вот теперь-то наконец мы нажмем "run". Если у вас меньше 128 мегабайт оперативной памяти, то вы услышите интенсивный звук прыгающих головок винчестера и откроется еще одно окно, которое показано ниже.

При этом в окне отображения событий появится единственный поток приложения (непопятно почему изображенный катушкой с зелеными нитками), а ниже — встретившаяся ошибка (умышленно допущенная разработчиками для демонстрации возможностей BC), но сейчас не будем акцентировать на ней внимания.

Винчестер продолжает интенсивно шуршать, и мы, заботясь о свободном пространстве на своем диске, поспешим прервать выполнение контрольного приложения нажатием кнопки "stop" или закрытием его окна.

И где же ожидаемый нами протокол? Почему он по-прежнему выглядит пустым? Куда подевались все функции API, ведь наши ушки явно слышали, что в протокол записывалось что-то "тяжелое"! Не будем волноваться. На самом деле все сработало успешно, просто BC по умолчанию не показывает в окне событий ничего, кроме ошибок и потоков.

Нажатием правой кнопки мыши, вызовем контекстное меню и установим галочку над пунктом "Show All Events". А затем "Expand All". Если все было сделано правильно, то в окне появится длинный перечень вызываемых функций.

Окно отображения событий Окно стека
Окно исходного текста

Прокрутим его немного вниз, пока не встретим вызов функции CreateWindo-wExA. Щелкнув по нему, мы увидим в окне стека, расположенном справа, все передаваемые параметры в удобочитаемом виде. Из этой информации можно узнать много интересного. Но важно уже то, что мы знаем, какой именно функцией было создано окно.

Теперь, если потребуется перехватить момент создания окна, у нас уже не возникнет вопрос, на какую функцию ставить Breakpoint. Вряд ли нужно говорить, что приложение может создать NagScreen множеством способов и вручную перебрать их все будет очень затруднительно. BC же позволяет нам заглянуть "под капот" выполняемой программы, чтобы узнать, какие именно функции оно для этого вызывает.

Подключение нестандартных DLL

"...использовать грубую силу — значит оказаться во власти гораздо более могущественных сил"

Ф. Херберт. "Дюна".

BC реализует далеко не лучшую схему перехвата вызовов API- и DLL-функций. Это отличает его, например от продвинутых шпионов, перехватывающих на лету все функции загруженных приложением модулей. Причем последние даже не обязательно должны быть DLL-модулями. Например, IDA поддерживает пла-гины, которые представляют собой PE- или LE-файлы, экспортирующие определенные функции. А IDA в свою очередь предоставляет им собственный API.

Для исследования механизма взаимодействия последней и отладки своих собственных плагинов необходимо написать шпиона, отслеживающего протекающие "под капотом" процессы. BC с такой задачей справится не может. Впрочем, это понятно и нареканий не вызывает. Разумеется, редкий инструмент общего применения идеально подойдет для произвольной узкоспециализированной задачи.

BC будет бессилен и в том случае, если защитный механизм помещен в DLL (о которой парни из NuMega, естественно, не имели никакого представления) и интенсивно взаимодействует с изучаемым приложением. Задачей взломщика будет в первую очередь понять, какие функции выполняет защитная DLL и можно ли каким-либо образом эмулировать последние.

Если бы научить BC "видеть" эту DLL, то никаких проблем бы не было и мы получили бы интересный протокол, раскрывающий все секреты защиты. К счастью, NuMega

предусмотрела этот момент и снабдила пользователей инструментарием для поддержки "их собственных DLL". Понятно, что BC не может с уверенностью сказать, ваша это DLL или еще чья-то. При перехвате BC никак не изменит исследуемую DLL, а поэтому это действие не противоречит нашему законодательству.

Чтобы понять, как происходит подключение нестандартных DLL, нужно представить себе механизм перехвата экспортируемых ими функций. К сожалению, это очень емкая тема, для которой требуется отдельная книга; кроме того, она уже многократно и исчерпывающе освещалась. Поэтому ограничусь двумя словами. Для каждой перехватываемой DLL нужно построить определенную заглушку. BC подменяет вызовы оригинальной DLL на "свои". При этом он полностью контролирует управление и передаваемые параметры. Разумеется, для этого должна существовать специальная база данных, которая содержит типы и допустимые параметры. Впрочем, последнее нас интересует меньше всего, ибо нам необходимо отметить сам факт вызова какой-то функции, и только. Это облегчит нам задачу написания "заглушки".

В нашем распоряжении большие возможности, так как можно внедрить любой мыслимый и немыслимый код, делающий практически любые операции и могущий содержать даже наш собственный интегрированный отладчик — или, по крайней мере, вызов внешнего, если нам не хватает времени и квалификации.

Однако написание заглушки — очень трудоемкий процесс, требующий определенных навыков и знаний. Впрочем, специалисты (а именно такой специалист подразумевается под словом "хакер") не боятся этих трудностей. Была бы только документация и хотя бы краткое описание. В том-то и беда, что его нет! Веселенькое начало нам обеспечила NuMega! Как же быть? Остается использовать мастера, который сгенерирует за вас весь код автоматически, на C++.

Работу мастера описывать я не буду. Скажу только, что она проста, и этот мастер, как любой другой, потребует от вас минимальных умственных и физических усилий. Разумеется, возможности выбора окажутся очень ограниченными, но, как бы то ни было, на финальном пункте появятся три файла — *.cpp, *.h и *rnak. При этом откомпилировать это MS VC можно только с ручной правкой некоторых мест, что не приносит абсолютно никакого удовольствия и тормозит весь процесс.

По моему мнению пользоваться мастером стоит только ради пары-тройки примеров, а далее вы сможете писать свои собственные примеры уже без посторонней помощи. Предупреждаю: для некоторых, ранее не имевших дела с подобными вещами, задача окажется достаточно трудной, и для достижения поставленной цели потребуется много усилий, особенно если учесть, что в коде, сгенерированном мастером, очень много лишнего, мешающего пониманию сути.

Пункты меню

"Вся жизнь его в этот момент представилась ему веткой, дрожащей после взлета птицы, а эта птица была — возможность. Свободная воля."

Ф. Херберт. "Мессия Дюны".

Я уже предупреждал, что BC имеет крайне скудный для профессионала интерфейс. Если вы хотите всерьез заняться изучением программ с помощью перехвата API-функций, то лучше все же не пожалеть времени и написать необходимый инструментарий самостоятельно. Однако большинство программистов слишком лениво или занято, и поэтому BC стал стандартом "де-факто".

Кому-то такая критика может показаться несправедливой. Быть может, это слишком сурово по отношению к этой великолепной утилите? Кто знает... когда конкуренты спят и не предлагают ничего лучшего, то на безрыбье и рак рыба. Хотя есть еще и WinScope, написанный фирмой Periscope, но он почему-то не получил широкого распространения.

Меню "файл" типично для Windows-приложений. Пункты Open/Close/Save As комментариев не требуют. Заострим внимание только на одном моменте: "сохранить" подразумевает сохранить текущее состояние и перехваченные вызовы в файл. К

сожалению, используется не текстовый, а внутренний формат BC, а это означает, что написание собственного навигатора усложняется. Для этого потребуется проделать кропотливую работу по изучению сложной, а местами и запутанной структуры файла.

С другой стороны, подобный анализ — очень хорошая практика, которая не раз и не два пригодится в дальнейшем, а также обеспечит хорошие навыки работы с IDA и Soft-Ice, да и просто поможет оценить общие концепции. Но я отклонился от темы.

"Print" и "Print PreView" позволят распечатать или хотя бы показать на экране протокол в более удобочитаемом виде. К сожалению, для его распечаты-вания потребуется тонны бумаги (соответственно, литры чернил в случае струйного принтера), что вряд ли оправдано. Особенно если вспомнить про сегодняшний кризис и стоимость расходных материалов... Непосредственная печать в файл невозможна, поскольку BC представляет все данные в графическом, а не текстовом формате.

Меню "Edit" содержит традиционную команду "Find" (Ctrl-F). Вызов ее приводит к появлению стандартного диалога поиска, которое вроде бы не поддерживает ни шаблонного, ни мультистрокового поиска. Однако при близком рассмотрении справа от окна ввода обнаруживается маленькая прямоугольная кнопочка со стрелкой, обращенной вправо. Нажатие ее раскрывает подменю поиска, которое притягивает к себе с первого взгляда.

Error
Resource, memory or interface leak
API or OLE method call
API or OLE method return
Windows of Dialog Message
HOOK
Comment
Thread start of switch

Большая часть пунктов нас, конечно, интересовать не будет. Например, поиск утечки памяти (или ресурсов), столь полезный для разработчика в уже отлаженной программе, просто маловероятен и, кроме того, вряд ли каким-то образом связан с защитным механизмом.

Поиск вызовов API был бы полезен, если бы была хоть какая-то возможность указывать сложную маску. Фактически их так много, что следующий вызов нетрудно найти и ручным поиском. Более того, он скорее всего, окажется в пределах окна событий, так что не потребуется даже прокрутки последнего.

Фактически единственным используемым пунктом будет 'Windows of Dialog Message' — так как не всегда легко найти сообщения, относящиеся к окну (диалогу) среди множества малоинформативных вызовов функций API.

Поиск переключения контекста потока очень полезен при отладке многопоточных приложений. Однако такие защиты пока не получили распространения. Впрочем, отдельные экземпляры существуют и устойчиво сопротивляются всяче-

ским попыткам их взломать. В этом случае BC будет, вероятно, единственным средством, позволяющим быстро понять, что же все-таки происходит в отлаживаемом приложении.

Приятно, что BC поддерживает поиск не только вперед, но и назад. Это весьма упрощает навигацию и значительно экономит драгоценное время разработчика.

Заметим, что окно поиска вынесено и в панель инструментов. Как и следует ожидать, оно поддерживает историю ввода; пара кнопок, расположенных рядом, задает направление поиска — вперед и назад. Но фильтр можно вызвать только через меню. Впрочем, редкое его использование не позволяет считать отсутствие его в панели инструментов поводом для критики или возмущения. Даже наоборот — панель меньше загромождается ненужными кнопками.

Меню VIEW начинается с пункта 'Event Summary'. Это своего рода и "легенда" всех условных обозначений, но в то же время — статистика вызовов функций API, оконных сообщений и всего остального в виде сводной таблицы. Эта бесполезная (но очень любопытная) информация не несет в себе ничего, что можно было бы использовать для анализа программы. Ну какой прок знать, что приложение вызвало 2562 функций API и приняло/передало 115 сообщений?

Следующие три пункта:

Show Error and leak only (показывать только ошибки и утечки)

Show All Events (показывать все события)

Show Error and Specific event (показывать ошибки и выбранные события) Specific event нам уже должны быть хорошо знакомы по "быстрому старту". Обычно используется "Show All Events" или "Show Error and Specific event". При этом можно выбрать произвольный набор отображаемых событий. К несчастью для хакера, крайне грубый и непрактичный.

Все, что можно сделать, — это разделить вызовы OLE и API. Последнее, конечно, полезно, но все же жалко, что нельзя разделить сами вызовы API по каким-либо группам или хотя бы замаскировать некоторые из них. Как уже было отмечено, протокол будет в основном забит вызовами API, крутящимися в цикле выборки сообщений, и нет никакой возможности "отфильтровать" эти функции от других.

Установка флажка "Arguments" приводит к тому, что в круглых скобках будут указываться аргументы каждой из отображаемых функций. Это в самом деле очень удобно. При этом BC распознает строки и представляет их в символьном виде. Поэтому мы имеем все шансы встретить введенный пароль и найти функции, манипулирующие им. То же верно и по отношению к другим типам переменных. Если при этом последняя представляет собой какую-то сложную структуру, то она будет отображена в развернутом виде в окне стека, расположенном справа.

По умолчанию этот флаг опущен. Настоятельно рекомендуется его взвести и больше никогда не опускать. То же самое можно сказать и о 'Sequence number'— последовательном номере функции, отображаемом слева. Это повышает эстетическое восприятие текста, а также дает понять, в каком месте протокола вы в данный момент находитесь (и помогает собрать рассыпавшиеся листы распечатки, если вы все же решитесь его распечатать).

Дальше идут два пункта, актуальные только для многопоточных приложений. Они позволяют быст переключаться между несколькими потоками.

Следующие несколько пунктов меню управления позволяют свернуть или развернуть все отобража ветви. Учитывая, что с последним неплохо справляется и мышь, трудно понять, чем вызвано дублир их в меню, причем без ассоциации с "горячими клавишами". Местами интерфейс BC и вправду плох продуман.

Меню "run" содержит только один интересный пункт "Setting", который и будет подробно рассмотр ниже. Если кликнуть по нему мышкой, то появится окно с несколькими закладками.

Первая из них относится непосредственно к выбору уровня строгости контроля ошибок (в том числ ручной его настройке), но к анализу заведомо устойчиво работающих программ никакого отношения имеет, а поэтому рассматриваться в данном описании не будет.

Гораздо более интересна следующая закладка — 'Events reporting', уже рассмотренная выше. Думае что уже нечего добавить, разве лишь напомнить еще раз о необходимости взвести нужные флажки: в противном случае просто ничего не будет работать.

"Program Info" позволяет задавать командную строку приложения (что уже давно не актуально для Windows-приложений, но, может быть, когда-нибудь и пригодится) и рабочую директорию (по умол выбирается текущая); а также указать путь к исходным текстам, которых скорее всего просто в наше распоряжении не окажется (а потому никакого пути и указывать не придется).

А вот 'Error Supressions'— очень полезная и актуальная функция. Помните, я говорил, что нет никак возможности задать фильтр для произвольных и мешающих функций? Так вот: я вас обманул. Эта возможность есть, правда, в очень грубом варианте ее реализации. Можно вообще отключить перехв интересующих нас в данный момент функций. Но это не самый лучший выход. Если в процессе изуч рапорта потребуется информация о "заблокированных" функциях, она уже никак не может быть пол разве что полным ре-анализом, что удовольствия не приносит.

Но, к сожалению, выбирать не приходится. Обычно мы вынуждены действовать по следующему алгоритму. Первым делом вносятся в "черный список" все функции, работающие с выборкой сообще из цикла. Запускаем приложение и смотрим в протокол. Если последний по-прежнему забит ненужн малоинформативными функциями, то добавляем их к "черному списку" и повторяем анализ вновь. И до тех пор, пока рапорт не окажется хотя бы минимально читабельным.

Описанный способ иногда приводит к тупику: когда на первый взгляд бесполезная (а в действитель ключевая) функция заносится в "черный список" и выпадает из нашего поля зрения. Но ничего лучш предложить не могу.

Как сломать TRIAL.COM

"*— Я говорю людям: смотрите! У меня есть руки! Но мне говорят: "А что такое руки?"*

Ф. Херберт. "Дюна".

Однажды вздумалось мне создать свою группу. Мысль, естественно, бредовая, но с другой стороны заманчивая. Многие проекты просто невозможно реализовать в одиночку. Группа же из пяти-десяти профессионалов уже способна написать хотя бы свою операционную систему или полный эмулятор включая и защищенный режим.

Одним из способов определить профессионализм человека — подсунуть ему crack_me и посмотрет он его взломает. В последнее время, кстати, хороших . crack_me стало очень мало. Обычно закладыв какой-нибудь математический алгоритм, решение которого требуется найти. Или, что еще хуже, про зашифровывают достаточно криптостойким алгоритмом — так, чтобы полный анализ был не возмож Утомительные часы перебора (а то и дни) — кому они интересны?

Я попытался (и не без успеха) предложить нетривиальную задачу, рассчитанную на нестандартное мышление, при этом предоставив неограниченную свободу творчества. Увы, я был разочарован. Ник и не приблизился к решению. Самое парадоксальное, что задачка была очень простой и... А, впрочем зачем "и" — все равно никто не решил.

- Crack Me by [KPNC] ',ODh,Oah
- Если вы сломаете данный crack me, то получите членство в группе [KPNC]
- Под "сломать" понимается узнать, на кого зарегистрирован demo_def: точнее,
- расшифровать его. Там содержится имя экс-президента группы. Скажу сразу:
- оно не мое, поэтому забудьте про атаку по открытому тексту. Скажу больше

- для расшифровки нужно найти ключ длиной в шесть байт. Два из них –
- OxFO 0x83. Остается четыре. Ну что — найдете? Аль нет? Взломать будет
- непросто — эти шесть байт просто отсутствуют в ключе, (см. сам ключ)
- Есть два независимых метода взлома, так что дерзайте...
- [KPNC@ID.ru]

Ниже будет дано подробное решение. Быть может, оно кого-то рассмешит. Но... действительно, зад
проста (см. trial.corn). Условие, конечно, сформулировано нечетко, и прежде всего надо было опреде
что собственно делать? Достоверно было известно только то, что в файле demo_def содержится имя
президента группы (и успел же он появиться!). И его надобно найти. Рассмотрим внимательнее этот

```
00000000: BA 01 14 00-5B 4B 50 4E-43 5D 77 77-B4 01 10 00 Цхх [KPNC]ww-lxx
00000010: 42 02 C3 01-4C 01 04 00-9F 01 EE 00-55 01 10 00 BxxxЬxxxЯхюхЦхх
00000020: 24 04 CD 01-EA 00 82 02-48 02 03 77-11 83 17 33 $x=хъхВхНххыхГхЗ
00000030: 84 02 82 02-6C 03      -         -         ДхВх1х
```

Ясно, что [KPNC] — это никак не искомое имя. Тогда... посмотрим как работает расшифровщик:

```
00000061: BF9701        mov     di, 00097 ; Приемник
……..
0000002E: AD            lodsw           ; Читаем слово шифротекста<—1 0000002F: 33DB          xor
bx,bx   ; BX == NULL
00000031: 86E3          xchg    ah,bl   ; BL == HiByte(AX)
00000033: F6F3          div     Ы       ; AL === AH/ ~AL
00000035: AA            stosb           ; Пишем расшифрованный байт 00000036: E2F6          loop
00000002E; —(1)—1
00000038: E85COO        call    000000097; Вызываем расшифрованный ключ 0000003B: СЗ
; Возврат.
```

Э, да шифровка-то тривиальна. И все бы коту масленица, да разработчик упоминал о каких-то шест
"слизанных" байтах, в которых и "зарыта собака". Действительно, в demo_def шесть нулевых байт! И
старшие, т.е. делители. Ясно, что x/O смысла не имеет, и программа просто аварийно завершается.
Разумеется, пропавшие байты по заверениям автора не относятся собственно к имени, да и не могли
относиться, так как иначе было бы попросту нечестно. Хотя, в любом случае это не важно — уж сли
беспорядочно они разбросаны по тексту...

Давайте подумаем: что можно выжать из этого файла и какую тактику нам применить? Конечно же
открытому тексту! А почему нет? Мы же знаем очень много об этом коде, который выводит сообще
Места хватит только на то, чтобы вывести его через функцию f.9\int 21, или, во всяком случае, через
Хотя f.6 откинем — рассмотрим последнее слово шифротекста — OxЗбС. Расшифруем его — OxбС
== 0x36 == '$'. Опля! Кой-что интересное! Не так ли, господа? Пойдем дальше, точнее, вернемся наз
расшифруем еще два символа — 0x82 \ 2 == 0x41 == 'A' и 0x84 \ 2 == 0x42 == *'B'. Итак,* экс-президе
зовут *BA. А дальше нам путь преграждает любопытнейшая комбинация 0x17 \ 0x33 == 0? Запахло
нечистым... 03\77;11\83; 17\ 33 целых три символа шифротекста, равные нулю, а в сумме шесть байт
их сложить с теми загадочными нулями, то... в общем многовато получится!

82\ 02 == 0x41 'A' и 48\ 02= 0x24 = '$' как-то непонятно выглядит. Однако так или иначе коду,
выводящему это безобразие, отводится все меньше и меньше места, как раз ровно столько, сколько х
на f.9, — но в таком случае мы имеем открытый текст, т.е. здоровую дыру, в которую можно пролез
Заметим, что вызов int 0x21 трудно замаскировать. И точно, мы видим...

```
00000000: BA 01 14 00-5B 4B 50 4E-43 5D 77 77-B4 01 10 00 UXXX[KPNC]WW-)XX
00000010: 42 02 C3 01-4C 01 04 00-9F 01 EE 00-55 01 10 00 Bxl-xLxxxAxioxUxx
00000020: 24 04 CD 01-EA 00 82 02-48 02 03 77-11 83 17 33 $x=хъхВхНххчхГхЗ
00000030: 84 02 82 02-6C 03      -         -         ДхВх1х
```

Хотя, конечно, автор мог использовать call на тело своей программы, где стоит int 21 (OxE8), но Ox
коде не наблюдается, а OxCD ('=') есть. А что есть еще? Есть 0x21 (offset 0x10), но почему-то стоящи
далеко от OxCD и OxB4 есть (mov ah,9), ну и 0x9 можно найти (offset 0x20, 0x24\0x4). Есть также M
DX, ? (OxBA) и его старший байт — операнды (0x1 — 0x77 \0x77).

Ситуация становится все более загадочной и непонятной — весь код присутствует, но странным об
перемешан. Выходит, мы столкнулись с шифром перестановки. Попробуем найти ключ. Вполне возм

что он генерируется всего на основе 6 байт. Это звучит вполне правдоподобно. При этом нулевые ба
могут быть просто не значащими и их можно отбросить. Но как найти ключ?

Пусть нам с некоторой достоверностью известен фрагмент оригинального текста, тогда можно проследить логику перемешивания байт. Допустим, OxCD и 0x21. Их отделяют всего 0x16 байт, или точнее, OxEA в знаковом представлении.

CD OI-EA оо

Поразительно! Такое совпадение! Выходит, ключ содержится в файле! Ну а как пара mov ah,9 (OxB 0x9)? Их отделяют 0x10 байт. И снова

в4 0110 оо

Итак, кажется, что-то прояснилось. Логика шифрования понятна. Можно хоть сейчас садиться и пи собственный дешифровщик. Ноль обозначает, что следующий расшифровываемый символ находитс (где п — значение младшего байта) расстоянии от текущего. Любопытный прием. Криптор может "блохой" прыгать по файлу, собирая разбросанные байты.

ОК. Это мы выяснили. Теперь можно расшифровать demo_def, "выцарапать" из него имя и зашифр снова. Но не будем торопиться. Это была только первая самая легкая стадия проверки на знание осн криптоанализа. (Примитивнейшая атака по открытому тексту, что может быть еще проще?) Поразит но никто этого так и не решил! Печально, очень печально.

Ну да ладно, вторая стадия — на сообразительность. Чтобы trial.corn вывел это имя на экран, необх всего лишь изменить его расшифровщик. Это нетрудно сделать, если обратить внимание на следую строки:

00000156: E81600 call 00000016F 00000159: 49 dec ex 0000015A: mov ah,03F ;"?" 0000015C: CD21 int 021

(см. сам файл). Т.е. можно записать ключ по любому адресу в файле! И кое-что это подтверждает!

00000166: 803C90 cmp b, [si],090 ;"P" 00000169: 7403 je 0000016 ---- (з) 0000016B: E8COFF call 00000012E -------- (4) 0000016E: СЗ ret

Смысл этих строк такой — если первый байт ключа '90' == NOP, то он затирает trial.corn и тот, даж делая попытки его расшифровать, переходит к другому ключу!

Так, про недостающие шесть байт —

00000002: 98 cbw 00000003: 83C406 add sp,006 00000006: 03FO add si,ax

их нужно вписать в обработчик int 0x0. При этом он будет корректно работать.

Примеры реальных взломов

В настоящей книге все атаки рассматривались исключительно на примерах, специально написанны демонстрации того или иного алгоритма программ 'CrackMe'. При этом многие из них были слишком искусственными и далекими от реальных защитных механизмов. Это было удобно для изложения материала, но не отражало реальных существующих защит.

Поэтому я решил включить в приложения некоторые примеры реальных взломов. Все эти програм широко распространены и отражают средний уровень защиты условно-бесплатных программ. Замет что он существенно ниже, чем многие из предложенных в книге реализаций.

Напоминаю, что взлом в той или иной мере конфликтует с российским и международным законодательством. Поэтому необходимо помнить, что взлом не освобождает от регистрации и може использован только в образовательных целях, но никак не для получения какой-либо выгоды. В этом случае конфликтов с законодательством не возникнет.

Subj: Ломать нечего?

В одной конференции некогда мне встретился следующий диалог трех молодых людей (в просторе ламеров).

SP> Аидстест поковыряй :)) Он такой извращенный, оказывается... Че-то там в SP> памяти елозит, расшифровывается... Боится, что его поганую рекламу SP> повыкидывают:))

AL> Гы, У нас преп на кафедре развлекается: патчит aidstest так, что у AL> того реклама вверх ног переворачивается :))) *50*> Чет сомневаюсь я, что он аидстест именно патчит. Скорей дописывает к н S0> [цензура], которая графический экран с рекламой переворачивает. Это куда S0> проще.

На самом деле это действительно легко правится заменой всего двух команд в исполняемом файле. Логично, что реклама выводится построчно сверху вниз. Поэтому, указав новое "начало" вывода стр

изменив инкремент на декремент, мы сможем вывести изображение в обратном порядке. Т.е. "вверх ногами".

С последним проще (и DEC и INC — однобайтовые коды), а вот "начало" экрана передается проце как PUSH ES:[BP+xxx]. В отладчике трудно, не выходя из процедуры, найти код, который передает аргумент. Но можно не сходя с текущей позиции курсора, заменить пятибайтовую команду PUSH ES [BP+xxxx] на трехбайтовую PUSH 0x150 (где 0x150 представляет приблизительное число строк экра два оставшиеся байта заменить командами NOP.

Можно наслаждаться перевернутой рекламой. Неплохая первоапрельская шутка или приятный (точ неожиданный) сюрприз для пользователей. Интересно, сколько из них сочтут это проявлениями ков вируса?

Пару слов об "извращенности" AIDSTESTa. Любопытно, но я надеюсь, что SP, говоря о защите рек (которая представляет собой обычный pcx-файл, дописанный к файлу, даже не зашифрованный и ле распознаваемый на глаз в дампе файла при минимальном опыте работы с графикой), не имел ввиду AidsTest'a обычным exe-пакером.

Впрочем, по непонятной причине все выводимые антивирусом тексты слегка зашифрованы. Но настолько тривиально, что это не заслуживает упоминания. Криптор построен всего из двух команд al,el/add c1,7, но шифротекст может быть раскрыт даже без знаний этого алгоритма ввиду его полной некрип-тостойкости.

Самое интересное, что на фоне всего этого верификация собственного кода напрочь отсутствует! В всяком случае, я заменил тексты, перевернул рекламу, потом отключил ее, скорректировал контроль "старости", затем выкинул его напрочь и... никаких ругательств со стороны антивируса.

Плохо, очень плохо выпускать такие программы. Они не обеспечивают даже минимальной безопас и легко могут быть видоизменены вирусом или злоумышленником в своих целях. Еще совсем недав когда отсутствовал Интернет (а вместе с ним быстрые и легальные каналы получения ПО), антивиру распространялись (между прочим, незаконно) через сеть BBS. Нередки были случаи, когда "свежая" версия представляла собой старую с измененной "вручную" версией продукта, но зараженную новы вирусом. Так был широко распространен, например, вирус "Фантом". Помнится, что тогда Дмитрий Николаевич горько сетовал по поводу морали вирусописателей. Это все верно, конечно. Но не задумывался ли он (а вместе с ним и его клиенты), что очень нехорошо, когда массовая антивирусна система имеет такой потрясающе низкий уровень защиты? Небрежность разработчика в который раз причиной эпидемии. А вирусы... (точнее, их авторы) были единственными "козлами отпущения".

Однако я до сих пор не сказал, как найти эти самые команды в сотнях килобайт запутанного кода, которые следует исправить? Есть много путей. Один из них — найти саму рекламу непосредственно памяти. Как уже отмечалось, это обычный pcx-файл. Остается только поставить точку останова на л байт, принадлежащий этому фрагменту, и код, читающий ее, будет искомой функцией вывода карти экран. Отмечу, что не стоит ставить точку останова на диапазон памяти, так как это сильно замедлит выполнение программы, и будет ничуть не эффективнее всего одной точки.

Ничего интересного в AidsTest'e так и не наблюдалось...

Subj: ARJ

В описании заголовка ARJ Роберт Янг не раскрывает значение байта OxB заголовка, оставляя его "зарезервированным". Между тем он активно используется при шифровке паролем!

В arj используется система шифрования Вернама, которая в источнике "Почему криптосистемы ненадежны?" Павла Семьянова неверно отождествлена с гаммированием. На самом деле все очень и просто, и говорить о "криптостойкости" arj можно только в переносном смысле (впрочем, телефоны любовниц в нем — с некоторой натяжкой — еще держать можно).

Пусть пароль представлен как SIS2S3. Получим бесконечную последовательность sls2s3sls2s3sls2s тогда NN-ый символ открытого текста преобразуется в шифротекст простым XOR [Sn],[Nn]+MAG_VALUE. Что такое MAG_VALUE? А это и есть тот "зарезервированный" символ, назначение которого Роберт не раскрыл в документации.

Однако, наивно было бы полагать, что этим можно повысить криптостойкость алгоритма шифрова Это "волшебное значение", видимо, генерируется случайным образом для каждого файла. Назначени не ясно. Никакого влияния на криптостойкость оно не оказывает и вряд ли может быть названо

усовершенствованием алгоритма, поскольку для атаки можно использовать те же самые методы (что "чистого" шифра Вернама) и с тем же успехом.

Любопытно, что пароли abc и abcabc ('5555' и '5') абсолютно идентичны! (И, кстати, об этом умалчи в документации)! Отсюда вытекает, что выбор "неудачного" пароля может позволить "прямую" атак шифр!

Заметим, что если нам известен фрагмент шифротекста в открытом виде, равный (или больший) дл пароля, то "взлом" возможен без трудозатрат и мгновенно. Однако, то, что файлы сжаты, усложняет Необходимо в самом простом случае иметь один файл из архива в незашифрованном виде. С другой стороны, вовсе не факт, что в сжатом тексте нельзя предсказать появления заданных символов в конкретном месте, что раскроет по крайней мере некоторые символы пароля или даже пароль целик это требует дальнейших исследований. И в заключение скажу еще раз: наивно думать, что "хакеры" "компьютерные гуру" — ибо во всех имеющихся у меня хакерских материалах четко прослеживаетс мысль "не просите нас ломать пароли архиваторов".

Subj: AVP разбор полетов

Необходимое замечание

Вирусная энциклопедия AVPVE (кстати, положенная в основу книги "Компьютерные вирусы в MS DOS") — это великолепное и уникальное произведение, на сегодняшний день не имеющее аналогов.

Однако реализация оболочки просмотра оставляет желать лучшего. Не предусмотрен ни контекстн поиск, ни возможность вывода на принтер; нет ни истории, ни других "вкусностей". Кроме того, мое естественное желание получить "линейный" текст (для распечатки и глобального контекстного поис штатными средствами было невыполнимо.

Словом, хотелось выбрать индивидуально-предпочтительный интерфейс работы с энциклопедией (сконвертировать текст в HTML для просмотра через браузер; извлечь все "демонстрационные эффе отдельные файлы для последующего дизассемблирования любопытных экземпляров и т.д.)

По-видимому, я не был одинок в своих желаниях, поскольку в разных источниках появлялись "выломанные" из AVPVE фрагменты описаний и демок. Судя по "мусору" в концах файлов, ясно, чт выдирали "вручную" в дебагере, а тексты (скорее всего) получали экранными грабилками. Понятно, перепечатки без ведома автора — плохая практика.

Каким бы экономичным в плане мыслительной деятельности этот путь ни был, в плане физической (деятельности) он нерационально утомителен. "Расщепление" же всей энциклопедии таким способом потребовало бы немыслимого труда и времени. Кроме того, при выходе новой версии всю эту бессмысленную работу пришлось бы переделывать (а новые версии периодически выходят — имейт виду).

Поэтому я решил исследовать формат файлов для написания собственной гляделки/конвертора или еще-там-понадобится. А понадобится может много чего. Например, неплохо бы при пополнении антивирусной базы (в том числе и пользовательской) добавлять в хелп новые описания и спецэффект

Я отдаю себе отчет в том, что, зная формат dem-файлов, некоторые личности смогут их видоизмен таким образом, что демонстрация может (к большому удивлению юзера) покрошить ему диск. Поэто настоятельно рекомендую не "сливать" никаких компонентов "Вирусной энциклопедии" из сомните источников.

Однако я не нахожу достаточных причин, препятствующих распространению этой информации, поскольку сомневаюсь, что троянизированные компоненты могут получить широкое распространен ведь злоумышленник может уничтожить информацию гораздо более гарантированными способами.

Результат своих изысканий я привожу ниже. Замечу, что пока я не разобрался с некоторыми (в общ некритичными) полями, поскольку необходимости в них не возникло.

Формат файлов (общее)

По моему мнению, формат файлов вирусной энциклопедии плохо продуман, неоптимально реализо часто противоречив и местами исходит из соображений "авось и так сойдет".

Сказанное выше не повод для наезда или оскорбления, а констатация очевидных фактов. Однако эт никак не должно ущемлять Автора, ибо излишняя гениальность в наше время до добра никого не до и экономически нецелесообразна. Вся информация, приведенная ниже, получена путем дизассембли вания вирусной энциклопедии avpve.exe IDA 3.6 и уточнения ряда моментов под Soft-Ice 2.8; назнач

многих полей определялось простым визуальным просмотром, благо что было логичным и вытекающ
содержимого.

Большинство секций всех файлов упакованы по усовершенствованному LZk (k от Касперского) алгоритму, который не меняется от файла к файлу и от версии к версии.

Некоторые секции шифрованы тривиальным XOR [b],OxAD. Смысл этого мне абсолютно не ясен (
погонять процессор), но, как говорят, хозяин — барин.

Алгоритм LZk

Префикс word data Префикс word data Префикс word ~/~

Структура префикса:

lxjxixixixixlxlxlxlxlxlxlxlxlll Data I • '•

^ ^

x - незначащий бит Неупакованный байт данных

^ ^ Указатель Длина L упакованного фрагмента данных + 2

Указатель Длина L упакованного фрагмента данных + 2

Если L==1, то пропустить следующий за префиксом байт как незначащий. Если L==0, то следующий за префиксом байт—длина фрагмента+1.

Если L==0 и следующий за префиксом байт==0, то конец распаковке.

Данной информации хватает для написания пакера/анпакера, хотя пакер необходим только для эффективной перепаковки после внесенных изменений. Если дисковое пространство не критично, то можно прописать в префиксах Oxffff, оставив файлы не упакованными. Но лучше все же написать полноценный упаковщик.

Данный механизм обеспечивает сжатие примерно в 1.8 раз для текстовых данных с незначительными накладными расходами.

Формат файла языковой поддержки AVP.LNG

Большинство сообщений системы Касперского вынесено в файл * .Ing, что удобно для его изменения, редактирования и т.д. Впрочем, для этого понадобится написать специальный компилятор, ибо вручную это сделать не представляется возможным. С этой целью я описываю ниже формат файла.

Ограничения: секция строк ограничена Ox7DOO байтами, (32000 в десятичном представлении), а секция адресов Ox7DO (2000 в десятичном представлении), впрочем, размер секции адресов при редактировании файла не изменяется, поэтому это ограничение можно не учитывать.

Файл состоит из трех частей: заголовка Ox1A байт, следующей за ним секции строк и адресов.

Заголовок Секция строк Секция адресов Ox1A

Рассмотрим подробнее заголовок:

section.string section.addr

"-VLanguageSupport",0 UNP_SIZE PCK_SIZE UNP_SIZE PCK_SIZE

0x12 word word word word

Первые 0x12 байт — это сигнатура, которая проверяется при загрузке файла, UNP_SIZE — оригинальный размер упакованной секции строк/адресов, PCK_SIZE — размер запакованной секции строк/адресов.

Секция строк представляет обычные строковые ресурсы а-ля-Виндоус, которая поксорена по XOR [byte], OxAD; больше ничего интересного не наблюдается. Никаких CRC не предусмотрено.

Сразу за секцией строк начинается секция адресов. Для нее все аналогично, с тем различием, что она не зашифрована.

Обе секции пожаты по описанному выше алгоритму LZk. Секция адресов состоит из far-указателей на строки. Сегмент может быть произвольным, так как при загрузке он устанавливается программой самостоятельно; смещение отсчитывается от 0x4 (именно

такое смещение бывает у выделенных блоков функцией malloc). Это совсем неправильно, ибо нельзя столь беспечно полагаться на системную библиотеку компилятора. Секция адресов сжата, но не шифрована. Однако могу заметить, что глупо использовать такой алгоритм, так как нужно было сохранять одни смещения, а не тянуть за собой никому не нужный сегмент.

Формат файлов HLP

Файлы HLP содержат много структур, точнее, все вместе: и таблицы смещений, и данные, и ссылки на внешние файлы (например dem).

Формат HEAD OEM section.TitleOffset section.OffsetOffset section.text

Формат файлов HLP довольно необычен и содержит несколько подструктур, хотя можно было бы обойтись и одной глобальной структурой; но зато он оптимизирован для медленных машин и ограниченного объема памяти и tiny(!) модели памяти.

Алгоритм работы в общих чертах такой — сначала распаковывается section.OffsetOffset, которая содержит точки входа в структуру TitleHeap. Каждый вход содержит подструктуру TitleOffset, которая имеет переменную длину (вот поэтому и потребовалась вспомогательная подструктура OffsetOffset). В структуре TitleOffset описываются гиперссылки, ссылки на секцию text и ссылки на внешний ресурс.

Заголовок

'.VHG'	1p * OffsetOffset	lp * text	OEM text
dword	dword	dword	not define

Заголовок содержит, кроме традиционной сигнатуры, ссылки на две секции section.OffsetOffset и section.Text; смещения отсчитываются от начала файла.

Указатель на секцию section. TitleOffset в заголовке отсутствует по той причине, что он на него ссылается. Элемент OxC секции OffsetOffset. OEM text, показанный здесь входящим в заголовок для наглядности, на самом деле к заголовку скорее всего не относится, — но это сути не меняет, ибо он совершенно бесполезен... Разве что как копирайт.

Section. OffsetOffset

OffsetOffset — это основной ключ к пониманию структуры файлов hip. Эта секция содержит смещения относительно начала файла на структуры, которые содержат все необходимые ссылки на нужные ресурсы. Все гиперссылки ссылаются на section.OffsetOffset.

Сжатый формат OffsetOffset

Numer Index	Real Numer Index	not use	данные
word	word	word	

Numer Index — число индексов в секции. Все индексы — двойные слова. Real Numer — число реально используемых индексов. not use — а вот это поле однозначно не используется в AVPVE.exe. Секция OffsetOffset упакована по алгоритму LZk и не шифрована.

Распакованный формат

????	____	INDEX I	INDEX 2	INDEX 3	-/-0x30
dword		dword	dword		

Все индексы содержат смещения относительно начала файла.

Назначение первых 0x30 байт неясно. Но и без их понимания все хорошо работает.

Структура TITLE

x_lnk	x_offs	lp *demo	N_Link	LINK	loc_offset	lp *vol
word	word	dword	word	***	word	dword

x_lnk — это поле не используется в текущих версиях и вряд ли его использование намечается в дальнейшем, но содержимое его очевидно — число гиперссылок во внешнем ресурсе.

x_offs — это поле не используется в текущих версиях, но, возможно, будет использовано в последующих. Содержит смещение топика в разделе.

lp *dem — смещение относительно начала файла внешнего ресурса dern; если равно нулю, то внешнего ресурса нет.

N_LINK — число гиперссылок в разделе. Если нуль, то гиперссылок нет. LINK — секция описания гиперссылок (локальная). Размер секции 5*N_LINK. loc_off — локальное смещение подзаголовка в разделе. lp *vol — смещения раздела относительно начала секции section.text.

Замечание. По общему мнению, данная структура неудобна и переменная длина вызывает необходимость введения еще одной структуры OffsetOffset.

Но внимание привлекает любопытная деталь — встроенное кеширование. Дело в том, что несколько заголовков (для улучшения сжатия?) пакуются в один раздел, поэтому для определения необходимого подраздела вводится еще одно двухбайтовое поле loc_offset, которое представляет смещение в распакованном фрагменте. Двухбайтовая величина наводит на мысль от том, что максимальный размер раздела Oxffff байт — полезный штрих для практической реализации.

Интересный момент — avpve.exe не использует кеширования и при чтении следующей главы в уже распакованном разделе повторно его распаковывает! Ну что тут можно сказать...

Структура ссылок

INDEX offset Length

word word byte

INDEX — это индекс структуры OffsetOffset.

offset — смещение начала гиперссылки в разделе. Используется браузером для "подсветки" гиперссылок.

Length — длина гиперссылки, также нужна браузеру для выделения.

Структура TITLE

Первая секция:

N_TOPIC SIZE TOPIC flag? ____text____

word word word

Последующие секции:

SIZE TOPIC flag? text

word word

N_TOPIC — число подразделов в разделе. SIZE_TOPIC — размер подраздела в байтах. flag — не разбирался. Но и без него работает. **Структура DEM-файлов**

a?-v word

DEM-файлы (как внешний ресурс) своей структуры как таковой не имеют, ибо все ссылки на них находятся в базовом HLP-файле. DEM-файл — это просто набор ресурсов, смещения которых хранятся в структуре TitleHeap в HLP-файле. Поэтому читайте ниже о формате ресурсов, а пока обратите внимание, на то, что dern-файлы имеют свою сигнатуру в заголовке, без которой стандартный просмотрщик откажется их просматривать.

Формат ресурсов

PCIGSIZE d a t a dword

Ресурсы dem-файла — это "демонстрации вирусных эффектов", уже "готовые к употреблению" corn-файлы. Остается их только извлечь.

Поле PCK_SIZE — это размер упакованного ресурса. Двойное слово. Сами данные упакованы и шифрованы.

Subj: Bounds-Checker 5

Для ковыряния в недрах программ, написанных под Windows, и ломания под ней же, Bounds-Checker в сочетании с Soft-Ice просто превосходен. Например, не нужно "угадывать", какой же функцией манипулирует защита: GetDIgIternTextA, GetWindowTextA, GetWindowText или, может быть, чем-то другим? Достаточно просто изучить рапорт Bounds-Checker-a.

И была бы коту масленица, но преподнесла судьба ему сюрприз в виде 'evaluation version' в том смысле, что после 30 дней надо готовить $. А если их нет? Но если у человека нет $,

то у него наверняка есть отладчик. Вот им мы и воспользуемся. Я вообще смутно понимаю мотивы, побуждающие ставить любого рода защиты на хакер-ориентированное программное обеспечение, которое часто отламывается меньше чем за минуту! Не был исключением и этот случай.

Процедура регистрации наводит на мысль, что вероятнее всего используется привязка к имени владельца/фирмы, но это не так. "Отпирающий код" не зависит от вашего имени. Чтобы в этом убедиться, достаточно поставить брейк-пойнт на GetWindowTextA, дальше брейк-пойнт на введенный вами код и... нет, ну это просто издевательство — тривиальное сравнение двух строк! Запомним эталонный код (или запишем его на бумажку). Вводим его — работает?! Обидно все-таки — за кого нас принимают все время? Впрочем, я не уверен, что этот код будет одинаковый во всех версиях... возможно, есть множество версий с разными кодами (ибо, помнится, там еще и серийный номер высвечивался). Но я не думаю, что они защищены иначе. Кстати, при переустановке версия останется "Зарегистрированной", это приятно. И не надо патчить код.

Subj: CD-MAN EGA Version

Добрая, старая, великолепная игрушка! Выполненная в высшем EGA разрешении (будет почище некоторых VGA-шных), очень красочно нарисованная. Хорошо чувствуется вкус и тонкий добрый юмор автора. Занимая всего 380 килобайт и отвечая всем современным требованиям (только музыка на спикер...), CD-MAN вызывает трепещущее ностальгическое чувство: "Эх, делали же игры во времена нашей молодости...".

Но, попав на Pentium, Сиди-ман бегает как... словом, очень быстро! Надо исправить... Несколько раз остановив игру, мы, вероятно, должны попасть в тело процедуры ожидания (хотя это бывает и не всегда, но достаточно часто). То, что это именно процедура ожидания, легко узнать по характерным циклам. Вот что мы обнаружим. Хм... неплохо оптимизированный код! Нетрудно понять его смысл... увы! задержка привязывается не к таймеру, а к быстродействию компьютера. Посмотрим, что можно сделать...

Итак, ясно, что интервал задержки (передаваемый как параметр этой процедуре) хранится в слове DS:[ll20h], учитывая размер процедуры, можно было бы переписать ее полностью, используя RTC через функцию f.86h прерывания 15h. тогда бы она работала на всех компьютерах одинаково. Но давайте поступим иначе. В 6123h и 612Dh в CX грузятся константы. Ясно, что если их увеличить, то задержка возрастет. Увеличим пропорционально эти две константы — скажем, раз в пять — и полюбуемся результатом... Да, теперь скорость нормализовалась...

```
0000611D: 53                push by.
0000611E: 51                push ex
0000611F: 8B1E2011              mov bx,word ptr [1120]
00006123: B90800              mov ex, 0008          ;<—1
00006126: 803E540200            cmp byte ptr [0254] ,00
0000612B: 7403              jz 00006130         ;—,
0000612D: B90500              mov ex, 0005
00006130: E2FE              loop 00006130       ;«—^
00006132: 8BCB              mov cx,bx
00006134: 4B                dec bx
00006135: E2EC              loop 00006123       ;—
00006137: 59                pop ex
00006138: 5B                pop bx
00006139: C3                ret
```

Subj: F-PROT2.19

Вообще-то я не доверяю антивирусам... Как бы ни совершенствовался автоматический поиск, но до ручного ему еще далеко. Я не полагаюсь на антивирусы, а ищу и отлавливаю "насекомых" сам. F-PROT запустил из чистого любопытства. Версия 2.19, ссылаясь на устарелость версии, бесцеремонно "выплевывает" меня в DOS! Да, наглый нынче народ

стал. Ну что, будем лечить... Т.е. удалим проверку даты, а то лень переводить ее назад (да и глупо).

Нажимаю F3 и вижу (по отсутствию таблицы перемещаемых элементов), что файл явно упакован. Падчик лень делать (хотя я уже имел к этому времени готовый инструментарий, но я не сторонник падчиков), поэтому пытаюсь распаковать файл первым, что попадается под руку, — UNP, который сполна оправдывает возложенные на него надежды.

Запускаю распакованный файл... Мда, контроль своей длины и 'System Halted' — загружайся, мол, с чистой дискетки. Пытаюсь использовать "дурацкий" прием, который проходит со многими антивирусами. Длина упакованного файла — 109635, в шестнадцатиричном — Ox1AC43; пытаюсь найти в файле Ox43AC (надеюсь, понятно почему). Очень часто это проходит, но... не на этот раз. Жаль...

Гм, что бы еще сделать? Установить точку останова на 21h,f.3Dh (открытие файла)? Пробую... Увы, эффект нулевой. Как же еще можно вычислить длину файла? Через f.4Eh/f,4Fh — FindFirst/FindNext, но и это безрезультатно!

Что мы имеем? Мы испробовали все методы "от дурака". От вируса-дурака, от хакера-дурака. Следовательно, антивирус стоит чуточку выше! Это делает ему честь, но что же делать нам?! Вот тут главное не запаниковать и не растеряться. Из этой ситуации есть три выхода:

1) Просто трассировать программу, надеясь найти "Nag"-процедуру; но это настолько глупо, что бессмысленно рассматривать здесь.

2) Продолжить перебор попыток угадывания алгоритма определения длины файла. — Увы, это может слишком затянуться; кроме того, если он окажется чересчур нестандартным, мы рискуем потерять уйму времени без гарантии отыскать его. Мы и так потеряли пару минут, перебирая несколько очевидных вариантов...

3) "Zen-Method". Т.е. такой метод, когда мы не вникаем в механизм алгоритма, а просто локализуем относящийся к нему участок кода, который локально изучаем.

Что ж, пойдем третьим способом. Бесспорно, вызывая отладчик в "ругательном" сообщении, мы одним из методов сможем определить точку вызова. Скажем, по стеку, хотя очень часто "Nag"-экран находится внутри интересующей нас процедуры, а не выделен в отдельную. За неимением лучшего выхода вызовем отладчик в "Nag"-экране. И... мы глубоко в BIOS. Аккуратно выберемся из нее, потом из DOS. Оттуда в системную библиотеку и наконец, в вызывающий этот экран код. Привожу фрагмент, на котором следует заострить внимание.

```
OOOID25C: B86400          movax,0064
OOOID25F: 50              push ax
OOOID260: IE              push ds
OOOID261: B81C4E          mov ax,4EIC
OOOID264: 50              push ax
OOOID265: 56              push si
OOOID266: 9A24007F25      call 257F:0024
OOOID26B: 83C408          add sp,0008
OOOID26E: 8B16283D        mov dx,word ptr [3D28]
OOOID272: A1263D          mov ax,[3D26]
OOOID275: 056800          add ax,0068
OOOID278: 83D200          adc dx,0000
OOOID27B: 3B56FE          cmp dx,word ptr [bp-02]
OOOID27E: 753F            jnz OOOID2BF
OOOID280: 3B46FC          cmp ax,word ptr [bp-04)
OOOID283: 753A            jnz OOOID2BF
OOOID285: A1343D          mov ax,[3D34]
OOOID288: 3B06884E        cmp ax,word ptr [4E88]
OOOID28C: 7531            jnz OOOID2BF
OOOID28E; 803EIC4E46      cmp byte ptr [4EIC),46
```

```
  OOOID293: 752A                jnz OOOID2BF
  OOOID295: 803EID4E53            cmp byte ptr [4EID],53
  OOOID29A: 7523                jnz OOOID2BF
  OOOID29C: 833EE63800            cmp word ptr [38E6],0000
  OOOID2AI: 753B                jnz OOOID2DE
  OOOID2A3: 8B16864E              mov dx,word ptr [4E86]
  OOOID2A7: A1844E                mov ax,[4E84]
  OOOID2AA: 050400                add ax, 0004
  OOOID2AD: 83D200                adc dx,0000
  OOOID2BO: 52                  push dx
  OOOID2BI: 50                  push ax
  OOOID2B2: 56                  push si
  OOOID2B3: 9A05008D07            call 078D:0005
  OOOID2B8: 83C406                add sp,0006
  OOOID2BB: OBCO                  or ax, ax
  OOOID2BD: 751F                jnz OOOID2DE
  OOOID2BF: B80401                mov ax, 0104
  OOOID2C2: 50                  push ax
  OOOID2C3: 9ABF04CFOF            call OFCF:04BF
  OOOID2C8: 59                  pop ex
  OOOID2C9; EBOO                  jmp OOOID2CB
  OOOID2CB: 9A09005529            call 2955:0009        ; <=—Nag Proc OOOID2DO:
3DEIOO              cmp ax,OOEI         ; <—мы здесь OOOID2D3: 75F6                jnz
OOOID2CB
  OOOID2D5: 33CO                xor ax,ax
  OOOID2D7: 50                  push ax
  OOOID2D8: 9AOIOOA129            call 29Al:0001
  OOOID2DD: 59                  pop ex
  OOOID2DE: 81368A4EFFFF            xor word ptr [4E8A],FFFF
  OOOID2E4: 80368C4EFF            xor byte ptr [4E8C],FF
  OOOID2E9: 80368D4EFF            xor byte ptr [4E8D],FF
  OOOID2EE: 56                  push si
  OOOID2EF: 9A02009A2A            call 2A9A:0002
  OOOID2F4: 59                  pop ex
  OOOID2F5: 57                  push di
  OOOID2F6: 9A02009A2A            call 2A9A:0002
  OOOID2FB: 59                  pop ex
  OOOID2FC: 5F                  pop di
  OOOID2FD: 5E                  pop si
  OOOID2FE: 8BE5                  mov sp,bp
  OOOID300: 5D                  pop bp
  OOOID30J: CB                  retf
```

Мы очутимся в строке OOOID2DO. Теперь посмотрим, как мог быть выполнен переход на "Nag"-экран... Ближайший условный переход, возможно, подходящий на эту роль, — это OOOID2BD:jnz OOOID2DE; проверка показывает, что переход на указанный адрес приводит к нормальному продолжению работы программы. В большинстве случаев замены 0x75 на OxEB было бы достаточно, но не подсказывает ли вам интуиция, что сегодня этого будет мало? Жаль... действительно мало, и прога по-прежнему ругается и не запускается.

Посмотрите выше. Сколько разветвлений... мы, как сапер, должны все обезвредить, но для начала проанализируем их. Мы уже знаем, что переходы на строку ID2DE безопасны и должны быть заменены на безусловные. А остальные? Виднеется масса переходов на OOOID2BF. Легко проследить, что этот путь к "Nag"-экрану. Если выполняется цепочка

сравнений с переходами на одну строку в случае несовпадения, то легко догадаться, что это такое... Итак, заменяем их на 0x9090, а также в строке OOOID2BD запишем безусловный переход. Опля! Это работает! Но... но что-то слишком много замен, вы не находите? Нельзя ли сделать меньше? Да, можно. Если мы в строке OOOID2CB запишем переход на ID2DE, т.е. в первой строке "nag"-кода сделаем переход к правильному выполнению программы, то это будет работать, хотя мы заменили всего два байта. И это гораздо предпочтительнее, потому что не заставляет шарить по всему коду в поисках всех условных переходов на "nag"... Но есть более красивое решение.

Находим первое сравнение и оттуда делаем jrnp на ID2DE... Так я и поступил, изменив всего один байт.

Наконец-то программа перестала реагировать на изменившуюся длину. Должен сообщить, что это грязный и пошлый способ. Ибо теперь длина вообще не контролируется и при заражении вирусом программа об этом, увы, уже не сигнализирует. Желающие могут попробовать разобраться в алгоритме (благо код, выполняющий это, мы уже локализовали) и скорректировать новую длину файла.

Так, теперь проверка даты... Можно также было бы вызвать отладчик в "Nag"-экране, но это не лучший путь, не так ли? Логично, что программа должна опросить системную дату, чтобы узнать, насколько она устарела. Есть мало причин, которые могли бы заставить разработчика не использовать отличную от f.2Ah функцию операционной системы. В самом деле, остальные методы чреваты большей головной болью и меньшей совместимостью.

Ну что же, я свою ставку сделал, а вы как знаете! Опа! Сработало! Потрас-сировав самую малость, мы наталкиваемся на очень выразительный кусок кода.

```
00005CA7: 8956FE          movwordptr [bp-02],dx
00005CAA: BFBCOO            mov di,OOBC
00005CAD: 3BF7            cmp si,di
00005CAF: 7DOC             jnl 00005CBD
00005CBI: B81100           mov ax, 0011
00005CB4: 50             push ax
00005CB5: 9ABF04CFOF          call OFCF:04BF        ; <—Nag Scr 00005CBA: 59
pop ex
00005CBB: EB41             jmp 00005CFE
00005CBD: 803E654EOO          cmp byte ptr [4E65],00
00005CC2: 741F           jz 00005CE3
00005CC4: A0654E            mov al,[4E65]
00005CC7: 98             cbw
00005CC8: 03C7             add ax,di
00005CCA: 3BC6              cmp ax, si
00005CCC: 7D15             jnl 00005CE3
00005CCE: B84702            mov ax, 0247
00005CDI: 50             push ax
00005CD2: 9ABF04CFOF          call OFCF:04BF
00005CD7: 59             pop ex
00005CD8: B80100            mov ax, 0001
00005CDB: 50             push ax
00005CDC: OE              push cs
00005CDD: E8D4FE            call 00005BB4
00005CEO: 59             pop ex
00005CEI: EBIB             jmp 00005CFE
00005CE3: 833ED23800          cmp word ptr [38D2] ,0000
00005CE8: 7514             jnz 00005CFE
00005CEA: A0614E            mov al, [4E61]
00005CED: 98             cbw
```

```
00005CEE: 03C7                  add ax,di
00005CFO: 3BC6                   cmp ax, si
00005CF2: 7DOA                   jnl 00005CFE
00005CF4: B81200                 mov ax, 0012
00005CF7: 50                   push ax
00005CF8: 9ABF04CFOF                call OFCF:04BF
00005CFD: 59                    pop ex
```

```
00005CFE: 833ED23800          cmpwordptr [38D2] ,0000
00005D03: 7515                jnz 00005DIA
00005D05: A0614E              mov ai,[4E61]
00005D08: 98                  cbw
00005D09: 0346FE              add ax,word ptr [bp-02]
00005DOC: 3BC6                cmp ax, si
00005DOE: 7DOA                jnl 00005DIA
00005DIO: B8F901              mov ax,OIF9
00005D13: 50                  push ax
00005D14: 9ABF04CFOF          call OFCF:04BF
00005D19: 59                  pop ex
00005DIA: 5F                  pop di
00005DIB: 5E                  pop si
00005DIC: 8BE5                mov sp,bp
00005DIE: 5D                  pop bp
00005DIF: CB                  retf
```

Не нужно быть чересчур опытным, чтобы догадаться, что call OFCF:04BF и есть та процедура, которая выводит ругательное сообщение; впрочем, это так же легко выяснить экспериментальным путем. Дальше, как говорится, дело техники. Мы находим все условные переходы, которые могли бы "шунтировать" эту процедуру и, судя по обстоятельствам, заменяем их. Как и следовало ожидать, проверка не одна. Причем проверяется даже такая экзотика, как некорректность системной даты... Сколько лишнего кода... А где оптимизация?

Даже беглый взгляд показывает, что не все "nag"-и приводят к выходу в DOS. Более "честный" вариант взлома — не выходить в DOS, а просто сообщить о "старости" и продолжить работу. Но это решать вам... Можно также "обновить" дату, но и это бессмысленно, ибо антивирус уже устарел. От того что он заругается, скажем, через месяц, смешно не будет. Пусть уж постоянно ругается (но в DOS не выходит, как все нормальные программы), либо вообще ничего не проверяет. Как видите, варианты есть.

Вот я и показал действие "Zen"-метода: когда мы, абсолютно не вникая в используемые антивирусом алгоримы, сумели добиться поставленных целей. Конечно, дальнейшие действия немыслимы хотя бы без поверхностного анализа, но главное сделано — измененная программа запускается!

А антивирус действительно ничего... Обширная база, неплохой эвристик (хотя с уймой ложных срабатываний), возможность собственноручного пополнения сигнатур... Конечно, по сравнению с Касперским это все детский лепет, хотя лично мне времени, потраченного на взлом, было не жалко — реакция антивируса на мою коллекцию вирусов меня достаточно позабавила и окупила время, затраченное на его "покусывание"...

Subj: FDR2.1

FDR 2.1 — великолепная программа, предназначенная для качественного восстановления разрушенных дискет. И хотя мне не довелось ее испытать в работе (последнее время у меня дискеты не рушились), подобная утилита всегда должна быть под рукой.

Довольно витиеватый графический интерфейс, стилизованный под ранний MicroSoft, но выполненный аккуратно и со вкусом. В общем, прога мне так понравилась, что даже возникло желание потратить свои кровные 5$ на регистрацию, ибо она того стоила. Но... мы живем не в идеальном мире...

Краем уха я слышал о якобы "крутой" защите FDR, что не могло меня (как любопытного человека) оставить равнодушным.

Вскрытие этой программы разочарования не оставило, хотя и не было сложным. Очень простая защита в простом варианте ее реализации. Антиотладочные приемы были, но увы безнадежно устаревшие. Например, в таблицу векторов прерываний записывались константы, которыми спустя некоторое время расшифровывался один фрагмент. Наивно, да? Правда, сам файл зашифрованный, причем он активно взаимодействует со своим телом на диске, поэтому расшифровка его приведет к краху программы. Но это лишняя головная боль для автора, а хакеры, вооруженные TSR-падчерами, мгновенно сведут на на нет его усилия.

Единственный привлекательный в защите момент (настойчиво рекомендуемый мною всем авторам защит) — то, что регистрационный номер нигде явно не проверяется (проверяется только CRC), а сам используется для =расшифровки== некоторой критической секции кода. Правильно реализовав этот метод, можно было бы сделать взлом по меньшей мере нецелесообразным или сводящимся к утомительному поиску ключа (хотя, имея в наличии хотя бы одну зарегистрированную копию, можно было вычислить пароль). Но увы, данная защита реализована с ошибками. Сдается, что она расшифровывает не "недостающий код", а процедуру, которая, цепляясь на int 21h, "откликается" на вызов модуля защиты. Таким образом, это можно вскрыть приемлемыми усилиями, даже не имея ни одной легальной копии. Впрочем, мне повезло. Одна легальная копия была в моем распоряжении, поэтому не составило труда разобраться в механизме генерации паролей и составить key-generator.

Ниже я предоставляю исчерпывающую информацию, необходимую для написания собственного генератора. В поле "Name" вводится текстовая строка до 24 символов, из которых недостающие символы заполняются пробелами. В поле "reg code" вводится ASCII- строка шестнадцатиричных символов '0-F', которая преобразуется в числовую последовательность (назовем ее ключом) несколько необычным способом. Младший и старший полубайты инвертированы, т.е. ТГ преобразуется в IFh. Ключ представляет собой следующую структуру:

Данные
C R C Шифрованное по паролю выше -имя-
(слово) (не менее 20 символов)

Первые четыре ASCII-символа (т.е. одно слово) — это CRC. Алгоритм подсчета CRC следующий. Сумма 18h ASCII кодов имени (недостающие дополняются пробелами) складывается с побайтовой суммой в "д а иных" ключа (в числовом представлении!), а затем дополняется до нуля.

Т.е. CRC+sum(#Name)+sum(OxKeyDate) = 0. Данные получаются шифрованием имени по магическому слову 'Pink Floyd' следующим алгоритмом, который демонстрирует следующий ассемблерный фрагмент:

```
XOR    BX, BX                    unsigned char A, B = 0;
MOV    CX, OAh                     for (c=0;c<OxA;c++)
s_repeat:                          (
ADD    AH, [BX+Offset _Names]        A+=_Name[c];
MOV    AL, [BX+Offset _Magic]        B=_Magic[c];
SUB    AL, AH                    B-=A;
XOR    AL, 49h                   B= B ″ 0x49;
MOV    [_KeyDate+BX), AL             _KeyDate[c]=B;
INC    BX                      C++;
LOOP   s_repeat                  }
```

Однако я не сторонник генераторов серийных номеров. Я предпочитаю "выкусывать" защиту из тела программы, и если мне захочется покопаться в ней, то я это сделаю.

Subj: HEXEDIT.EXE Version 1.5

Полезная утилитка, но большей частью для новичков, так как очень неудобна в обращении при хаканье кода... но зато под Windows! Разумеется, ShareWare; т.е. хочешь пользоваться — регистрируйся. Не то чтобы в незарегистрированной версии были заблокированы какие-то полезные возможности... но все же надпись "UnRegistred" довольно неприятна...

Ломается это не сложно — для регистрации выбираем специальный пункт меню, куда вводим свое имя, а затем код регистрации. Первая мысль, приходящая в голову: BPX GetWindowText. Оппс! Это сработало... теперь трассируем и видим, что условный переход выполнялся после сравнения с переменной [OECOh], которая в регистрированной версии не должна быть равна нулю. Ну, тут теперь и нечего делать — BPM DS:OECOh и перезагружаем программу. Ответ отладчика на BPM:

```
C6 06 CO OE 00   MOV Byte ptr [OECOJ.O <-- это
C4 7E 06         LES Dl, (BP+6]
```

Теперь заменим это на:

```
C6 06 CO OE 01   MOV Byte ptr [OECO],1
C4 7E 06         LES Dl, [BP+6]
```

Интересно, что сейчас в окне регистрации можно вводить что угодно и это работает!

Примененный способ достаточно универсален для защит "регистрации" — приложение вводит переменную "Registred", обычно булевскую (хотя и не обязательно), и в инициализационных процедурах, делая проверку регистрации, устаналивает ее. Вот эту-то строку мы изменили в данном случае! (И в других случаях тоже перспективнее именно такая замена)...

Теперь все отлично, но все же почему-то в заголовке окна вместо имени редактируемого файла горит 'Unregistred version...' Непорядок! Ну, тут задачка для начинающих — ставим BPX SetWindowText и ждем появления оного в заголовке. Оп-пс! Сначала появился 'верный' заголовок, затем затерся просьбой о регистрации... Просматривая окружающие коды в дизассемблере, находим те, которые могут шунтировать это сообщение. Где-то сверху на приличном расстоянии мы их и находим. Вот они:

```
7403         JE Loc_l
E9 7F 00     JMP Ругательное_сообщение
:Loc_l
26 3B 85 54 04 CMP AX,ES:[DI+454]
75 78        JNZ Ругательное сообщение
: Normal Cont
```

Хм, можно сразу заменить JE Loc_l на NormaLCont, а можно последовательно: JE Loc_l -> JMP Loc_l; JNZ NormaLCont -> NOP+NOP. Как кому по душе...

Вот "защита" и взломана...

Subj: Поле чудес

Деятель из Арзамаса создал "Капитал-Шоу", которая теперь чуть ли не в любой конторе на любой машине стоит. И все бы хорошо, да вот досада — не наградила судьба утилитой

для редактирования и добавления слов. Автору, видно, лень было клавиши нажимать, и ввел он всего-то 450 слов. Для заядлых поклонников, как говорится, на одну затяжку...

Однажды меня попросили кой-чего в игре доделать. Понятно, что исправлять чужую программу неэтично, так что это между нами. Но вот утилиту для разборки/сборки словаря на всеобщее растерзание отдать можно.

Пару слов о программе. Она написана на Turbo Pascal 6.0 с самопальными графическими библиотеками. Как и все программы, написанные на языке Pascal, в исполняемом коде выглядит ужасно. Мастдаевский стиль автора для "того" времени любопытен. В смысле оптимизации (ее отсутствия). Неупакованный словарь можно было бы объяснить параноидальной склонностью к оптимизации по скорости, однако на диске словарь занимает больше двух собственных размеров.

Словарь расположен в файле pole. ovi, но ни по структуре, ни по способу загрузки это совсем не ovi, а обычный типизированный паскалевский файл, в котором хранится массив строк stringl0xl4]. Первый элемент массива — число слов в строковом представлении.

Каждое слово представлено двумя элементами массива — первый элемент слово, второй — тема. Да-да, одна и та же строка с темой повторяется десятки раз! Словом, тут не все оптимально... Особенно если учесть, что лишь очень редкое слово занимает 0x14 байт, а место под него отводится.

Замечу по ходу дела, что в конце строк расположен "мусор", однако этот мусор "вытянут" с машины автора без его ведома и согласия. Что в стеке было, то и попало. В подобных системах (разве что резервирующих места на два порядка поболее этого) есть определенный риск "упустить" все что угодно — от фрагментов конфиденциального текста до интернетовских паролей. Так что систем без дыр не бывает.

Однако визуально слова имеют довольно плохую читабельность. Сразу в голову лезут кодировки разные там, циклические сдвиги и ксоренья. Увы. Это бывает часто, но здесь причина другая. Плохая читабельность — это не попытка сокрыть слова от пользовательского взгляда и HIEW-a, а еще одно проявление "трудолюбия" автора, которому, видимо, лень было строить таблицу для перевода строчных русских букв в заглавные. Проще добавлять/отнимать 0x20.

Но я никак не пойму: зачем? Зачем автор вводит с клавиатуры буквы в верхнем регистре, весьма коряво переводит их в нижний, который и сохраняет в словаре, а потом обратно переводит в верхний. Как я ни старался, но убедительных причин найти для объяснения необходимости таких махинаций я не нашел.

Таким образом, словарь нетрудно просмотреть/изменить HIEW-OM, а для ленивых использовать мою утилиту.

Формат:

```
str     Все строки паскалевские, для незнающих    Procedure Decode (s: string)
                  String [$14] выглядит так:              var
_N_WORD_____________________________________________  a: word;
 word_l    Length      string      мусор                 begin
 team_i    byte        length                            for a:=1 to Length (s) do
 word_2                0x15                              asm
 team_2      N_Word в строковом десятичном или             les bx, [BP+4]
             шестнадцатиричном представлении               add bx+a
                                                    >SUB Byte ptr ES: [bx], 20h _//_ end
                                                           end;
```

Subj: SGWW password protection 'WhiteEagle'

Попался мне на CD электронный журнал SGWW. Но, попытавшись раскрыть arj-архив, я с удивлением наткнулся на пароль. Ну пароль и пароль. Но когда на другом диске (скорее всего, перепечатка) я наткнулся на тот же самый "закрученный" файл, то призадумался. К тому времени я собрал подборку журналов SGWW, которую с интересом пролистывал. Но "закрученного" номера в моей коллекции еще не было. Жаль. Но слабость алгоритма

шифрования arj уже стала "притчей во языцех", и даже "лобовой" атаки на ключ методом перебора не требовалось. Впрочем, я ради эксперимента оставил на ночь "атакующую" программу, которая к утру так и ничего не нашла.

Известно, что используемая в arj схема шифрования при наличии открытого фрагмента шифротекста может быть легко раскрыта. Однако, в этом-то обычно и заключается проблема: далеко не всегда "злоумышленнику" доступен хотя бы фрагмент архива в нешифрованном виде. Но у меня такой фрагмент был! (Впрочем, это не было очевидно.) Им был файл pgpkey в каталоге INFO — он совпадал с имеющимся у меня из восьмого номера (что видно по длине и дате файла).

Таким образом, мне не составило труда расшифровать все остальные файлы и найти пароль "WhiteEagle". Десятисимвольный пароль можно было найти и лобовой атакой на ключ. Так что защита все же несерьезна, что бы об этом ни говорили. Атака по словарю также могла бы раскрыть это за доступное для анализа время.

Поэтому еще раз хотелось бы обратить внимание на то, что человек, заявляющий: "только не просите меня взломать пароль ARJ" вряд ли может называться хакером. Право, не стоит переоценивать возможности существующих широко распространенных защит, и тем более полагаться на них в серьезных случаях.

Subj: SOURCER 5.10

Скачал я с CD пятый SOURCER и с удивлением увидел, что он требует серийного номера. Откуда же мне его знать? Но хакер я или нет?!

Вообще-то мне непонятен смысл защиты на SOURCER. Не знаю, как там "у них", а у нас это чисто хакерская подручная утилита. Не думаю, чтобы прикладные программисты держали ее на своих компьютерах... Но ставить защиту такого уровня — это совсем глупо. Авторы SOURCER-а, видимо, всерьез думают, что ломают программы только с серьезными защитами. Или они нас совсем не уважают... Мне не потребовалось много времени, чтобы "вскрыть" эту "защиту". Интересно, на кого же они рассчитывали?! Ни одной хитрой ловушки, ни одного антиотладочного приема, ни одной недокументированной возможности, ни даже проверки собственного кода! Неужели кто-то всерьез думает, что даже начинающего хакера можно спугнуть "nag screen"-ом и заставить побежать регистрироваться?

Впрочем, к чести защиты можно сказать, что "убить" ее, изменив пару-тройку байт, никак не удастся. Слишком много проверок в самых разных местах. Довольно забавно смотрится эта наивная технология. Понятно, что вылавливание всех проверок занимает больше времени, но жаль, если это единственное, чем авторы надеялись затруднить взлом.

Subj: Как взломать Emulated Solar CPU

Это очень нетривиальный CrackMe и очень приятный. Ломается, правда, просто, но не всеми. Этот опус меня побудил написать один мой знакомый, который уже неделю его копает.

Все-таки математику и хакерам знать не помешает, хотя бы на уровне популярных книжек. В эмуляторе используются только основы булевой алгебры, которые (если я не ошибаюсь) входят и в программу школьного курса информатики.

Solar Designer пишет:

- Ну, кто сломает мое творение? ;) Кода 80x86 — меньше 100 байт.
- Шифрованного кода нет. Защиты от отладки тоже нет.
- Пароль сознательно зашифрован так, чтобы можно было разобравшись в коде его
- проверки, вычислить подходящий. И тем не
- менее сломать будет непросто. ;)

Тем не менее сломать просто, и даже очень просто! Ну-с, начнем... Как уже понятно, нам придется иметь дело с эмулятором. Или интерпретатором, кому как больше нравится. Никаких конвейеров, циклов выборки, распараллеленных дешифраторов команд нет, т.е. приятного особо не много. Минимально рабочий интерпретатор а-ля риск с фиксированной длиной и адресацией всех команд.

Но что примечательно, так это необычная архитектура. Не Фон-Неймановская. Исполняемый код и данные перемешаны. Правда, реализация не очень аккуратная и навешано очень много заплаток 186, что портит все удовольствие и сводит сложность взлома к нулю.

Далее, автор думает, что...

- В защитах я такого не встречал, а ведь именно в них это
- полезно, т.к. не дает возможности полной декомпиляции из-за невозможности
- распознать границы команд более высокого уровня (в данном случае они были
- на макросах).

Математику знать надо. A+B+C == (A+B)+C во всяком случае. Чем мы ниже с успехом и воспользуемся. Я думаю, что стрелка Пирса — это не самое лучшее для затруднения декомпиляции. А вот скорость замедляет изрядно. И серьезная защита будет сильно "тормозить". А что булевы термы ассоциативны, надо все же иногда иметь в виду.

Для написания декомпилятора или отладчика мы должны сначала разобраться с механизмом самого эмулятора. Поскольку защиты нет, подойдет даже Debug.com.

```
> 16FO:0101 8B365201   MOV     SI, [0152] ; Указатель команд
> 16FO:0105 AD         LODSW              ; Читаем 1st операнд
> 16FO:0106 97         XCHG    DI.AX      ; Dl :″ [1st]
> 16FO.-0107 8B3D      MOV     Dl, [Dl]   ; "
> 16FO:0109 AD         LODSW              ; Читаем 2st операнд
> 16FO:OIOA 93         XCHG    BX.AX      ; BX := &2st
> 16FO;OIOB OB3F       OR      Dl, [BX]   ; tO := [1st) I [2st]
> 16FO:OIOD AD         LODSW              ; Читаем 3st операнд
> 16FO:OIOE 97         XCHG    Dl,AX      ; AX := tO ; Dl = 3st
> 16FO:OIOF F7DO       NOT     AX         ; AX := NOT tO
> 16FO:0111 89365201   MOV     [0152],SI .• Opdate ernIP
> 16FO:0115 AB         STOSW              ; mov [3st],NOT(OR [1st], [2st])
> 16FO:0116 EBE9       JMP     0101       ; -- цикл выборки команд --"
> 16FO:0152 5A01                          ; ernREG :: ernIP
```

Обратите внимание на строку 0115—в ней заключена вся логика эмулятора. Именно тут собака зарыта. MOV [3st], NOT(OR [1st),[2st]). Всего одной этой команды достаточно для реализации всех остальных.

Как учит булевская алгебра, есть только две логические операции NOT и OR, и все остальные (AND, XOR) можно выразить через эти две. Причем, учитывая, что OR A,A = A, можно сказать, что данная функция может действовать как NOT (PIRS A,A), а следовательно, и как OR (PIRS(PIRS(A.B), PIRS(A,B)). Поскольку приемник представляет собой непосредственное значение, то можно также создать MOV и JMP (последний — изменяя "регистр" указателя команд).

И все. Защите конец. Мы уже знаем как декодировать числовые последовательности и можем без напряжения вычислить "границы команд более высокого уровня". Те, кто знаком с булем, могут пропустить следующий абзац, а для остальных маленький ликбез. Итак:

```
AND == NOT[( OR(NOT(A), NOT (B))];
XOR == AND[ OR (A,B), NOT(AND (A,B))];
MOV == NOT(NOT(A)) или OR (A,A)
JMP =- MOV ([emlP],[A])
JMP == JMP [t0l\t0 DW offset Label
```

Таким образом, мы теперь имеем базовый набор команд для создания логики "второго уровня". В отличие от первого уровня, одни и те же определения второго могут быть построены различными комбинациями. Это не позволит применить шаблоны. Но что-нибудь еще придумаем.

Автор сдуру включил фрагменты исходников "чтобы легче было ломать". Я так и не понял глубокий смысл этой акции, но, раз нам дали кусок исходника, давайте проверим на совпадение команд первого уровня.

- ernNOT macro Src, Dst
- ernNOR Src, Src, Dst
- endm
- emOR macro Srcl, Src2, Dst
- ernNOR Srcl, Src2, emRI
- ernNOT emRI, Dst
- endm

Эти два фрагмента явно включены ради трафика, поскольку никаких других (кроме очевидно избыточных) реализаций данных команд не существует.

>emAND macro Srcl, Src2, Dst

- ernNOT Srcl, emRI
- ernNOT Src2, emR2
- ernNOR emRI, emR2, Dst
- endm

Таким образом, математика торжествует и, как мы видим, для написания декомпилятора исходники абсолютно не нужны.

Переходим к логике. Все пестрое множество реализаций может крутиться только вокруг двух формул:

A ===> B == OR(NOT(A),B)

A <--> B == OR [AND (A,B),AND (NOT(A),NOT(B))]

Например, нам встретится следующая последовательность:

OR [AND (A,B), NOT(OR(A.NOT(C)))]

Тех, кто еще не понял, что это делает, попрошу составить логическую матрицу, в которой перечислить все возможные состояния (благо их всего восемь).

Если то же самое записать на привычном сишном наречии, то получится что-то типа a ? Ь: с.

Здравый смысл торжествует еще раз! Воодушевленные успехом, мы садимся за написание декомпилятора или отладчика. По мне лучше отладчик. Впрочем, нетрудно будет декомпилировать это с карандашом и листком бумаги. Тут я вашу свободу не ограничиваю.

Анализ программы не должен представить особых трудностей, и я его опускаю. Не хочу вам портить последнее удовольствие, да и в любом случае вопрос дизассемблирования программ далеко выходит за рамки данной заметки.

Но на механизме проверки подлинности пароля я все же остановлюсь. Алгоритм до ужаса простой, и для нахождения пароля нужно решить одно • простенькое линейное уравнение. Сам алгоритм на 186 диалекте выглядит так:

```
MOV cx,o
LEA SI .Buffer
Repeat:
MOV DX,CX
XOR CX, [SI]
JMF Repeat
Check:
CMP CX,lstConst
JNZ Nag
CMP DX,2stConst
JNZ Nag
```

Собственно "двойная" проверка обеспечивает дыру достаточных размеров, чтобы в нее можно было без труда пролезть. В самом деле, последние два символа пароля будут равны :

XOR lstConst,2stConst = XOR 7528h, 784Dh = OD65h = *'e*

Пароль едва ли не сам приходит на ум, но мы пойдем другим путем :) Найти подходящий пароль несложно; более того, последовательности в стиле <a хог Ь хог с хог d хог е> описываются едва ли не в каждом букваре! Нетрудно даже скалькулировать необходимое число шагов в наихудшем случае в данной разрядной сетке. Оно слишком мало. Самый криптостойкий пароль вскрывается даже не за 2^ итераций, а гораздо быстрее. Т.е. криптостойкость меньше выбранной разрядной сетки!

Замечу еще раз, что предложенная мной реализация атаки базируется на элементарной алгебре и в принципе должна быть доступна каждому хакеру.

Subj: Как был взломан POCSAG 32

О качестве программы судить воздержусь (потому что не знаю), но как ее взломать (т.е. зарегистрировать), расскажу. Как обычно, нас просят ввести User Name и соответствующий ему код. Не особо напрягаясь, под Windows можно считать содержимое окна редактирования двумя функциями Win32 API GetWin-dowTextA и GetDIgItemTextA. Но, может быть автор использовал уникод? Или кодировал под Win 16? Так на какую же функцию нам поставить брейк-поинт? Первая здравая мысль, пришедшая на ум, — посмотреть таблицу импорта функций user.dll. Воспользуемся тем, что всегда под рукой, — Quick View.

Среди множества импортируемых функций user32.dll GetDIgItemText отсутствует, зато есть GetWindowTextA. Вот на нее мы и поставим Ьрх. Без сюрпризов и неожиданностей мы оказываемся во всплывшем Win-Ice. Теперь самое время узнать адрес буфера строки. Для этого необходимо вспомнить, какие параметры принимает функция. Воспользуемся Visual Studio C++, а точнее, электронной документацией MSDN. Через пару секунд выясняем следующее:

```
int GetWindowText (
HWND hWnd,                // handle of window or control
with text LPTSTR IpString, // address of buffer for text
int nMaxCount     );      // maximum number of characters to copy
```

Таким образом, необходимый нам ipString находится по адресу SS:[ESP+8+8]. Попутно замечаем, что этот адрес совпадает со значением eax. Это очень похоже на Visual C++ (впрочем, и на другие компиляторы Си). Нашу догадку подтверждает копирайт, который легко найти в исполняемом файле. Судя по размеру исполняемого файла, а также по отсутствию импортируемых функций mfc.(Ill, нетрудно предположить, что mfc скомпонована как статичная библиотека. А вот это уже нехорошо. Символьной информации о классах mfc мы не получим. Да и места на винте мне, ей-богу, жалко. Но с другой стороны, функции каркасной библиотеки — это большей частью простые переходники к соответствующим функциям API. Поэтому, подняв хвост, кидаемся в гущу событий. Выполняем следующую последовательность команд:

```
{ dd
d ssiesp+IO
d eax
bpm eax
bd * p ret
be *
x )
```

Убеждаемся, что программа считала введенный нами от балды регистрационный номер. Вскоре мы вываливаемся в очень тривиальный кусок кода, сравнивающий две строки каким-то странным способом. Дамп esi-I покажет нам сгенерированную строку. Аккуратно запишем ее и... Все! Мы зарегистрированные пользователи. А как быть с нашими друзьями? Думаете, им будет по вкусу ваше имя? Ну что же, напишем генератор регистрационных номеров. Сразу предупреждаю, что это на порядок сложнее, чем подсмотреть регнум, не вникая в механизм его генерации.

Возвращаемся к исходной точке. Опять вводим имя и произвольный номер. Нас будет интересовать сам процесс генерации ключа, поэтому теперь мы трапим не регнум, а имя. И дожидаемся кода, который его читает. Им оказывается movsx ebp, byte ptr [esi+eax]. Сразу видно, что сделано со вкусом и размахом. Судя по тому, что идущие подряд команды модифицируют один регистр, оптимизацией тут и не пахнет.

Теперь нам необходимо вникнуть в суть данного фрагмента кода и написать аналогичный для своего генератора. Обратите внимание на цикл в строке Ox40E2EB. Даже беглого взгляда на код достаточно, чтобы понять, что это простая и нестандартная хеш-функция, возвращающая сразу две свертки! (Видно, стандартное CRC32 автору реализовать было лень, впрочем, в данном случае это не страшно.) Пишем собственный код генератора:

```
void KeyGen::GenPolyO(CString Name)
(
unsigned long int poliO = OxADACAFFE;
unsigned long int polil = OxAFFEADAC;
unsigned long int Len    = Hame.GetLength();
unsigned long int tempO;
unsigned long int tempi;
unsigned char     transmit;
for (unsigned int a=0;a<Ьеп;a++)
(
transmit=Name.GetAt(a);
temp=transinit;
temp0= (tempO « ( a & 0x7));
poliO = (poliO ^ tempO);
transmit=llame.GetAt (Len-a-I);
templ=transmit;
templ=(templ « (a & OxF ));
polil= (polil " tempi);
}
TRACEC'lx polyl = %x \n",poliO,polil);
)
```

Просматривая код дальше, мы видим два вызова функции Ox41D150, принимающей полученные свертки. Тут надо полагать, что свертки превратятся в развертки, ибо 64 бита ключа автору защиты показалось мало, и он ухитрился обеспечить работой и себя, и нас.

Заходим внутрь функции развертки. Автор делит свертку на 0x24 и к остатку прибавляет 0x30, если тот меньше 0x9, и 0x57 в противном случае. И так до тех пор, пока в результате не получится ноль. После чего, преследуя одному Богу понятную цель, переворачивает строку?! Пишем еще немного кода.

```
CString KeyGen::GenSeq(unsigned long int poly)
{
CString s0="";
unsigned char    zzz;
while (true)
 (
zzz = (unsigned char) (poly % 0x24);
if (zzz<=9) zzz°zzz+0x30;
else zzz=zzz+0x57;
s0=s0+(char) zzz;
if (! (poly = poly / 0x24)) break;
}
for (int a=0;a<(sO.GetLength()/2);a++)
(
char c=s0[a];
sO.SetAt(a,sO.GetAt(sO.GetLength()-a-l) ) ;
sO.SetAt (sO.GetLengthO -a-l,c) ;
}
TRACEO(sO) ;
return sO;
}
```

Осталось превратить это в готовый продукт. О вкусах и привычках не спорят, поэтому примите Visual C++ не как руководство к действию, а как возможный вариант. Ничем не хуже будет сделать то же на ассемблере, Дельфи и даже на Visual Basic.