exploits review
15h выпуск

крис касперски ака мыщъх, a.k.a. nezumi, a.k.a elraton, a.k.a. souriz, no-email

парад дыр в NT продолжается! как и в прошлом выпуске, мыщъх делится своими находками, впрочем, не претендуя, что это именно _его_ находки, но, во всяком случае, их явного описания в сети не нашлось, да и так ли важно, кто первый сказал "гав"?! главное, что это реальные дыры, работающие на всех современных операционных системах от Microsoft от W2K по Вислу включительно. да будет свет! тушите свечи! ведь настоящие хакеры дебажат в полной темноте и точат ништяки на глубине норы!

Microsoft Windows – PE loader BSOD

brief: еще весной 2004 года мыщъх обнаружил гремучую уязвимость в загрузчике PE-файлов, роняющую W2K SP3 в BSOD с прикладного режима даже без прав администратора, и хотя дыра была описана в "системном администраторе", русском издании "компьютерных вирусов снаружи и изнутри", а так же "Shellcoder's programming uncovered", выпущенной на английском языке – воз и ныне там. разработчики XP SP2 исправили большое количество дыр в PE-загрузчике, но эту пропустили. не заметили ее и разработчики Вислы, а мыщъх к настоящему времени уже вплотную подошел к созданию боевого exploit'a, позволяющего загружать shell-код в ядерное пространство, не требуя от него ни цифровой подписи ни даже прав администратора. Позволю себе привести цитату из своей собственной статьи четырехлетней давности: "поля File Alignment и Section Alignment задают кратность выравнивания секций на диске и в памяти, соответственно. официально о кратности выравнивая известно лишь то, что она представляет собой степень двойки, причем: а) Section Alignment должно быть больше или равно 1000h байт; б) File Alignment должно быть больше или равно 200h байт; в) Section Alignment должно быть больше или равно File Alignment. Если хотя бы одно из этих условий не соблюдается, файл не будет загружен. В Windows NT существует недокументированная возможность отключения выравнивания, основанная на том, что загрузку прикладных исполняемых файлов/динамических библиотек и системных драйверов обрабатывает один и тот же загрузчик. Если Section Alignment == File Alignment, то последнее поле может принимать любой значение, представляющее собой степень двойки и превышающее 10h. Условимся называть такие файлы "невыровненными". Хотя этот термин не вполне корректен, лучшего пока не придумали. К не выровненным файлам предъявляется следующее, достаточно жесткое требование – виртуальные и физические адреса всех секций обязаны совпадать, т. е. страничный имидж должен полностью соответствовать своему дисковому образу. Впрочем, никакое правило не обходится без исключений и виртуальный размер секций может быть меньше их физического размера, но не более чем Section Alignment - 1 байт (т. е. секция все равно будет выровнена в памяти). Самое интересное, что это физический размер последней секции "вылетает" за пределы загружаемого файла, операционная система выбрасывает голубой экран смерти";

targets: NT, W2K, XP, Server 2003, Server 2008, Висла;

exploit: демонстрационный exploit выложен на мыщъх'ином сервере по адресу http://nezumi.org.ru/souriz/hack/PE_BSOD (это все тот же древний exploit, датируемый весной 2004 года, и не учитывающий некоторые особенности более поздних осей, полноценный exploit, загружающий неподписанный код в ядро 64-битных систем планируется приложить на диск нового издания книги "компьютерные вирусы снаружи и изнутри");

solution: отсутствует;

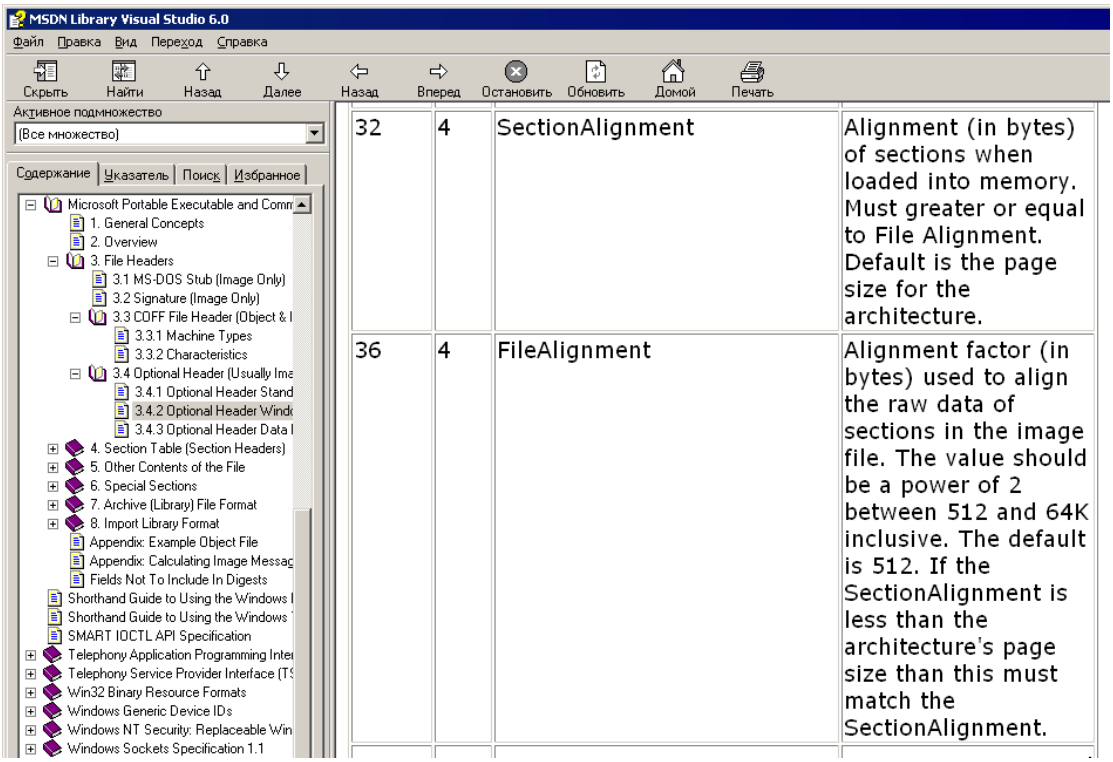

Рисунок 1 спецификация PE-файла от Microsoft

Microsoft Vista – отключение DEP, ASLR, SafeSEH, etc

brief: начиная с XP парни из Microsoft всерьез озаботились безопасностью и уже в XP SP2 появился DEP (поддержка NX/XD битов страниц, препятствующих выполнению кода на куче и в стеке), механизм SafeSEH, реально разработавший только в Висле, ну а сама Висла явила нам рандомизацию адресного пространства, она же ASLR. хакерам сразу стало интересно как отключить все эти хорошие вещи, затрудняющие атаки и отравляющие жизнь. вместе с этим они задумались: как же в Висле ухитряются работать старые программы, использующие хитрые антиотладочные трюки, противоречащие политике защитных механизмов. ну, плохая совместимость Вислы с программным обеспечением, написанным до нее — всем хорошо известна, однако, для популярных защитных пакетов (ASPack, Start Force) в NTDLL.DLL была оставлена специальная "нычка", распознающая запротекченные файлы и молчаливо отключающая защитные механизмы Вислы, препятствующие их функционированию. опознание происходит по… именем секций PE-файла и если это '.sforce', '.pcle' или '.aspack', файл автоматически получает "иммунитет". сброс поля COFF-заголовка "characteristics" в 010Eh (210E для динамических библиотек) так же отключает множество защит (даже тех, что еще не появились на свет), причем, SFC (система автоматической проверки целостности системных файлов) _не_ контролирует поле "characteristics" и потому отключение защиты возможно осуществлять даже для системных файлов: exe, dll, ocx. естественно, удаленно этого не сделать и для реализации атаки необходимо найти дыру в каком-нибудь приложении, однако, это ничуть не снижает актуальности угрозы;

targets: XP, Server 2003, Server 2008, Висла;

exploit: не требуется;

solution: отсутствует;

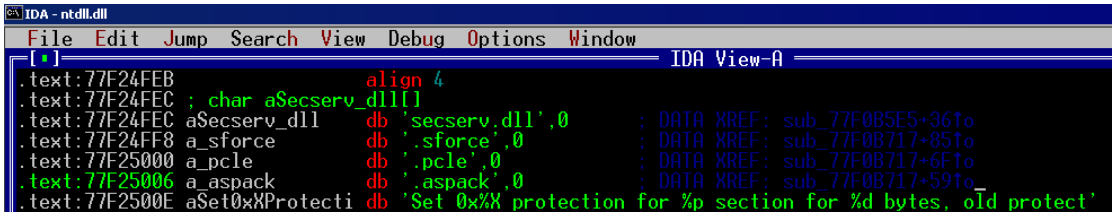

Рисунок 2 лазейка для некоторых популярных защит, оставленная в файле NTDLL.DLL от Вислы

Microsoft Windows – EFS на страже малвари

brief: начиная с W2K в Windows появилась поддержка шифрованной файловой системы (Encrypting File System или, сокращенно, EFS), реализованной в виде расширения к NTFS и архитектурно находящейся на один уровень ниже ее, в результате чего все операции с зашифрованными файлами протекают абсолютно прозрачно как для пользователя, так и для прикладных приложений. удобно, однако! но что же скрывается за этим уродством, тьфу, простите, оговорился, удобством? для каждого пользователя произвольным образом генерируется пара асимметричных ключей: публичный ключ (доступный всем) и приватный ключ, который по теории не должен быть доступен никому, кроме его самого. публичный ключ копируется в каталог \documents-n-settings\user-name\application data\microsoft\systemcertificates\user-name\certificates\, а приватный ключ помещается в \documents-n-settings\user-name\application data\microsoft\crypto\rsa\. однако, поскольку асимметричная криптография — тормозная штука, Microsoft придумала шифровать файлы симметричным ключом пользователя (File Encryption Key или, сокращенно, FEK), так же генерируемом произвольным образом. содержимое файла шифруется FEK-ключом по алгоритму DESX (W2K) или AES (XP и выше), а сам FEK ключ шифруется публичным ключом и в зашифрованном виде сохраняется в файле в отдельном именованном NTFS-потоке $EFS. специальный компонент системы, называемый агентом восстановления, считывает приватный ключ пользователя, извлекает из $EFS потока зашифрованный FEK-ключ, расшифровывает его, после чего расшифровывает FEK-ключом содержимое самого зашифрованного файла. достоинство W2K в том, что в ней имеется агент восстановления по умолчанию, представленный в лице администратора системы, в хранилище сертификатов которого автоматически копируются все приватные пользовательские ключи, а потому администратор может расшифровать файл, зашифрованный любым пользователем, даже если толь угробит свою учетную запись. начиная с XP, агент восстановления по умолчанию ушел в отставку и хотя администратор по-прежнему может расшифровать файлы, зашифрованные пользователями, ему необходимо _вручную_ скопировать их приватные ключи в свое хранилище сертификатов. это была предыстория. а теперь — уязвимость! поскольку, антивирусное программное обеспечение как правило работает с правами администратора (или из-под учетной записи спец пользователя), чтобы видеть все файлы, то зловредному программному обеспечению ничего не стоит скрыться от его глаз — достаточно просто присвоить своим файлам атрибут "encrypted" и все! а сертификат, кстати говоря, можно динамически удалять/добавлять из хранилища, используя его только на время активной фазы работы — это предотвратит его ручное копирование администратором. правда, если жертва сидит под "админом", то… хм, ну и это не преграда, ведь малварь может создать для себя отдельного пользователя, запуская файлы через runas и антивирус их никак не сможет захавать. конечно, существование самих файлов не скрыть и при попытке доступа к ним, антивирус получит ошибку отказа в доступе, но… это все-таки не тоже самое, что антивирусная тревога!

targets: XP, Server 2003, Server 2008, Висла;

exploit: не требуется;

solution: отсутствует;

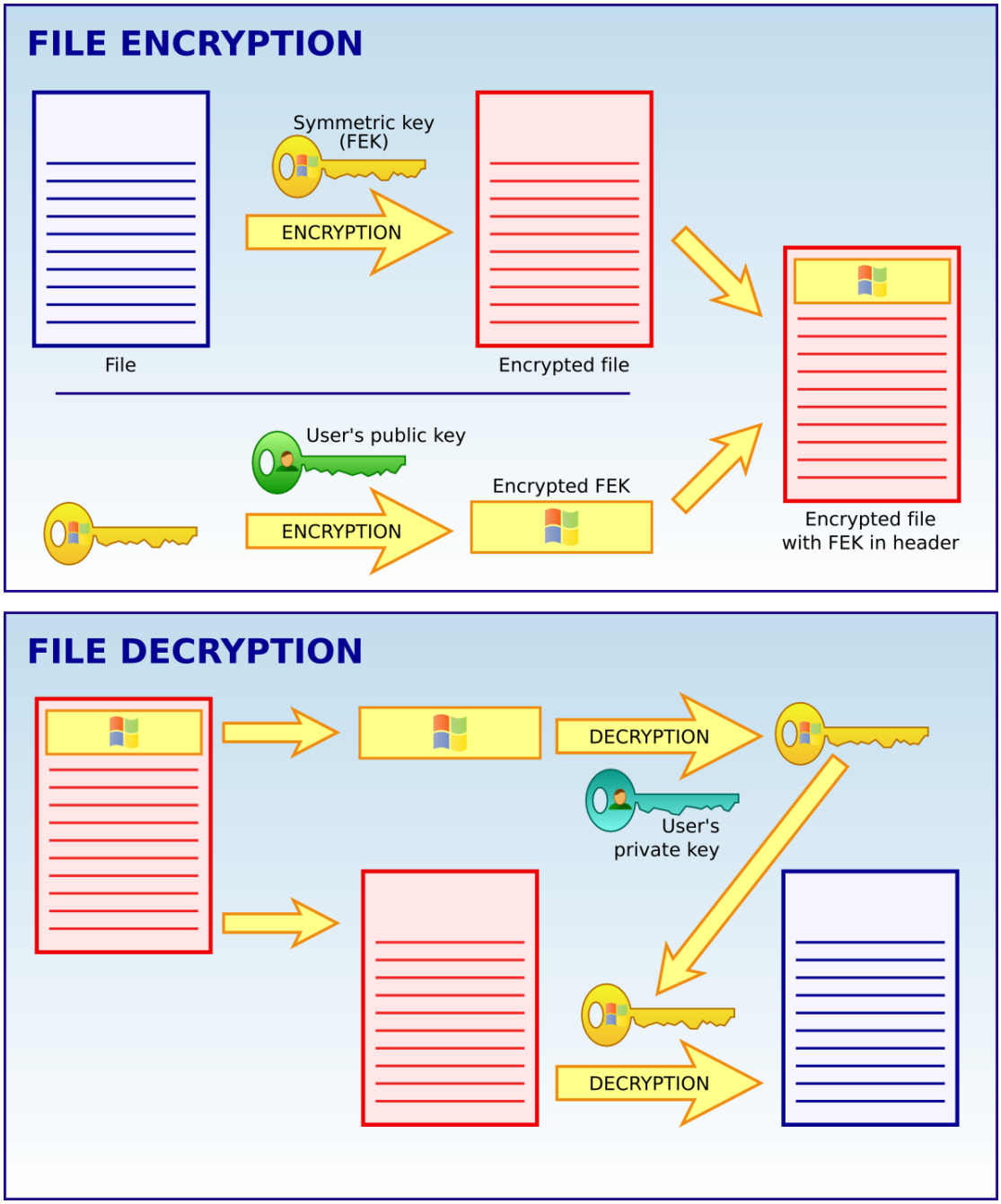

Рисунок 3 механизм шифрования файлов в W2K/XP/Висла

full disclose:
честная кража процессорного времени

В многозадачных (и особенно многопользовательских!) системах очень важно знать сколько процессорного времени "скушала" та или иная задача, чтобы планировать загрузку ЦП, не допуская нежелательных тормозов, мешающих всем пользователям. Большинство коммерческих провайдеров накладывают жесткие ограничения на предельное время исполнения скрипта, зачастую приостанавливая хостинг при его превышении, а некоторые вычислительные центры напрямую торгуют машинным временем (они бы еще и воздухом торговали, блин!). Спрос рождает предложение и возникает естественная потребность — заплатить поменьше, а получить побольше, то есть, другими словами, украсть машинное время, что актуально не только для суперкомпьютеров, но и обычных серверов, поскольку, большинство малвари палится как раз потому, что не умеет правильно воровать.

Все операционные системы линейки NT (как сервера так и рабочие станции) поддерживают развитую систему мониторинга счетчиков производительности, отображая в реальном времени загрузку ЦП и выводя полное время, проведенное каждым процессом в пользовательском режиме и в режиме ядра. Увидеть его можно как с помощью "Системного монитора" (см. рис. 4), так и "Диспетчера Задач". Разницы между ними никакой нет, т. к. они считают показания одних и тех же счетчиков. Утилиты сторонних разработчиков (например, "Process Explorer") Марка Руссиновчика так же выводят данную информацию, причем в более наглядной форме. И ведь многие ей верят…

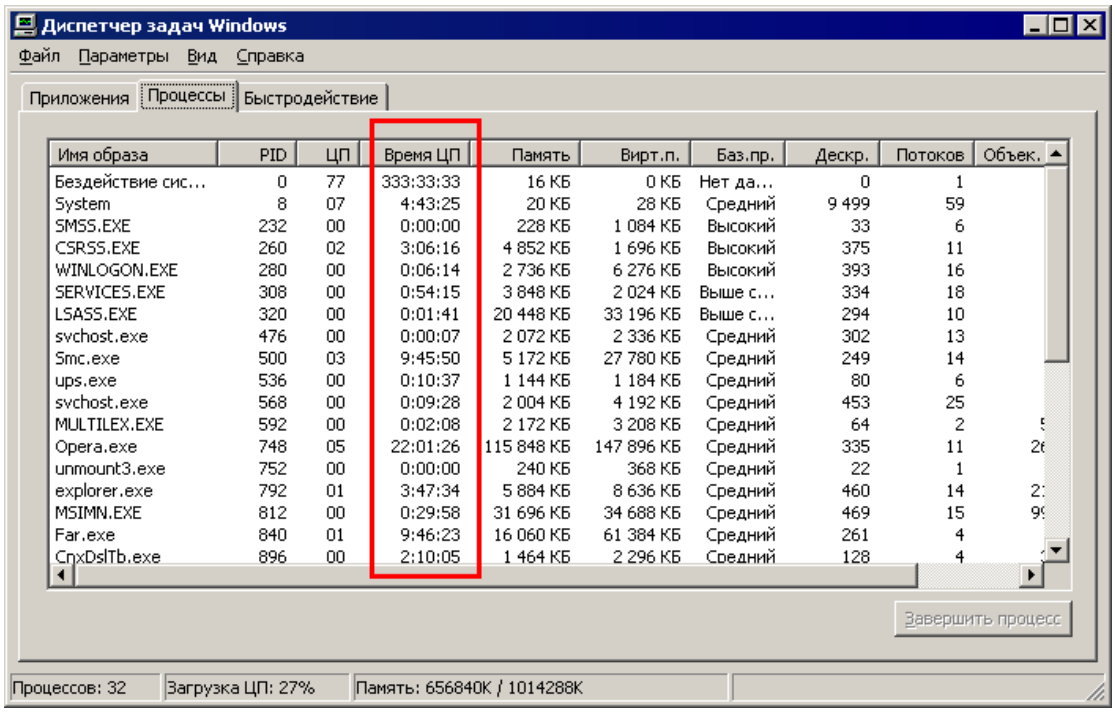

Рисунок 4 колонка "Время ЦП" вместе с колонкой "загрузка ЦП" Диспетчера Задач отображают неверные данные, которым нельзя доверять

Лишь немногие задумываются насколько точны эти показания, и можно ли их подделать, а если можно, то как?! Достопочтенный "Червь Морриса" периодически расщеплял себя на два, уничтожая материнский процесс. Счетчик потребления процессорного времени дочернего процесса при этом, естественно, устанавливался в ноль, что позволило Моррису избежать "накопления" времени ЦП, однако, подобная активность слишком заметна и привлекает к себе внимание, что не есть гуд.

Разумеется, это слишком грубая схема и существует намного более изощренные способы хищения процессорного времени, впервые продемонстрированные и теоретически обоснованные известным экспертом по безопасности Tsutomu Shimomura (тем самым, который ловил Митника, сцука) в далеком 1980 году, но никакого внимание на него так и не обратили. Лишь в 2007 году (т. е. двадцать семь лет спустя – для компьютерной индустрии огромный срок!) два студента Израильского университета "School of Computer Science and Engineering The Hebrew University" Yoav Etsion и Dror G. Feitelson в соавторстве с Dan'ом Tsafrir — сотрудником корпорации IBM, опубликовали статью "Secretly Monopolizing the CPU Without Superuser Privileges", показывающую, что за минувшие годы ничего не изменилось и кража процессорного времени по-прежнему остается возможной во всех операционных системах: Linux, BSD, NT, etc.

К тем же самым выводам и (приблизительно) в тоже самое время пришел и мыщъх, исследующий устройство счетчиков производительности и алгоритм определения загрузки ЦП/процессорного времени в W2K, Server 2003 и Висле. Выяснилось, что система измеряет не загрузку ЦП как таковую, а готовность системного планировщика предоставить процессорное время потоку по первому требованию. Это очень грубый показатель, а методы его измерения вообще таковы, что вызывают шевеление волос в разных местах. Упрощенно все происходит приблизительно так. Имеется рабочий, который работает. Или не работает. И специальный "инспектор" через регулярные промежутки времени (например, каждый час) приходит и

поверяет чем тот в данный момент занимается. Если рабочий вкалывает как пара Карло, то ему ставится "зачот" за весь отчетный период и, соответственно, наоборот.

Вообразим вполне вероятную ситуацию, при которой рабочий устраивает себе пятиминутный перекур каждый час. Как нетрудно рассчитать, его "загрузка" составит ~90%, но!!! Если перекуры совпадут с приходом инспектора, мы получим… нулевую загрузку!!! Шутки в сторону, господа! При желании можно написать программу, потребляющую свыше 90% времени ЦП, но "Диспетчер Задач" (а вместе с ним и "Системный Монитор") будут осцилировать в переделах абсолютного нуля. Для этого достаточно отдавать остаток кванта времени за несколько миллисекунд до его истечения. Планировщик, обнаружив, что поток не исполняется, ошибочно пропишет ему ноль в графе "использование процессорного времени".

Длительность одного кванта в NT-подобных системах составляет порядка 100 мс, отдать остаток неиспользуемого кванта можно при помощи API-функции Sleep(0), которая соответствует функции nanosleep() в UNIX-системах. Самое сложное — это синхронизовать отдачу остатка кванта с приходом "инспектора", тогда, даже отдавая ничтожный остаток кванта, мы снизим потребление процессорного времени до абсолютного нуля (а, точнее, украдем все процессорное время).

Поскольку, алгоритмы планировки потоков меняются от одной версии системы к другой, написать переносимую программу довольно сложно, да и ненужно. Легче отдавать _существенный_ остаток кванта функцией Sleep(1), вгоняющий поток в сон на 1 мс (на самом деле, 10 мс, что связано с дискретностью системного таймера), гарантированно обеспечивая здоровый сон потока к моменту прихода инспектора, а, поскольку, при пробуждении, поток получает полный квант, открывающий новый "отчетный период", это естественным образом синхронизует его с приходом "инспектора". КПД программы, конечно, упадет (поскольку, часть времени она будет спать, вместо того, чтобы работать), но… куда нам спешить?

Напишем несложную тестовую утилиту (см. листинг 1) и проведем с ней несколько познавательных экспериментов:

```
// total - общее кол-во вычислений некоторого типа (неважно каких)
#define total  100000000

// steal - кол-во вычислений, выполняемых перед отдачей остатка кванта
#define steal  30000

main()
{
        // локальные переменные, которые мы будем использовать
        int a, b, c, sum = 1; DWORD A1;

        // засекаем время начала выполнения вычислительного цикла
        A1=GetTickCount();
        for (b = 0; b < total/steal; b++)     // мотаем главный цикл
        {
                // мотаем цикл, исполняющийся в пределах одного кванта
                for (a = 0;a < steal; a++) sum+=sum % 3 + 1;

                // …а остаток кванта мы спим как сурки
                //Sleep(1);
        }

        // выводим _реальное_ время, потраченное на вычисления
        printf("==%d sec\n", (GetTickCount() - A1)/1000);

        // возвращаем sum взад, чтобы оптимизирующий компилятор
        // не принял ее за неиспользуемую переменную и не выкинул
        // результаты вычислений, исказив результаты экспериментов
        return sum;
}
```

Листинг 1 cheater.c – программа, демонстрирующая честную кражу процессорного времени

Транслируем программу компилятором Microsoft Visual C++ со следующими ключами командной строки: "cl.exe /Ox cheater.c" (где /Ox – максимальная оптимизация) и запускам полученный файл cheater.exe на выполнение.

Поскольку функция Sleep(1) пока закомментирована, программа использует все доступное процессорное время и потому загрузка процессора (на однопроцессорной машине без поддержки Hyper-Threading или многоядерности) вплотную приближается к 100% (см. рис. 5), а

полное время выполнение (по показаниям самой программы) на P-III 733 MHz составляет 7 сек, что вполне соответствует показаниям счетчика производительности, оценившего потребление процессорного времени в 6 сек. (в данном случае, счетчику производительности можно верить, поскольку он вычисляет из общего времени выполнения то время, которое программа была вынуждена разделять с другими — в фоне играл winamp, пара файлов качалась из сети).

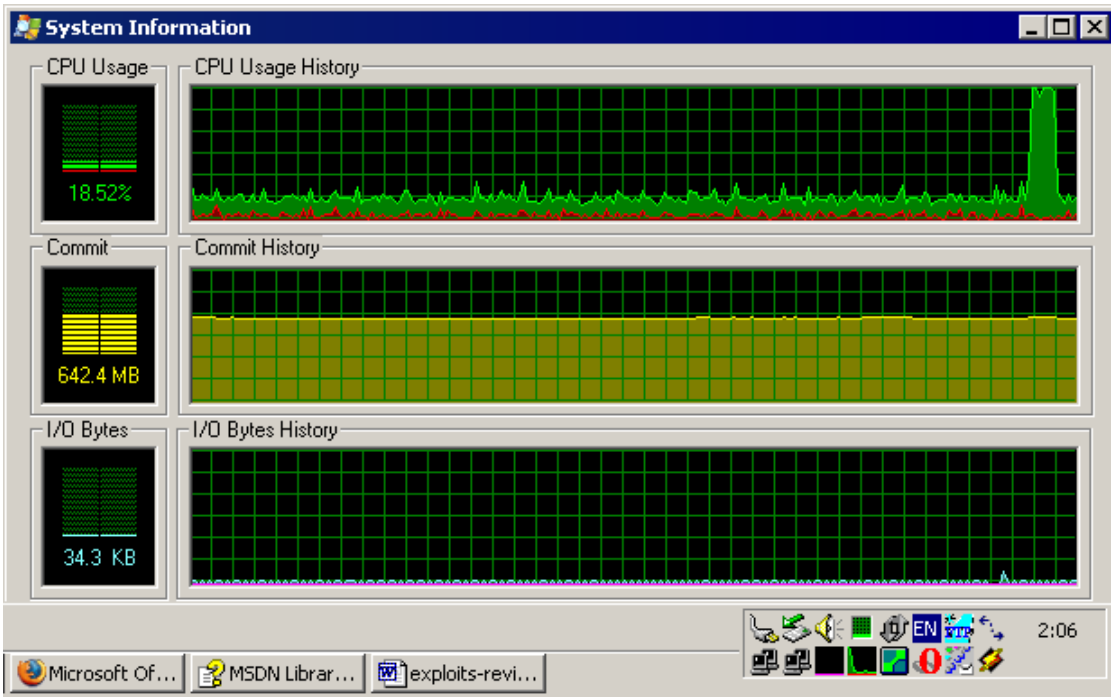

Рисунок 5 "честная" загрузка ЦП вычислительной задачей, выполняемой программой cheater.exe

А теперь раскомментируем Sleep(1), перекомпилируем программу и запустим ее по новой. Как нетрудно видеть (см. рис. 6), загрузка процессора существенно снизилась, достигнув в максимуме ~80%, а полное время выполнения программы увеличилось на одну секунду (8 сек.), однако, показания "Диспетчера Задач" изменились более, чем радикально и он показал всего 3 сек., что, очевидно, не соответствует действительности! Ведь объем вычислений не изменился!!! И потому _подлинное_ значение потребления процессорного времени никак не могло упасть!!! А ведь упало. Причем в два раза!

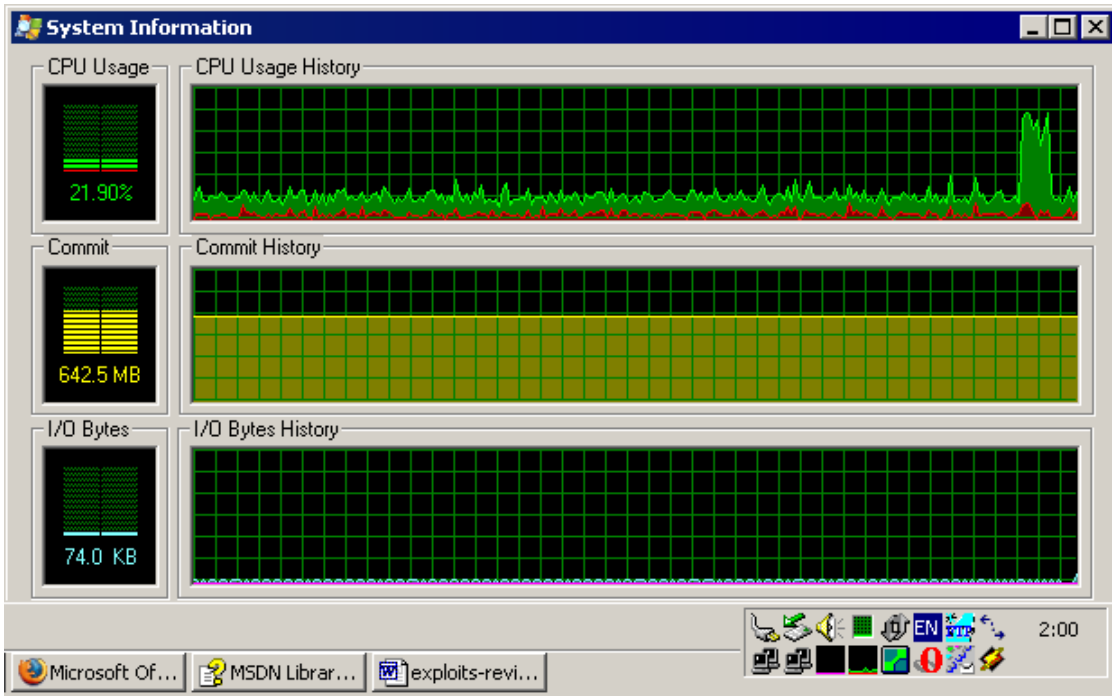

Рисунок 6 при отдаче остатков квантов загрузка ЦП заметно снижается (steal == 30000)

ОК, уменьшаем значение переменной steal с 30000 до 20000 и вновь перекомпилируем программу. На этот раз (см. рис. 7) загрузка процессора падает до ~50% (а ширина "холмика", соответственно увеличивается). Полное время выполнения (вследствие падения КПД) составляет уже 11 сек, но показания счетчика производительности продолжают уменьшаться и колеблются между 2 и 3 сек., что дает нам среднее значение в 2,5 сек.

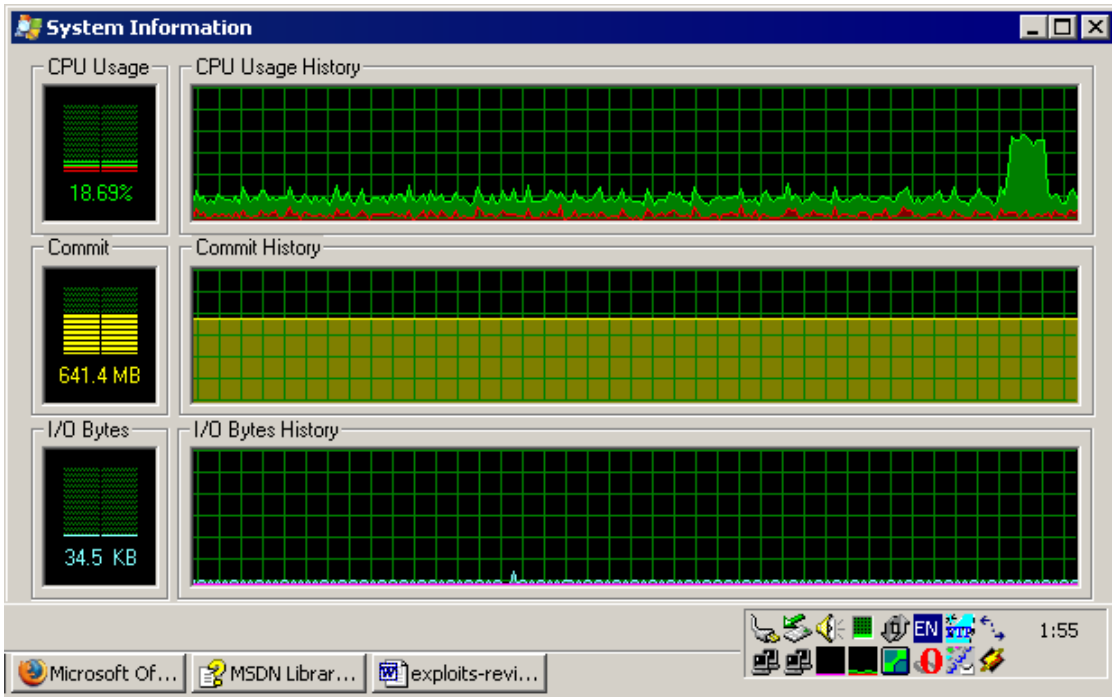

Рисунок 7 чем больший остаток кванта мы отдаем — тем ниже загрузка ЦП (steal == 20000)

А теперь — смертельный номер! Уменьшаем steal с 20000 до 10000, чтобы поток существенную часть своего времени проводил во сне. И что же?! Загрузка процессора (см. рис. 8) упала настолько, что его "холмик" практически сравнятся с "шумом" фоновых выплесков. Полное время выполнения программы увеличилось аж до 13 сек, (что вдвое больше

первоначального результата с закомментированной функцией Sleep), но зато счетчик производительности показывает 0 сек. потребления процессорного времени (или 1 сек — в худшем случае).

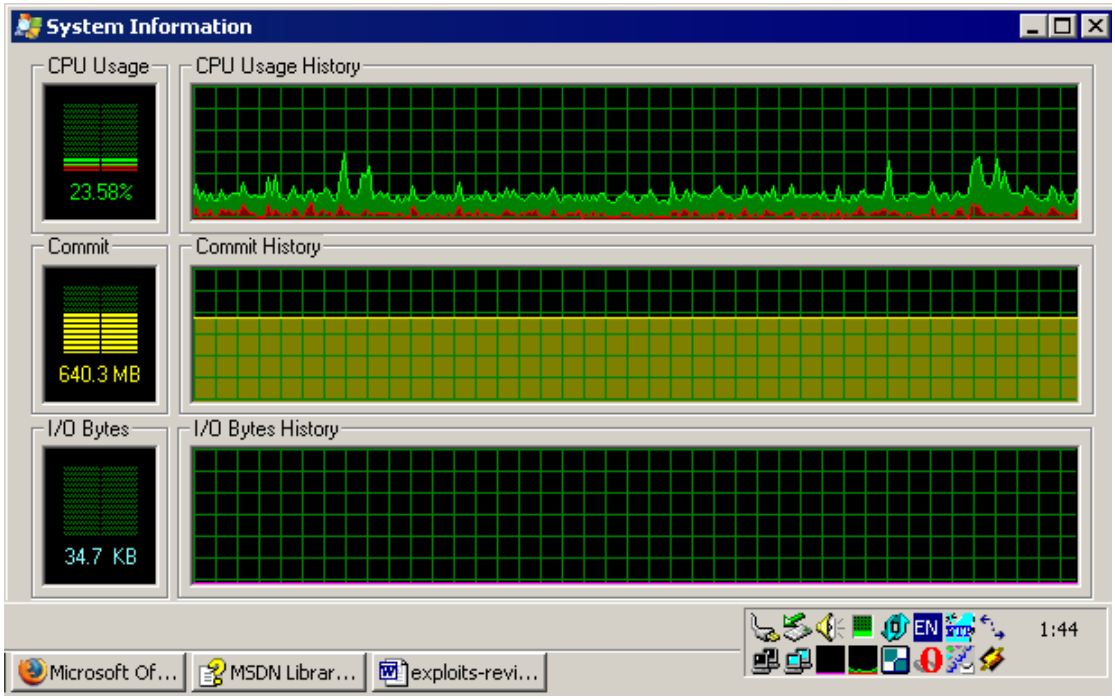

Рисунок 8 при отдаче половины кванта (steal == 10000) программа 50% времени проводит во сне, а 50% — интенсивно нагружает ЦП, но, по данным счетчиков производительности, загрузка ЦП там мала, что практически полностью тонет в фоновых "шумах"

На двухпроцессорных машинах украсть машинное время еще легче (тоже самое относится к Hyper-Threading и многоядерным ЦП), поскольку наш вычислительный поток в каждый момент выполняется на одном процессоре (хотя и может мигрировать с процессора на процессор в процессе своей жизнедеятельности - это зависит от того, какой процессор быстрее освободится), но подсчет машинного времени обычно усредняется для всех процессоров. Даже при выводе раздельных графиков загрузки (по одному на каждый ЦП) за счет того, что наш поток периодически перебрасывается с одного процессора на другой, значения счетчиков производительности усредняются, в результате чего происходит их "сглаживание".

Естественно, для эффективной кражи процессорного времени необходимо знать тактовую частоту целевого процессора (процессоров), подбирая оптимальное значение переменной steal, определяющий объем вычислений, которые программа проводит прежде чем отдать остаток кванта другому потоку. Чем выше тактовая частота — тем выше значение steal, и, соответственно, наоборот.

значение steal	время ЦП по счетчику производительности	полное время выполнения программы
– (кванты не отдаются)	06 сек.	07 сек.
30000	03 сек.	08 сек.
20000	2 - 3 сек.	11 сек.
10000	0 - 1 сек.	13 сек.

Таблица 1 итоги кражи процессорного времени