жизнь после BSOD

крис касперски и жирный хомяк

опыт - это то, что получаешь, не получив того, что хотел. учение - изучение правил. опыт - изучение исключений.

афоризм

все знают что такое BSOD (он же "голубой экран смерти" — Blue Screen Of Death). это последний вздох операционной системы, после которого она сбрасывает дамп и уходит на перезагрузку, теряя все не сохраненные данные. однако, на самом деле BSOD еще не конец и если перезагрузку заменить на реанимациею, в 9 из 10 случаев можно возвратиться в нормальный режим и успеть зашутданить систему перед тем, как она умрет окончательно

введение

Синий экран появляется всякий раз, когда ядро возбуждает необрабатываемое исключение (скажем, обращение по нулевому указателю) или отлавливает заведомо "левую" операцию (повторное освобождение уже освобожденной памяти, например). А что делает Жирный Хомяк (он же Системный Администратор), когда видит BSOD? Быстро шепчет заклинание: "*Память-видюха-драйвер, на фиг идите от компа, уроды, буду разбираться!*".

Во всех этих случаях управление передается функции **KeBugCheckEx**, описание которой можно найти в NT DDK, и которая завершает работу системы в аварийном режиме, при необходимости сбрасывая дамп памяти, поковырявшись в котором, можно определить причину сбоя.

Функция KeBugCheckEx принимает четыре аргумента, важнейшим из которых является *BugCheckCode*, определяющий причину сбоя. Всего существует свыше сотни кодов ошибок, документированных в DDK (ищите их в руководстве по отладчику "Using Microsoft Debugger"), однако, в действительности их намного больше. Дизассемблирование ядра W2K SP2 показывает, что KeBugCheckEx вызывается из 387 мест! И, преимущественно, с различными параметрами.

Разумеется, не все ошибки одинаковы по своей фатальности. В многоядерных осях это вообще не проблема. Падение одного ядра, не затрагивает других. Все ядра работают в раздельных адресных пространствах и частично или полностью изолированы друг от друга. Разрушать такую систему очень трудно, многоядерная архитектура чрезвычайно устойчива к сбоям, но… как же при этом она тормозит! Межъядерный обмен съедает уйму процессорного времени, а если запихать все компоненты в одно ядро, мы получим… монолитное ядро по типу LINUX'а (что, кстати говоря, явилось причиной яростной критики последнего со стороны многих теоретиков). В LINUX, как впрочем, и в BSD все компоненты ядра (там они называются *модулями*) исполняются в едином адресном пространстве и некорректно написанный модуль может непреднамеренно или умышленно надругаться над чужой собственностью (превратить данные в винегрет, например). Это факт! Однако, при возникновении необрабатываемого исключения в ядре (например, обращения по нулевому указателю), LINUX "грохает" только тот модуль, который это исключение и породил, не трогая все остальные. Аварийный останов системы происходит только по серьезному поводу, когда рушится что-то очень фундаментальное, делающее дальнейшую работу ядра действительно невозможной. Конечно, если "полетел" драйвер жесткого диска — это кранты, но вот, например, без драйвера звуковой карты можно какое-то время и обойтись, сохранив все не сохраненные данные и только потом перезагрузившись.

Операционные системы семейства NT используют гибридную архитектуру, сочетающую сильные стороны монолитных и микро-ядер (так же называемую "архитектурой модифицированного микроядра"), что теоретически, должно обеспечить превосходство над монолитным LINUX'ом (кстати говоря, экспериментальное ядро GNU/HURD построено как раз по микроядерной архитектуре). Легендарно устойчивую NT/XP, которую по слухам можно "уронить" только вместе с сервером, на самом деле очень легко вогнать в голубой экран. Достаточно любому, я повторяю любому, драйверу сделать что-то недозволенное, как система автоматически катапультирует пользователя, заботясь о нем. Хорошо, что Microsoft не строит авиалайнеры!

Рисунок 1 авиалайнер от Microsoft

Если бы было возможно пересесть на HURD! Но увы… совместимость не дает. Вцепилась зубами и не пускает! Далеко не каждый может безболезненно отказаться от своей любимой NT. Так что, не будем сетовать на неизбежность судьбы, а лучше возьмем в руки ассемблер и попытаемся, что-то такое запрограммировать. Что-то такое, что решит все наши проблемы (закопать Билла Гейтста на 640 кб ниже асфальта не предлагать).

Рисунок 2 обычно после BSOD наступает смерть…

>>> врезка чего не умеет NTFS

Для минимализации последствий краха системы, NT поддерживает специальные call-back'и. Всякий драйвер может вызывать функцию KeRegisterBugCheckCallback и зарегистрировать специальный обработчик, который будет получать управление в момент возникновения "голубого экрана". Это позволяет корректно останавливать оборудование, например, парковать головки жесткого диска. Шутка! А вот драйверу файловой системы сбросить свои буфера ничуть не помешало бы, тем более что он может проверить их целостность по CRC или любым другим путем. Ходят устойчивые слухи, что NTFS именно так и поступает. Как бы не так! Мыщъх дизассемблировал NTFS.SYS и не нашел там никаких признаков вызова KeRegisterBugCheckCallback! В момент аварии буфера NTFS остаются не сброшенными и она выживает только благодаря поддержке транзакций, при которых гарантируется атомарность всех операций, т.е. операция либо выполняется, либо нет. Обновление файловой записи не может произойти "наполовину" и потому в отличии от FAT потерянные кластеры на ней не образуются. Ну… практически никогда не образуются.

чем мы будем заниматься

Аварийно завершить работу системы, выбросив синий экран, — самое простое, что только можно сделать при крахе системы. Microsoft неспроста пошла по этому пути, пути наименьшего сопротивления. Мы же покажем, как выйти из "голубого экрана" в нормальный режим, чтобы успеть сохранить все данные еще до того, как система рухнет окончательно. Это довольно рискованный трюк. В случае провала мы можем потерять все, даже наш дисковый том, который потом придется очень долго восстанавливать. Тем не менее, опасность не так уж и велика….

Сначала мы продемонстрируем технику преодоления голубого экрана, руками и головой, а затем напишем специальный драйвер, который будет это делать "автоматом".

Рисунок 3 голубой экран на платном Интернет-телефоне

что нам понадобиться

Все эксперименты мы будем проводить на девственной Windows 2000, без установленных пакетов обновления (остальные системы ведут себя так же, отличаются только адреса, но суть остается прежней). Чтобы ненароком не угробить основную систему, всю работу лучше всего выполнять на эмуляторе типа VM Ware, хотя это и необязательно.

Еще нам потребуется soft-ice (и вы знаете, где его взять), NT DDK (который можно взять там же, где и soft-ice, или приобрести за деньги у Microsoft) и комплект утилит Свена Шрайбера из его книги "Недокументированные возможности Windows 2000", который можно бесплатно скачать с сайта издательства "Питер" или http://irazin.ru/Downloads/BookSamples/Schreiber.zip.

Пиво и сушки выбираются по вкусу.

>>> врезка не шутку, не всерьез

…рискну предположить, что на английский "ядрена вошь" переводится как "kernel bug". Нет, вы только представьте себе — ещё наши (пра)*деды, испытывая недостаток

словарного запаса для излияния переполнявших их эмоций, пускались в нетривиальное обсуждение недостатков программных комплексов! Это ли не наглядное свидетельство генетической предрасположенности русского народа к высоким технологиям?!
(c) Leonid Loiterstein

преодоление голубого экрана с помощью soft-ice

Дождавшись окончания загрузки Windows 2000, мы запускаем драйвер-убийцу w2k_kill.sys, позаимствованный у Шрайбера, специально спроектированный так, чтобы вызывать голубой экран. Разумеется, из командной строки просто так драйвер хрен запустишь! Без загрузчика тут никак не обойтись! (Можно, конечно, прописать драйвер в реестре, но тогда система будет падать при каждом запуске, что в общем-то не входит в наши планы, каким бы коварными они ни были). Воспользуемся динамическиv загрузчиком w2k_load.exe, разработанным все тем же Шрайбером — w2k_load.exe. (NT поддерживает динамическую загрузку драйверов, но готовой утилиты в штатный комплект поставки не входит — все в духе Microsoft, а вот в LINUX с этим проблем нет).

Набираем в командной строке "w2k_load.exe w2k_kill.sys" и… система клеит ласты и падает в синий экран (кстати говоря, большую коллекцию голубых экранов от торговых автоатов до терминалов аэропорта можно найти на сайте http://www.dognoodle99.cjb.net/bsod/, после просмотра которого становится очень грустно, так грустно, что даже жить не хочется, даже после BSOD):

Рисунок 4 голубой экран, вызываемый драйвером-убийцей

Так происходит потому, что в процессе инициализации драйвера-убийцы выполняются следующие строки, обращающиеся к нулевой ячейке памяти, что строго-настрого запрещено:

```
NTSTATUS DriverEntry (PDRIVER_OBJECT  pDriverObject, PUNICODE_STRING pusRegistryPath)
{
        return *((NTSTATUS *) 0);
}
```

Листинг 1 фрагмент драйвера-убийцы, пытающийся прочитать двойное слово по нулевому указателю из режима ядра

Ну и на фига было ронять систему из-за такой ерунды?! Кому наш "убийца" реально мешает?! Ведь целостность системы ничуть не пострадала! Как объяснить этой тупой NT, что в Багдаде все спокойно, ни один мыщъх не пострадал, хорош отлынивать! Пора бы вернуться в user-mode и продолжить работу в штатном режиме.

Если soft-ice был заблаговременно запущен, он отловит это исключение и покажем свой экран, передавая нам все бразды правления (см. рис. 5).

Рисунок 5 soft-ice отлавливает большинство фатальных и не фатальных исключений, возникающих как на прикладном уровне, так и в режиме ядра

Если нажать "x" (или <Ctrl-D>), то немедленно после выхода из soft-ice вспыхнет синий экран и тогда чинить будет уже нечего. Но пока мы в находимся в soft-ice еще можно кое-что предпринять. А предпринять можно следующее:

- определить место сбоя (в нашем случае это обращение по нулевому указателю), исправить ситуацию (установить валидный указатель), вручную выйти из обработчика исключения, вернув CS:EIP на прежнее место: способ хороший, но увы — не универсальный и требующий определенного интеллекта, которого у машины нет;
- зациклить текущий поток к ядреной фене, воткнув в свободное место jmp $ и выйти из отладчика, разрешив прерывания командной r fl=I (если они вдруг были запрещены) — все будет ужасно тормозить, но ось продолжит работать и мы по крайней мере сможем корректно завершить ее работу;
- дождаться вызова функции KeBugCheckEx и сразу же выйти из нее, проигнорировав сбой и продолжив нормальное выполнение, правда, никаких гарантий, что система не рухнет окончательно, у нас нет;
- способ предложенный ms-rem — дикий, но иногда работающий: отдать команды r eip=0/r cs=1B, переключающие процессор на прикладной режим;

Короче, вариантов много. Можно сказать, толпа. В глазах рябит и разбегается, вызывая оживленное подергивание где-то в области хвоста. Ладно, попробуем для начала воспользоваться первым способом. Мы знаем, что в данном случае, авария произошла из-за ошибки нарушения доступа (в другом случае мы этого знать не будем). Следовательно, процессор возбудил исключение, забросил на вершину стека EIP/CS/FLAGS и передал управление обработчику исключений внутри которого мы сейчас и находится (примечание: иногда по не совсем понятной причине, soft-ice останавливается не на первой команде обработчика исключений, а на непосредственно на месте самого сбоя. Под VM Ware первый раз soft-ice 2.6 всегда останавливается в обработчике, а все последующие разы — на месте сбоя. Эффект сохраняется вплоть до перезапуска VM Ware).

Даем команду "d esp" для отображения содержимого стека и видим (примечание: для удобства рекомендуется переключить окно дампа в режим двойных слов, воспользовавшись командой "dd"):

```
:d esp
0010:F7443C88 BE67C000  00000008  00200202  804A4431      ..g....... .1DJ.
0010:F7443C98 81116AD0  8649D000  BE8F1D08  BE8F1D08      .j....I.........
0010:F7443CA8 81480020  F7443D34  745FFFFF  83A49E60       .H.4=D..._t`...
```

Листинг 2 содержимое стека на момент возникновения сбоя

Адрес инструкции, возбудившей исключение, лежит в первом двойном слове — BE67C000h (у вас это значение наверняка будет другим). Селектор CS идет следом. У всех нас он должен быть равен 08h. Третье двойное слово хранит содержимое регистра флагов — EFLAGS.

Теперь мы знаем место сбоя и можем вывести дизассемблерный листинг на экран. В этом нам поможет команда "u *esp" (дизассемблировать содержимое памяти по адресу, лежащему по адресу, содержащемуся в регистре esp) или "u be67c000":

```
:u *esp
0023:BE67C000  MOV             EAX,[00000000]
0023:BE67C005  RET     0008
0023:BE67C008  NOP
0023:BE67C009  NOP
0023:BE67C00A  NOP
0023:BE67C00B  NOP
```

Листинг 3 определение реального места сбоя

Рисунок 6 soft-ice показывает инструкцию, возбудившую исключение, которые при обычных обстоятельствах ведет к голубому экрану, затем наступает чернота… в которой ни черта не видно. смерть!

Вот она! Инструкция, вызвавшая сбой! А давайте ее "перепрыгнем", продолжив выполнение с RET 08h? Сказано — сделано. Но для начала нужно выйти из обработчика исключения. Для этого в soft-ice необходимо выполнить следующие команды:

- r eip = *esp + sizeof(mov eax,[0]); // устанавливаем регистр EIP на RET
- r cs = *(esp + 4); // устанавливаем селектор CS (не обязательно)
- r FL = I; // разрешаем прерывания;
- r esp = esp + C // снимаем со стека 3 дв.слова, заброшенные туда CPU
- x // выходим из отладчика

После выполнения этой "магической" последовательности команд, система продолжит свою нормальную работу и синий экран уже не появится. Фантастика! Невероятно! Мы только что избежали гибели, которая еще мгновение назад казалась неотвратимой!

Один маленький нюанс. Моя (и возможно ваша) версия soft-ice не умеет восстанавливать регистр ESP в обработчике исключения. Отладчик игнорирует команду r esp=esp +C, на самом деле только имитируя ее выполнение! А это значит, что стек остается несбалансированным и несмотря ни на какие усилия "медиков", система все-таки грохается. Приходится хитрить. Мы видим, что за RET 08h расположена длинная цепочка NOP'ов. А что если воткнуть сюда команду "ADD ESP,0Ch", чтобы стек сбалансировал сам процессор?

Говорим отладчику 'A BE67C008' (ассемблировать начиная с адреса BE67C008) и вводим следующие ассемблерные инструкции: ADD ESP,0C<ENTER>JMP BE67C005<ENTER> и еще один <ENTER> для завершения ввода. Переустанавливаем EIP на начало нашей "заплатки" — r eip =BE67C008 и... выходим из soft-ice. На этот раз у нас все получается!

На всякий случай, вот последовательность команд по реанимации системы. Напоминаю, что она применима только в данном частном случае:

```
u *esp
r eip = *esp
r eip = eip + 9
a eip
add esp,0c
jmp BE67C005h  ; адрес команды RET 8, в вашем случае будет другим
<ENTER>
r fl=I
x
```

Листинг 4 реанимация системы в условиях, приближенных к боевым

Рисунок 7 лучшая реклама товару — синий экран

автоматизируем нашу работу

Способ "ручного" восстановления, только что описанный выше, хорошо сочетается с духом системных программистов, постоянно пасущих soft-ice и умеющих фехтовать регистрами как рапирой. А вот простым смертным такой подход смерти подобен. Но почему бы нам не написать утилиту, зацикливающую сбойный поток или накоротко замыкающую KeBugCheckEx?

Написать-то такую штуку несложно (и мы действительно напишем ее), но… это все равно, что подложить полено под аварийный клапан. Если система пойдет в разнос, ее уже ничего не остановит и… как рванет! Хвост даже по запчастям не соберешь! Может пострадать даже файловая система (пусть это будет хоть NTFS). Пускай вероятность такой трагедии крайне мала, она все-таки возможна — имейте это ввиду. Тем не менее, рискнуть все-таки стоит, особенно в тех случаях, когда вы уверены, что это можно сделать.

Вот, например, возник у меня как-то конфликт между криво написанным драйвером DSL-модема и драйвером видеокарты из-за чего при просмотре видео иногда выскакивает BSOD. Поскольку, нормальных дров найти не удалось, я временно ограничился тем, что "закоротил" KeBugCheckEx перемычкой, изготовленной из команды JMP и… это "прижилось"!

Проведем следующий эксперимент. Нажмем <Ctrl-D> для вызова soft-ice, установим точку останова на KeBugCheckEx и… запустим наш драйвер-убийцу. Причем, точка останова обязательно должна быть аппаратной ("bpm KeBugCheckEx X"), а не программной (bpx KeBugCheckEx) иначе ничего не получится.

На этот раз вместо сообщения о ошибке страничного доступа, soft-ice всплывает по срабатыванию точки останова, высвечивая курсором первую команду функции KeBugCheckEx, которая в нашем случае располагается по адресу 8042BF14h.

Рисунок 8 перехват голубого экрана командой bpm "KeBugCheckEx X" в soft-ice

Прокручивая окно дизассемблера вниз, находим первую инструкцию "RET 14h" (в нашем случае она располагается по адресу 8042C1E9h). Это и есть команда выхода из функции на которую нужно сделать jmp. Для быстрого поиска можно попросить soft-ice сделать search ("s eip l -1 C2,14,00").

Говорим отладчику "r eip = 8042C1E9" (у вас адрес скорее всего будет другим) и давим на <Ctrl-D> для выхода. Отладчик всплывает повторно, в той же самой функции. У нас ничего не получилось?! Не торопитесь с выводами! Все идет по плану! Игнорирование критических ошибок вызывает целый каскад вторичных исключений, что в данном случае и происходит. Повторяем нашу команду "r eip = 8042C1E9" (для этого достаточно нажать стрелку вверх/<ENTER>) и… система возвращается в нормальный режим! Третий раз отладчик уже не всплывает. Мышь немного тормозит, однако, гонять ее по коврику вполне возможно.

Приступаем к созданию драйвера, который будет все это делать за нас. Для начала нам понадобится скелет. Выглядит он так:

```
.386                    ; использовать команды .386 ЦП
.model flat, stdcall    ; плоская модель памяти, stdcall-вызовы по умолчанию

 .code                  ; секция кода

DriverEntry proc        ; точка входа в драйвер

        ; код "драйвера"
        ;
        …
        …
        …
        ; возвращаем ошибку конфигурации
        mov eax, 0C0000182h; STATUS_DEVICE_CONFIGURATION_ERROR
        ret             ; выходим
DriverEntry endp

end DriverEntry
```

Листинг 5 скелет "псевдодрайвера", не управляющий никакими устройствами, но позволяющий нам выполнять код на уровне ядра

На самом деле это не совсем драйвер. Он не принимает никаких IRP-пакетов, не обслуживает никаких устройств и вообще не делает ничего, только загружается и выгружается. Но для нашей затеи этого будет вполне достаточно!

Весь код сосредоточен внутри процедуры DriverEntry – своеобразном аналоге функции main языка Си, которая выполняется при попытке загрузки драйвера, инициализируя все, что необходимо. Отсюда можно "дотянуться" до функции KeBugCheckEx и модифицировать ее по своему усмотрению.

Несмотря на то, что процедура DriverEntry выполняется на уровне ядра с максимальными привилегиями, попытка "правки" машинного кода приводит к нарушению доступа. Это срабатывает защита от непреднамеренного хака ядра некорректным драйвером. Как ее отключить?

Путь первый — через реестр. Создаем в разделе HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\Memory Management значение типа REG_DWORD с именем EnforceWriteProtection и значением 0 (это можно делать и с прикладного уровня). Все! Запись в ядро открыта! Кстати говоря, soft-ice именно так и работает.

Путь второй — репаминг страниц. Отображаем физический адрес страницы, в которой лежит KeBugCheckEx на виртуальное адресное пространство своего процесса посредством вызова функции NtMapViewOfSection, назначая все необходимые нам права. Репаминг осуществляется исключительно на уровне ядра, но к отображенной странице можно обращаться даже из прикладного уровня. Красота! По этой схеме работают многие брандмауэры и другие программы, нуждающиеся в перехвате ядерных функций, например, rootkit'ы. Подробности здесь: http://www.stanford.edu/~stinson/misc/curr_res/nt_hooking.txt.

Путь третий — сброс флага WP в регистре Cr0. Это достаточной грязный трюк с целой свитой противопоказаний и рекламаций, однако, для наших целей он вполне подходит. Используем его как самый простой и быстрый вариант, умещающийся всего в 3 (!) машинных команды:

```
mov     eax, cr0        ; грузим управляющий регистр cr0 в регистр eax
and     eax, 0FFFEFFFFh; сбрасываем бит WP, запрещающий запись
mov     cr0, eax        ; обновляем управляющий регистр cr0
```

Листинг 6 код, отключающий защиту ядра от записи

Соответственно, чтобы включить защиту, этот самый бит WP нужно установить, что и делают следующие машинные команды:

```
mov     eax, cr0        ; грузим управляющий регистр cr0 в регистр eax
or      eax, 10000h     ; сбрасываем бит WP, запрещающий запись
mov     cr0, eax        ; обновляем управляющий регистр cr0
```

Листинг 7 код, включающий защиту ядра

"Политически корректная" программа должна не просто отключать/включать защиту от записи, а запоминать текущее состояние бита WP перед его изменением, а затем восстанавливать его обратно "как было", иначе можно непроизвольно включить защиту в самый неподходящий момент, серьезно навредив, вирусу или rootlit'у. Только он ее того, как вдруг она!

"Закоротить" функцию KeBugCheckEx можно разными путями. Самое правильное (и надежное!) определить ее адрес путем разбора таблицы импорта, но это слишком долго, муторно, нудно, да еще и утомительно. Гораздо проще подставить готовые адреса, жестко прописав их в своей программе. Минус этого решения в том, что на других компьютерах она работать не будет. Стоит установить (или удалить) какой-то ServicePack, перейти на другую версию системы как все адреса тут же изменятся и произойдет сплошной завис. Тем не менее, имея исходные тексты драйвера под рукой его всегда можно исправить и перекомпилировать. Так что для "домашнего использования" такое решение вполне допустимо.

Главная тонкость в том, что мы не должны трогать первый байт функции KeBugChekEx, поскольку его уже потрогал soft-ice и весь вытрогал. Так же поступают и другие хакерские программы (например, API-шпионы), помещая сюда команду INT 03 (опкод CCh), предварительно сохранив прежнее содержимое где-то в другом месте.

ОК, пропустим первую команду (в нашем случае это PUSH EBP) к едреням и начнем внедрение со второй. Чтобы сбалансировать стек в противовес PUSH EBP говорим POP EAX, а затем либо jmp на RET 14h, либо непосредственно сам RET 14h. Последний вариант короче, да к тому же элегантнее. Реализуется он так:

```
mov dword ptr DS:[8042BF14h+1], 14C258h
```

Листинг 8 код, "закорачивающий" KeBugCheckEx

Здесь: 8042BF14h — адрес начала функции KeBugCheckEx (на всех машинах разный), 1 – длина инструкции PUSH EBP, а 14C258h — машинный код, представляющий собой последовательность двух команд: POP EAX (58h)/RET 14h (C2h 14h 00h).

Объединив все компоненты воедино мы получаем следующий папелац:

```
.386
.model flat, stdcall
.code
DriverEntry proc
        mov     eax, cr0                ; грузим управляющий регистр cr0 в регистр eax
        mov ebx, eax            ; сохраняем бит WP в регистре ebx
        and     eax, 0FFFEFFFFh         ; сбрасываем бит WP, запрещающий запись
        mov     cr0, eax                ; обновляем управляющий регистр cr0

        mov dword ptr DS:[8042BF14h+1], 14C258h 14C258
                                                ; "закорачиваем" KeBugCheckEx

        mov     cr0, ebx                ; восстанавливаем бит WP
        mov eax, 0C0000182h; STATUS_DEVICE_CONFIGURATION_ERROR
        ret
DriverEntry endp
end DriverEntry
```

Листинг 9 средство против BSOD, перед употреблением встряхнуть

Вот такой маленький драйвер, а сколько данных он может спасти! Остается только откомпилировать его и можно приступать к испытаниям:

```
ml /nologo /c /coff nobsod.asm
link /driver /base:0x10000 /align:32 /out:nobsod.sys /subsystem:native nobsod.obj
```

Листинг 10 ключи ассемблирования и линковки (используется пакет MASM из NT DDK)

Если все было сделано правильно, на диске образуется файл nobsod.sys, который мы загрузим с помощью динамического загрузчика w2k_load. Загрузчик конечно, заругается матом, что мол ERROR и драйвер ни хрена не грузится, но так и должно быть. Все нормально! Мы же возвратили код STATUS_DEVICE_CONFIGURATION_ERROR!

Внимание: под VM Ware такой трюк не срабатывает, поскольку она не полностью эмулирует регистр cr0 и таких шуток в упор не понимает, вызывая завис гостевой оси. В этом случае можно закомментировать все строки, относящиеся к регистру cr0 и отключить защиту через реестр, создав соответствующий ключ "Редактором Реестра". Кстати говоря, если на целевой машине установлен soft-ice – такой ключ уже создан и ничего делать не надо.

Рисунок 9 предсмертное сообщение VM Ware, которая она часто выдает, когда над ней проводят разные эксперименты, на которые она не была рассчитана

Загрузим драйвер-убийцу, чтобы проверить справиться ли с ним наше средство против BSOD или нет… soft-ice (если он установлен) несколько раз всплывает. Вот зануда! Гоните его прочь, нажимая x или <Ctrl-D>. Но так или иначе, голубой экран уже не появляется! Система жутко тормозит, но все-таки работает. И это — главное!

Плохо то, что теперь NT никак не может сигнализировать, что произошел системный сбой и что нужно побыстрее сматывать ласты, совершая shutdown. То есть почему это не может сигнализировать?! Самое простое — добавить в нашу "заплатку" на KeBugCheckEx несколько ассемблерных строк, которые "бибикнут" спикером или сыграют "семь-сорок" (как вариант "во поле береза стояла") на динамике. В принципе, можно даже разделить BugCheck коды на категории, каждой из которой будет соответствовать свое число гудков. За примерами далеко ходить не надо. Их можно выдрать из любого DOS-вируса. Техника программирования спикера на уровне ядра та же самая и она ничуть не изменилась.

Да много что можно сделать! Главное — фантазию иметь!

>>> врезка исключение и наказание

Всегда ли помогает "шунтирование" KeBugCheckEx? Насколько это безопасно? Это очень, очень опасно и помогает далеко не всегда. Вот, например, рассмотрим следующий пример кода, позаимствованный из ядра:

```
00565201       call    ExAllocatePoolWithTag  ; выделение памяти из лужи
00565206       cmp     eax, ebx               ; проверка успешности выделения памяти
00565208       mov     ds:dword_56BA84, eax
0056520D       jnz     short loc_56521C       ; -> нам дали память! живем, мужики!
0056520F       push    ebx                    ; \
00565210       push    ebx                    ; +
00565211       push    6                      ; +- с памятью вышел облом
00565213       push    5                      ; +- отправляемся на небеса
00565215       push    67h                    ; +
00565217       call    KeBugCheckEx           ; /
0056521C loc_56521C:                          ; CODE XREF: sub_5651C1+4Cj
0056521C       lea     eax, [ebp+var_C]       ; продолжаем нормальное выполнение
0056521F       push    ebx
00565220       push    eax
```

Листинг 11 фрагмент кода, при котором "шунтирование" KeBugCheckEx заканчивается очень печально

Система выделяет память из общего пула (в шутку называемого лужей) и если с памятью не облом, происходит нормальное продолжение, в противном случае вспыхивает голубой экран и хана. Допустим, мы "закоротили" KeBugCheckEx, что тогда? Нас обломали на память, а мы продолжаем нормальное выполнение как ни в чем не бывало, обращаясь по указателю, который указывает в никуда. Возникает целый каскад вторичных исключений, а все структуры данных превращается в труху и система рушится окончательно. Вот так.

заключение

Мы пережили самую страшную систему катастрофу — BSOD, после которой нам все по плечу! Конечно, неразумно практиковать такой подход на сервере, но для рабочих станций он вполне приемлем. Проверено на мыщъх'иной шкуре! Кстати говоря, некоторые вирусы, черви и rootkit'ы используют схожую технику для маскировки своего присутствия в системе. Некорректно написанный вирус может вызвать синий экран и в системном журнале появится соответствующая запись, помогающая администратору разобраться с проблемой. Если же "перемкнуть" KeBugCheckEx, то компьютер будет просто беспричинно тормозить (или виснуть), но в журнале ничего не появится!

Рисунок 10 дзенский сад голубых экранов смерти