exploits review III

крис касперски ака мыщъх, no-email

дыра в AMD K7/K8

brief: 18 апреля 2006 года обнаружилась дыра в процессорах AMD K7/K8 позволяющая одному процессору заглянуть внутрь сопроцессорного контекста другого процесса, что ведет к утечке данных и упрощает атаку на криптографические системы: www.securityfocus.com/bid/17600;
для быстрого переключения контекста Linux и BSD-системы используют пару команд `fxsave/fxrstor`, сохраняющих/восстанавливающих регистры сопроцессора в/из оперативной памяти. коварство `fxrstor` заключается в том, что она сохраняет указатель команд (`FIP`), указатель данных (`Data Pointer`) и опкод последней инструкции _только_ в том случае, если бит ES (exception summary) в статусном слове сопроцессора x87 установлен. при переключении контекста ось не обнуляет эти данные и они становятся доступны "посторонним" процессам: kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.16.9;

targets: строго говоря, это не ошибка, а документированная особенность процессоров AMD, на которую прежде никто не обращал внимания, поскольку с процессорами Intel в этом плане все OK. уязвимость затрагивает _все_ LINUX и BSD системы, а, возможно, и Windows.

exploit: для реализации атаки exploit'a не требуется, достаточно просто выполнять команду `fxsave` в цикле, надеясь, поймать что-то интересное.

solution разработчики LINIX/BSD уже выпустили патчи: securityfocus.com/bid/17600/solution, очищающее ES-бит и загружающие фиктивный double на стек сопроцессора. вот фрагмент заплатки для BSD:

```
static double dummy = 0.0;
static fpu_clean_state()
{
        u_short status;fnstsw(&status);
        if (status&0x80) fnclex();
        __asm __volatile("ffree %%st(7);fld %0" : : "m" (dummy));
}
```

Byte	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+16	Reserved				MXCSR				Reserved		DS		Data Pointer			
+0	Reserved		CS		Floating-Point EIP				FOP		FTW		FSW		FCW	

первые 32-байта данных сохраняемых/восстанавлимаемых инструкциями FXSAVE/FXRSTOR (32-битный формат)

Byte	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
+16	Reserved				MXCSR				Data Pointer							
+0	Floating-Point RIP								FOP		FTW		FSW		FCW	

первые 32-байта данных сохраняемых/восстанавлимаемых инструкциями FXSAVE/FXRSTOR (64-битный формат)

рисунок 1 значения EIP/RIP, FOP и Data Pointer сохраняются только, если бит ES установлен

дыра в AMD x86-64

brief: Jari Kirma — один из разработчиков FreeBSD — обратил внимание, что на AMD x86-64 непривилегированный пользователь может получить непосредственный доступ к оборудованию с прикладного уровня. это объясняется тем, что x86-64 имеет два механизма разделения привилегий: код, исполняющий на уровне ядра, имеет доступ ко всем портам ввода/вывода и обычно является "посредником" между железом и user-mode, что не всегда удобно, поэтому, процессор поддерживает специальную карту, позволяющую "открыть" часть портов, разрешив к ним доступ с прикладного уровня (подобная карта имеется у Intel'a, но там она по умолчанию заблокирована)

суть в том, что вплоть до FreeBSD/amd64 5.4-RELEASE эта таблица инициализировалась _неправильно_ образуя огромную дыру в системе безопасности. злоумышленник (или некорректно работающий код) может вызывать отказ в обслуживании, разрушать или похищать информацию и т. д., подробнее об этом можно прочитать на: www.security.freebsd.org/advisories/FreeBSD-SA-05:03.amd64.asc.

targets: уязвимость затрагивает только операционную систему FreeBSD версии 5.4 или ниже, работающую на платформе AMD x86-64. на все остальные системы эта дыра не распространяется;

exploit: для реализации данной уязвимости exploit не требуется, достаточно установить обработчик исключений и последовательно перебрать все порты, определяя какие из них доступны на запись/чтение, а какие нет.

solution существует несколько решений этой проблемы: например, обновить систему до версии 5-STABLE или наложить заплатку, которую можно скачать с сервера ftp://ftp.FreeBSD.org/pub/FreeBSD/CERT/patches/SA-05:03/amd64.patch, после чего перекомпилировать ядро.

рисунок 2 AMD-64 собственной персоной

дыра в привилегированном процессе winlogon.exe

brief: 8 августа 2006 года сразу два специалиста Reed Arvin из Canaudit, Inc и Matt Miller из Leviathan Security Group обратили внимание на то, что процесс winlogon.exe (ответственный за регистрацию пользователей в системе, блокировку компьютера и т. д.) начинает поиск динамических библиотек с домашнего каталога пользователя и только потом переходит к системному каталогу Windows, поэтому любой пользователь может легко повысить свои привилегии до SYSTEM, если положит в свой домашний каталог "заряженную" DLL. удаленная атака легко реализуется через зловредную web-страничку, запрашивающую имя пользователя и пароль. IE их высылает автоматически и, если компьютер допускает удаленные подключения (что характерно для серверов), злоумышленнику достаточно закинуть DLL в HOME, выйти из системы и войти еще раз. подробности тут: www.microsoft.com/technet/security/Bulletin/MS06-051.mspx;

targets: уязвимости подвержены: Windows 2000, XP SP1, XP SP 2, Server 2003 и Server 2003 SP1.

exploits: для реализации данной атаки exploit не требуется.

solution Microsoft уже выпустила заплатки для всех уязвимых системы, выложив их на download-сервер. если же, по каким-то причинам он недоступен, можно воспользоваться альтернативным решением, задействовав режим безопасного поиска

динамических библиотек. для этого необходимо запустить редактор реестра (regedt32.exe), открыть следующую ветвь реестра HKLM\SYSTEM\CurrentControlSet\Control\Session Manager и добавить значение "SafeDllSearchMode" типа DWORD, установленное в 1, после чего перезагрузить машину.

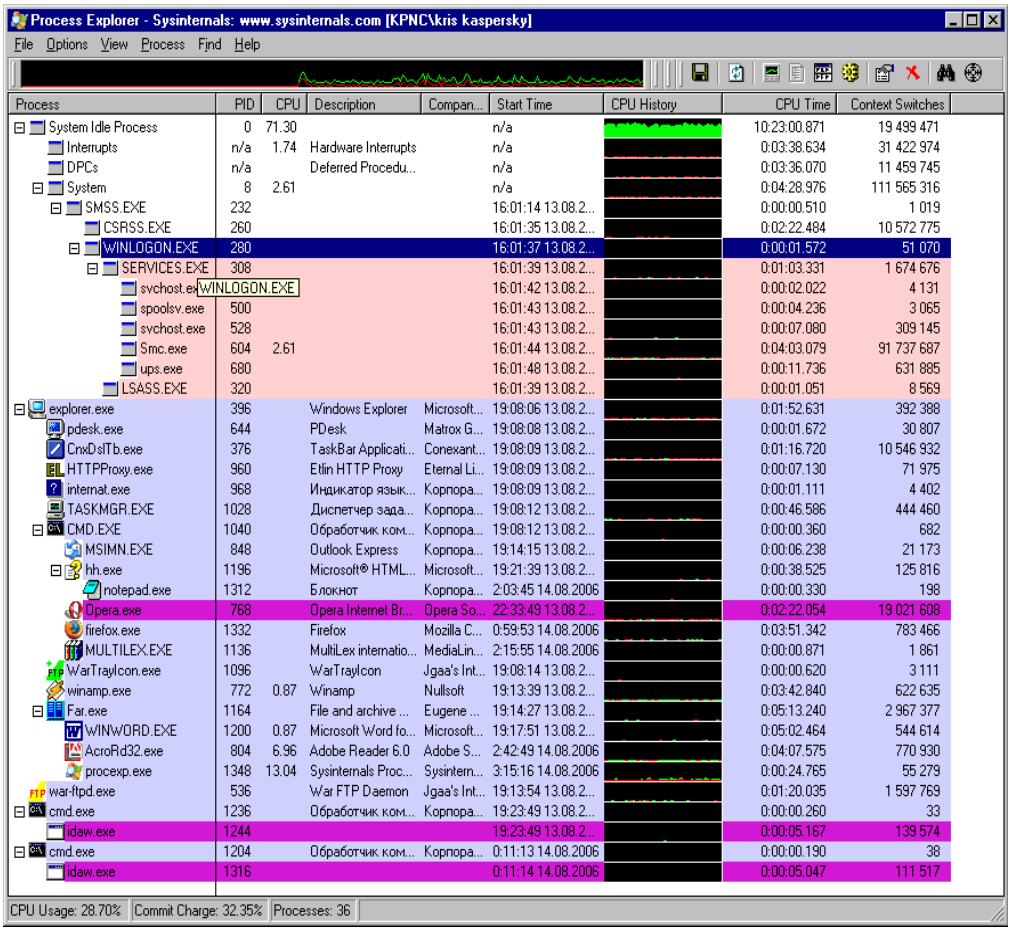

рисунок 3 процесс winlogon.exe по умолчанию ищет динамические библиотеки в домашней директории пользователя

full disclose
Microsoft Remix: или новая дыра в RPC

brief: за Microsoft уже закрепилась устойчивая репутации компании, никогда не исправляющий крупные ошибки с первого раза. вот и сейчас, когда эпидемия MSBLAST еще свежа в памяти (и мутированные черви до сих пор бродят по Сети), в RPC (Remote Procedure Call – механизм удаленного вызова процедур) обнаружилась новая ошибка переполнения, допускающая удаленную засылку shell-кода с захватом управления на правах SYSTEM.

мы не знаем, кто первый обнаружил ошибку, но 8 августа 2006 года Американское Общество US-CERT (United States Computer Emergency Readiness Team) и Калифорнийский Институт SANS (SysAdmin, Audit, Network, Security) практически одновременно выслали свои раппорты в Microsoft. публичный пресс-релиз (www.kb.cert.org/vuls/id/650769) не раскрывал никаких технических деталей, но несмотря на это, 10 августа уже появился рабочий exploit, являющийся частью проекта Metasploit Framework: www.metasploit.com.

Microsoft присвоила уязвимости критический уровень безопасности и в тот же день выпустила бюллетень MS06-040: microsoft.com/technet/security/Bulletin/MS06-040.mspx.

Рисунок 4 стартовый экран Metasploit Framework

targets: Microsoft занесла в список уязвимых систем Windows 2000, XP SP1, XP SP2, Server 2003 и Server 2003 SP1, в то время как создатели Metasploit Framework exploit'a претендовали на "поддержку": NT 4.0, Windows 2000 SP0-SP4, XP SP0-SP1, подчеркивая что удаленное выполнение shell-кода на XP SP2/Windows 2003 SP1 невозможно и максимум, что можно устроить это — отказ в обслуживании. парни из Microsoft Security Response Center Blog провели свое собственное расследование и выяснили, что удаленное выполнение кода возможно _только_ на **Windows 2000** и **XP SP 1**. им не удалось атаковать ни XP SP 2, ни Server 2003, ни Server 2003 SP 1: blogs.technet.com/msrc/archive/2006/08/11/446078.aspx (однако следует помнить, что между "не удалось атаковать" и "атаковать невозможно" огромная разница!).

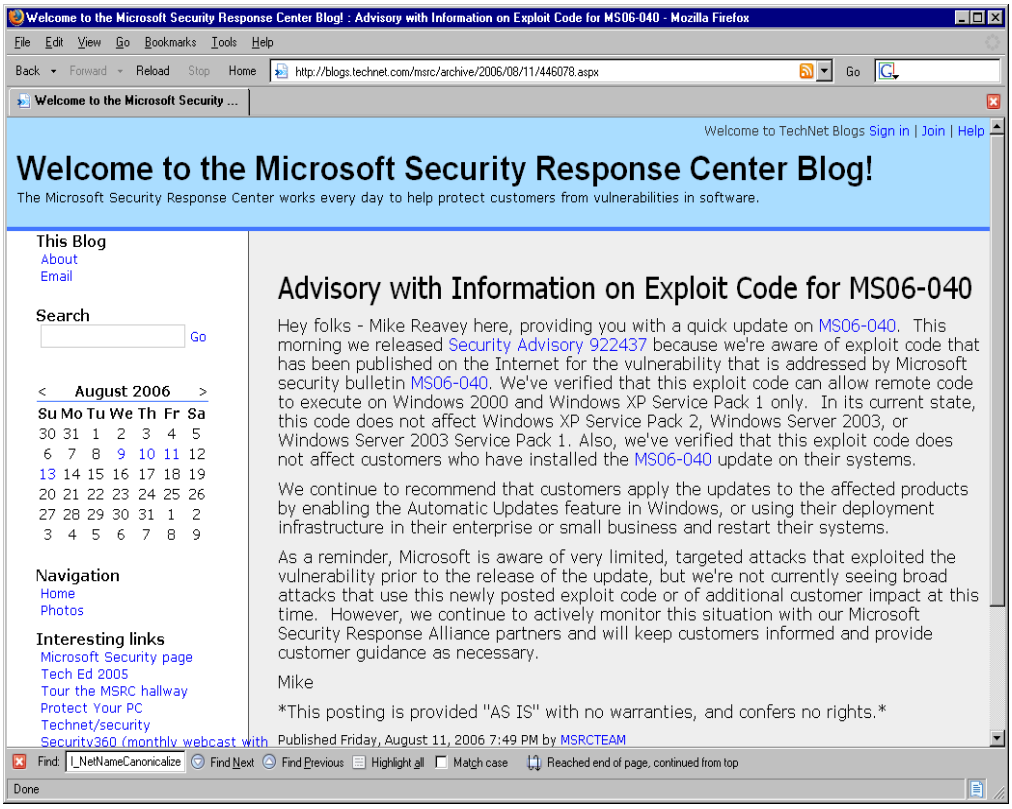

рисунок 5 блог команды Security Response Center

exploit: готовый exploit (написанный на языке Perl) можно свободно скачать по адресам: http://metasploit.com/projects/Framework/exploits.html#netapi_ms06_040 и http://milw0rm.com/exploits/2162;

Рисунок 6 exploit MS Windows NetpIsRemote() Remote Overflow на сайте MilwOrm

solution Microsoft уже выпустила заплатки для большинства своих систем, доступные через службу Windows Update, однако, при желании можно обойтись и без них: достаточно заблокировать SMB-трафик из внешней Сети, для чего необходимо закрыть два TCP порта — 139 и 445 (однако, это сделает невозможность работу приложений, работающих через SMB).

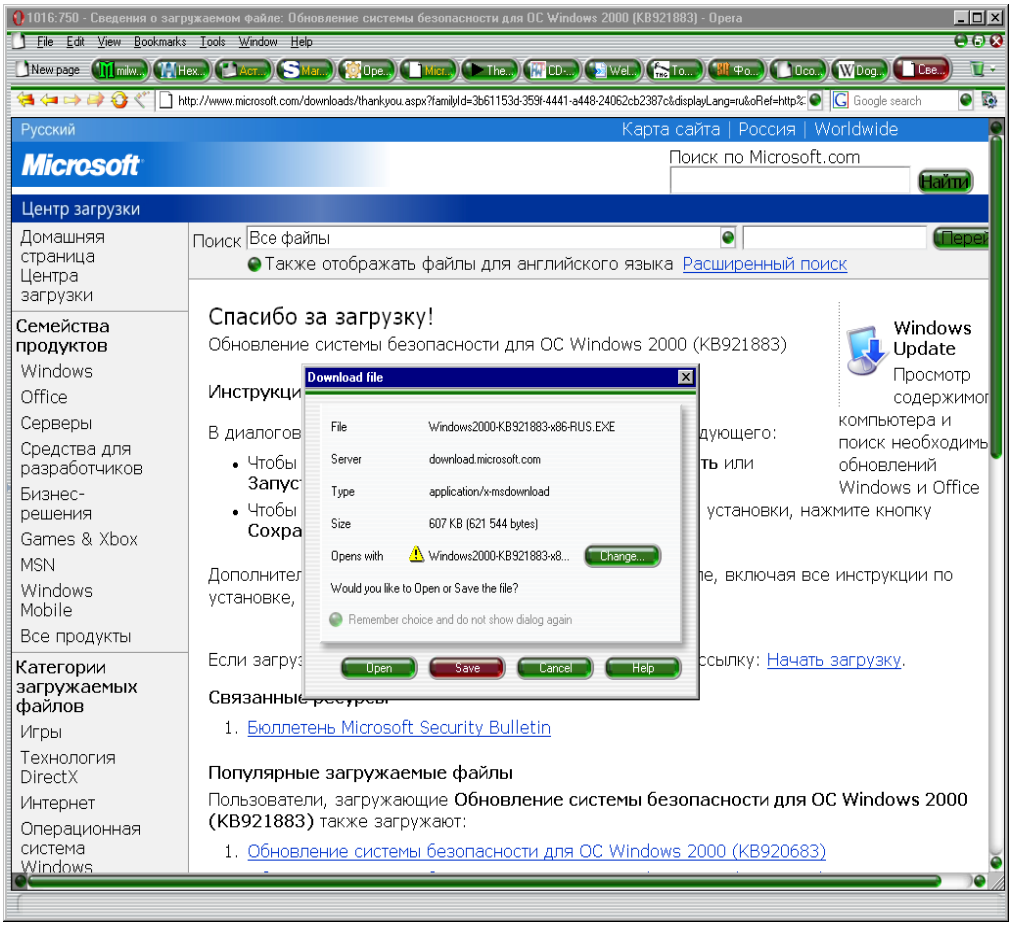

рисунок 7 тянем заплатку с сервера Microsoft

details: механизм RPC является низкоуровневым средством межкомпьютерного взаимодействия, предоставляющим прозрачный механизм удаленного вызова процедур, который с точки зрения прикладной программы выглядит так, как будто бы процедура находится на локальной машине.

на базе RPC построены многие системные сервисы и, в частности, SMB (Server Message Block), реализующий именованные каналы (named pipes) и использующий PRC в качестве транспорта. в свою очередь, SMB используется для удаленного доступа к файлам, папкам и принтерам.

и все бы ничего, но в функции **`NetpIsRemote()`**, сосредоточенной в библиотеке NetApi32.DLL допущена ошибка контроля границ, приводящая к традиционному стековому переполнению, с возможностью подмены адреса возврата. на системах с активным DEP, предотвращающим выполнение кода в неисполняемых областях памяти, приходится хитрить, атакуя жертву по одному из сценариев, описанных в статье "переполнение буфера на системах с неисполняемым стеком", которую можно найти на прилагаемом к журналу диске или скачать с мыщъх'ого ftp://nezumi.org.ru (напоминаю, что сервер установлен на рабочей машине и потому доступен не все время, а только тогда, когда мыщъх шевелит хвостом).

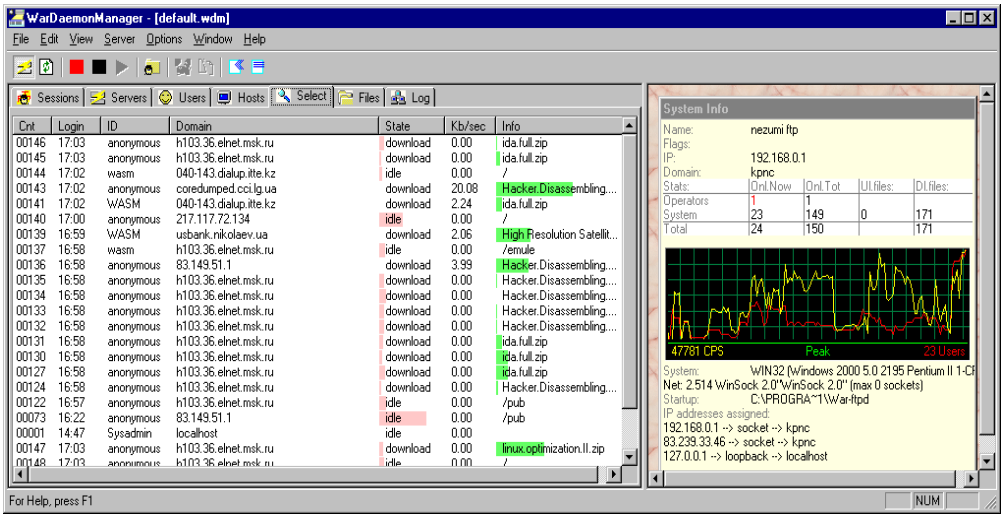

Рисунок 8 мыщъх'иный ftp на раздаче

если бы `NetpIsRemote()` была документированной функций, мы бы просто передавали ей аргументы различной длинны, пытаясь вызывать переполнение, но, к сожалению, ее прототип неизвестен, а потому приходится прибегать к тяжелой артиллерии, то есть к дизассемблированию.

загружаем библиотеку NetApi32.DLL в IDA Pro и начинаем исследовать функцию `NetpIsRemote()`, пытаясь "глазами" найти место, в котором происходит переполнение, в первую очередь обращающая внимания на циклы, копирующие блоки памяти (типа `mov ecx,[eax]/mov [ebx],eax/add eax,4`) или вызовы функций `memcpy()`, `memmove()`, `strcpy()`, `wcscpy()` и т. д.

функция `NetpIsRemote()` выглядит обманчиво маленькой, но если присмотреться повнимательнее, можно обнаружить множество условных переходов, ведущий на CHUNK'и (то есть, на ее продолжение), совокупный объем которых довольно велик и затруднителен для анализа.

```
File  Edit  Jump  Search  View  Debug  Options  Window                    ↓ AU: idle   READY
[•]                                          IDA View-A
.text:7CD17ADB NetpIsRemote    proc near               ; CODE XREF: I_NetNameCanonicalize+98↓p
.text:7CD17ADB                                         ; I_NetPathCanonicalize+B8↓p ...
.text:7CD17ADB
.text:7CD17ADB var_414         = dword ptr -414h
.text:7CD17ADB var_20C         = dword ptr -20Ch
.text:7CD17ADB var_4           = dword ptr -4
.text:7CD17ADB arg_4           = dword ptr  8
.text:7CD17ADB arg_8           = dword ptr  0Ch
.text:7CD17ADB arg_C           = dword ptr  10h
.text:7CD17ADB arg_10          = byte ptr  14h
.text:7CD17ADB
.text:7CD17ADB ; FUNCTION CHUNK AT .text:7CD25BB4 SIZE 000000E3 BYTES
.text:7CD17ADB
.text:7CD17ADB                 mov     eax, eax
.text:7CD17ADD                 push    ebp
.text:7CD17ADE                 mov     ebp, esp
.text:7CD17AE0                 sub     esp, 414h
.text:7CD17AE6                 mov     eax, [ebp+arg_4]
.text:7CD17AE9                 push    ebx
.text:7CD17AEA                 push    esi
.text:7CD17AEB                 push    edi
.text:7CD17AEC                 xor     edi, edi
.text:7CD17AEE                 mov     esi, 208h
.text:7CD17AF3                 cmp     eax, edi
.text:7CD17AF5                 mov     [ebp+var_4], edi
.text:7CD17AF8                 jnz     loc_7CD25BB4
.text:7CD17AFE
.text:7CD17AFE loc_7CD17AFE:                           ; CODE XREF: NetpIsRemote+E0DF↓j
.text:7CD17AFE                 cmp     [ebp+arg_C], edi
.text:7CD17B01                 jz      short loc_7CD17B0D
.text:7CD17B03                 test    [ebp+arg_10], 1
.text:7CD17B07                 jnz     loc_7CD25C02
.text:7CD17B0D
.text:7CD17B0D loc_7CD17B0D:                           ; CODE XREF: NetpIsRemote+26↑j
.text:7CD17B0D                 mov     eax, [ebp+arg_8]
.text:7CD17B10                 mov     [eax], edi
.text:7CD17B12
.text:7CD17B12 loc_7CD17B12:                           ; CODE XREF: NetpIsRemote+E1B7↓j
.text:7CD17B12                 xor     eax, eax
.text:7CD17B14
.text:7CD17B14 loc_7CD17B14:                           ; CODE XREF: NetpIsRemote+E119↓j
.text:7CD17B14                                         ; NetpIsRemote+E13F↓j
.text:7CD17B14                 pop     edi
.text:7CD17B15                 pop     esi
.text:7CD17B16                 pop     ebx
.text:7CD17B17                 leave
.text:7CD17B18                 retn    10h
```

рисунок 9 обольстительно короткая NetpIsRemote()

беглый поиск обнаруживает пару подозрительных функций `wcscpy()` и `wcscat()`, однако, `wcscpy()` отпадает сразу, поскольку копирует строку фиксированной длинны, жестко прошитую внутри NetApi32.DLL, а `wcscat()` восходит к функции `I_NetNameCanonicalize()`, перспективы переполнения которой на данном этапе исследований весьма туманны и неясны (см. листинг. 1).

```
7CD25C24        lea     eax,[ebp+var_20C]
7CD25C2A        push    esi
7CD25C2B        push    eax
7CD25C2C        mov     eax,[ebp+arg_4]
7CD25C2F        push    dword ptr [eax+4]
7CD25C32        push    edi
7CD25C33        call    I_NetNameCanonicalize
7CD25C38        cmp     [ebp+var_4],edi
7CD25C3B        jz      loc_7CD25C48
7CD25C3D        mov     eax,[ebp+arg_C]
7CD25C40        lea     ebx,[ebp+var_20C]
…
7CD25C6E        push    asc_7CD17CF4
7CD25C73        push    [ebp+arg_C]
7CD25C76        call    ds:__imp_wcscpy
7CD25C7C        pop     ecx
7CD25C7D        pop     ecx
7CD25C7E        push    ebx
7CD25C7F        push    [ebp+arg_C]
7CD25C82        call    ds:__imp_wcscat
```

Листинг 1 дизассемблерный фрагмент NetpIsRemote() с потенциально опасными функциями wcscpy() и wcscat()

но ведь в нашем распоряжении есть готовая заплатка! давайте, чтобы не блуждать впотьмах, просто сравним дизассемблерные листинги функции `NetpIsRemote()` до и после обновления — все изменения станут сразу очевидны!
скачиваем заплатку с официального сервера Microsoft (для моей Windows 2000 SP4 это download.microsoft.com/download/9f06d3f3-87d0-445d-8a41-d2ffef9a40ba/windows2000-kb921883-x86-rus.exe), представляющую собой обыкновенный самораспаковывающийся cab-архив. в прошлом ревю мы показывали как извлечь его содержимое с помощью hiew'a, однако, есть и более короткий путь — достаточно "скормить" исполняемый файл rar'у и все упакованные файлы предстанут перед нашими глазами!

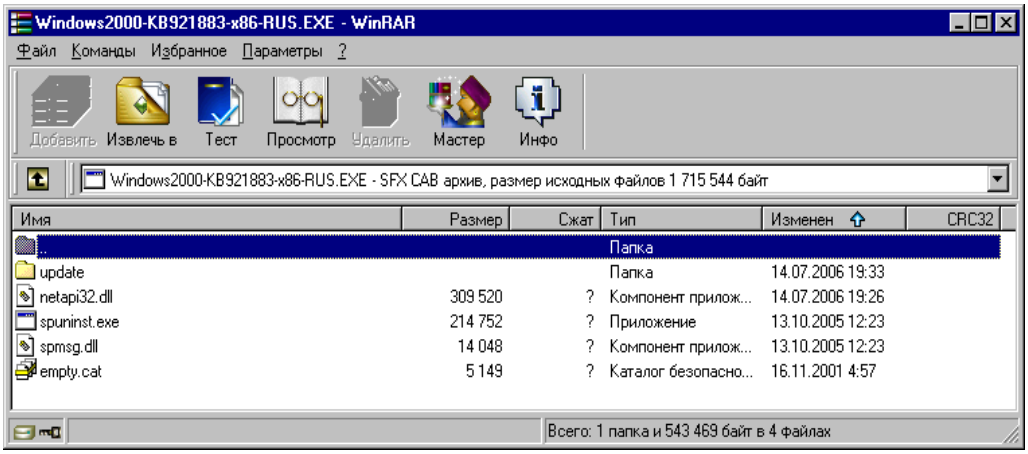

Рисунок 10 распаковка пакета обновления с помощью rar'a

извлекаем из архива NetApi32.DLL, переименовываем ее, например, в NetApi32-new.DLL и загружаем в дизассемблер. какие различия между старой и новой версией `NetpIsRemote()` мы увидим? в первую очередь это гнусная выходка компилятора, инвертировавшего условные переходы, в результате чего ветви программы поменялись местами:

```
JNZ branch_A    →      JZ branch_B
breanch_B                     breanch_A
```

Листинг 2 инверсия ветвей в старой и новой версиях NetApi32.DLL

непонятно? что ж, покажем это на конкретном примере:

```
7CD17AF3        cmp  eax, edi          7CD230FF        cmp  eax, edi
7CD17AF5        mov  [ebp+var_4], edi  7CD23101        mov  [ebp+var_4],edi
7CD17AF8        jnz  loc_7CD25BB4      7CD23104        jz   loc_7CD23150
7CD17AFE loc_7CD17AFE:                 7CD23106        mov  cx, [eax]
7CD17AFE        cmp  [ebp+arg_C],edi   7CD23109        cmp  cx, di
7CD17B01        jz   loc_7CD17B0D      7CD2310C        jz   loc_7CD23150
7CD17B03        test [ebp+arg_10], 1   …
7CD17B07        jnz  loc_7CD25C02      7CD23150 loc_7CD23150:
…                                      7CD23150        cmp  [ebp+arg_8], edi
7CD25BB4 loc_7CD25BB4:                 7CD23153        jz   loc_7CD23212
7CD25BB4        mov  cx, [eax]         7CD23159        test [ebp+arg_10], 1
7CD25BB7        cmp  cx, di            7CD2315D        jz   loc_7CD23212
7CD25BBA        jz   loc_7CD17AFE      7CD23163        mov  ebx, [ebp+arg_8]
```

Листинг 3 конкретный пример инверсии ветвей, слева показана старя версия NetApi32.DLL, слева — новая. красным цветом выделена ветвь A, синим — B

приведенный фрагмент функции `NetpIsRemote()` идентичен в обоих версиях, но порядок машинных команд поменялся местами, ослепляя большинство анализаторов типа fc.exe и wnindiff, которыми мы пользовались ранее. как же быть? а вот как! берем блок A из строй версии NetApi32.DLL и ищем похожий на него блок в NetApi32-new.DLL, затем добиваемся, чтобы первые строки блоках в обоих окнах дизассемблера совпадали (что легко осуществляется мышью или курсорными клавишами). теперь начинаем быстро переключаться с одного окна дизассемблера на другое по <ALT-TAB>, при этом, несовпадающие строки начинают интенсивно мерцать, выдавая

различия с головой (аналогичным образом астрономы ищут и новые звезды — т.е. звезды меняющий свой блеск — попеременно проецируя на экран несколько слайдов снятых в разное время они смотрят — не начнет ли какая точка ритмично мигать. способ древний как мамонт, но очень надежный).
действуя таким образом, мы довольно быстро найдем "линую" функцию **wcslen()**, которой не было ранее и которая контролирует длину строки перед копированием:

```
7CD25C2C        mov  eax, [ebp+arg_4]
7CD25C2F        push dword ptr [eax+4]
7CD25C32        push edi
7CD25C33        call I_NetNameCanonicalize
7CD25C38        cmp  [ebp+var_4],edi
7CD25C3B        jz   loc_7CD25C48
7CD25C3D        mov  eax, [ebp+arg_C]
7CD25C40        lea  ebx, [ebp+var_20C]
7CD25C46        jmp  loc_7CD25C60
7CD25C60 loc_7CD25C60:
7CD25C60        mov  ecx, [ebp+arg_8]
7CD25C63        neg  eax
7CD25C65        sbb  eax, eax
7CD25C67        cmp  [ebp+arg_C],edi
7CD25C6A        mov  [ecx],eax
7CD25C6C        jz   loc_7CD25C8A
7CD25C6E        push  asc_7CD17CF4
7CD25C73        push  [ebp+arg_C]
7CD25C76        call   ds:__imp_wcscpy
7CD25C7C        pop    ecx
7CD25C7D        pop    ecx
7CD25C7E        push   ebx
7CD25C7F        push   [ebp+arg_C]
7CD25C82        call   ds:__imp_wcscat
```

```
7CD2318D mov  eax,[ebp+arg_0]
7CD23190 push dword ptr [eax+4]
7CD23193 push edi
7CD23194 call I_NetNameCanonicaliz
7CD23199 cmp  [ebp+var_4],edi
7CD2319C jz   loc_7CD231A9
7CD2319E mov  eax, [ebp+arg_8]
7CD231A1 lea  ebx, [ebp+var_20C]
7CD231A7 jmp  loc_7CD231C1
7CD231C1 loc_7CD231C1:
7CD231C1 mov  ecx, [ebp+arg_4]
7CD231C4 neg  eax
7CD231C6 sbb  eax, eax
7CD231C8 cmp  [ebp+arg_8],edi
7CD231CB mov  [ecx],eax
7CD231CD jz   loc_7CD23208
7CD231CF push ebx
7CD231D0 call ds:__imp_wcslen
7CD231D6 add  eax, 3
7CD231D9 pop  ecx
7CD231DA cmp  [ebp+arg_C], eax
7CD231DD jnb  short loc_7CD231EC
…
7CD231EC loc_7CD231EC:
7CD231EC push asc_7CD11744
7CD231F1 push [ebp+arg_8]
7CD231F4 call ds:__imp_wcscpy
7CD231FA pop  ecx
7CD231FB pop  ecx
7CD231FC push ebx
7CD231FD push [ebp+arg_8]
7CD23200 call ds:__imp_wcscat
```

Листинг 4 старая (слева) и новая (справа) версии NetpIsRemote()

ага! вот оно! между функциями `I_NetNameCanonicalize()` и `wcscpy()/wcscat()` в обновленной версии NetApi32.DLL внедрен вызов **wcslen()**, проверяющий размер строки, возращенный **I_NetNameCanonicalize()** перед его копированием в… блок памяти, переданнй по указателю через аргумент функции (arg_C в старой и arg_8 в новой версии). сама же `I_NetNameCanonicalize()` так же принимает на "грудь" указатель, извлекаемый структуры, переданной `NetpIsRemote()` в качестве аргумента.
все это создает крайне благоприятные условия для реализации атаки и засылки shell-кода. достаточно "всего лишь" передать слишком длинную строку функци `I_NetNameCanonicalize()` вместе с указателем на крошечный буфер, неспособный вместить "канонизированное" имя. локальный exploit пишется без проблем — ведь функция `NetpIsRemote()` экспортируется и все, что нам нужно — это восстановить его прототип.
однако, локальный exploit – не слишком-то полезная штука, гораздо больший интерес вызывают удаленные exploit'ы. можем ли мы использовать эту уязвимость для атаки, и если да, то как?
переходим в начало функции `NetpIsRemote()` и, нажав <ALT-V> [view], <O> [open sub-view], <O> [cross references], смотрим по перекрестным ссылкам — какие функции ее вызывают. большинство функций семейства `I_*()` доступны через SMB Remote APIs, т. е. их можно вызывать удаленно по RPC-протоколу, инкапсулированному в SMB. подобнее об этом можно прочитать на форуме immunitysec'a в сообщении известного хакера H D Moore'a: lists.immunitysec.com/pipermail/dailydave/2006-August/003400.html, а в MSDN можно найти готовый пример реализации Bloodhound'a Parser DLL for SMB Remote APIs, расположенный в файле REMAPI.C и основанный на

функции `CreateProtocol()`, описанной в windowssdk.msdn.microsoft.com/en-us/library/ms707941.aspx.

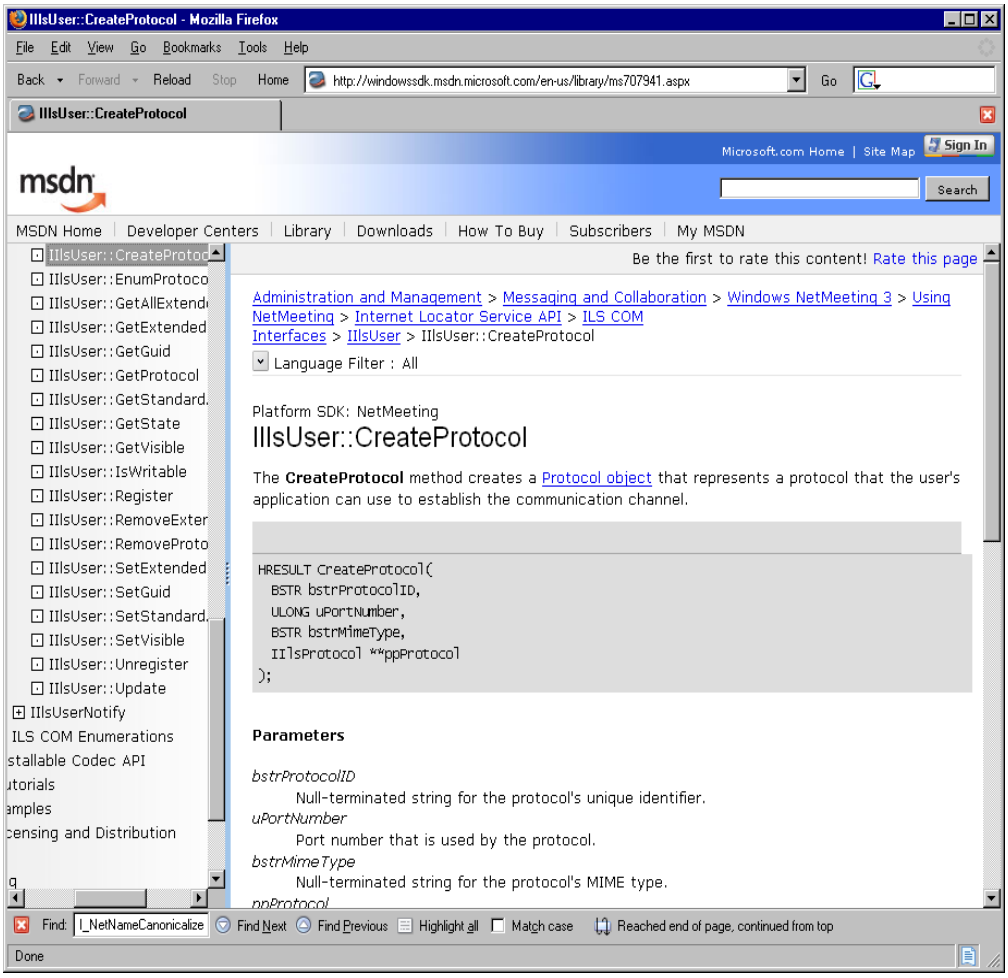

рисунок 11 CreateProtocol() на MSDN

основная ценность REMAPI.C заключается в том, что имена `I_*()` функций (совпадающих с именами команд SMB протокола) в нем перечислены "прямым текстом". фактически это единственный источник информации, который у нас только есть.

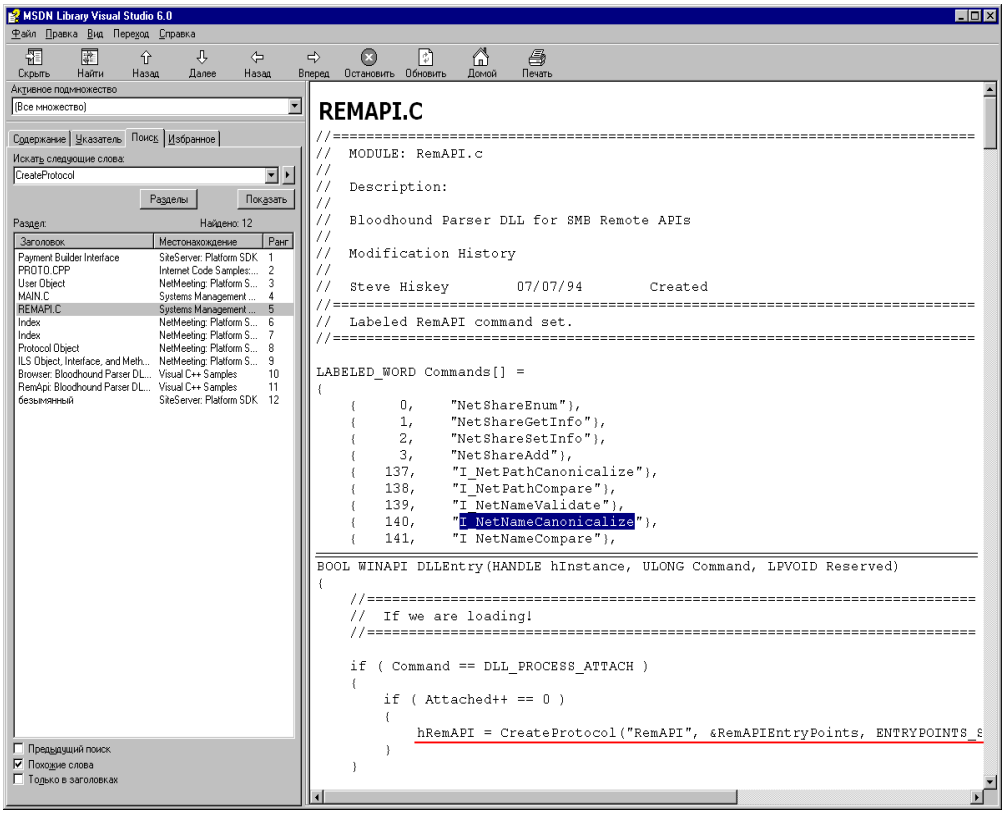

рисунок 12 REMAPIC.C – основной источник информации по SMB командам

просматривая список перекрестных ссылок, необходимо отобрать функции, присутствующие в файле REMAPI.C: `I_NetPathType()`, `I_NetNameValidate()`, `I_NetNameCanonicalize()`, `I_NetNameCompare()` и `I_NetPathCompare()`.

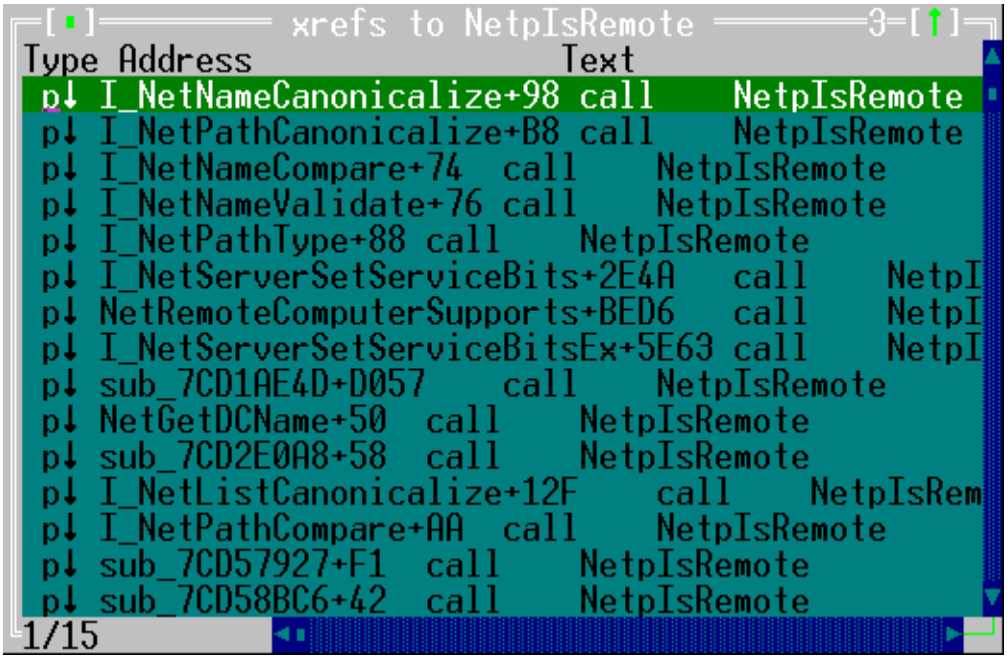

рисунок 13 добываем список перекрестных ссылок на NetpIsRemote()

теперь необходимо проанализировать дизассемблерный листинг каждой из них и найти хотя бы одну функцию, передающую `NetpIsRemote()` свой собственный аргумент вместе с указателем на локальный буфер фиксированного размера. как ни смешно, но этой функцией оказывается… `I_NetNameCanonicalize()`. нет, это не ошибка!

`I_NetNameCanonicalize()` вызывает `NetpIsRemote()`, а `NetpIsRemote()` вызывает `I_NetNameCanonicalize()`, вот такая, значит ### блядь ### рекурсия у нас получается. но довольно эмоций! ниже приведен дизассемблерный фрагмент `I_NetNameCanonicalize()` в котором и происходит переполнение.

```
7CD1F980 loc_7CD1F980:        ; CODE XREF: I_NetNameCanonicalize+A4↑j
7CD1F982       push    104h                  ; размер буфера(?)
7CD1F987       lea     eax,[ebp+var_230]     ; указатель на…
7CD1F98D       push    eax                   ; …локальный буфер.
7CD1F98E       lea     eax,[ebp+var_20]      ; указатель на другой…
7CD1F991       push    eax                   ; …локальный буфер
7CD1F992       push    [ebp+arg_0]           ; аргумент функции
7CD1F995       call    NetpIsRemote          ; вызов уязвимой функции
```

Листинг 5 вот где происходит переполнение

таким образом, все, что нам нужно — это вызвать функцию `I_NetNameCanonicalize()`, так же являющуюся командой протокола SMB и передать ей слишком длинную строку, вызывающую срыв стека, что легко осуществляется из любой точки Сети, если, конечно, внешний SMB-трафик не отсекается брандмауэром.

проблема в том, что `I_NetNameCanonicalize()` ни хрена не документирована и ее прототип неизвестен. кое-какую информацию удается нарыть в файле netapi32.inc, входящем в состав ассемблера MASM, но она слишком малоинформативна.

```
I_NetNameCanonicalize PROTO :DWORD,:DWORD,:DWORD,:DWORD,:DWORD,:DWORD
```

Листинг 6 прототип I_NetNameCanonicalize()

к счастью, у нас есть Bloodhound Parser, заботливо предоставленный Microsoft, позволяющий сниффить сеть и декодировать пролетающий SMB-команды. аргументы, правда, не декодируются, но об их назначении нетрудно догадаться и самостоятельно. еще можно дизассемблировать `I_NetNameCanonicalize()`, разобравшись какие аргументы она ожидает. это рутинная и неинтересная работа, к тому же уже проделанная авторами metasploit exploit'a, так что не будем повторяться, а лучше позаимствуем готовый "движок" и адоптируем его для собственных нужд.