exploit review – июнь 2006

крис касперски ака мыщъх, no-email

удаленное переполнение буфера в zlib

6 июня Tavis Ormandy обнаружил уязвимость библиотеки **ZLIB** версии 1.2.2 и более ранних, приводящую к переполнению буфера с возможностью выполнения произвольного кода на пораженной машине. Ошибка контроля допущена в функции **`inflate_table()`**, расположенной в файле inftrees.c, ключевой фрагмент которой идет ниже:

```
/* check for an over-subscribed or incomplete set of lengths */
left = 1;
for (len = 1; len <= MAXBITS; len++)
{
       left <<= 1; left -= count[len];
       if (left < 0) return -1;             /* over-subscribed */
}
if (left > 0 && (type == CODES || (codes - count[0] != 1)))
return -1;                                  /* incomplete set */
```

Листинг 1 ошибка контроля в функции inflate_table() библиотеки zlib-1.2.2

Исправленный вариант выглядит так:

```
/* check for an over-subscribed or incomplete set of lengths */
left = 1;
for (len = 1; len <= MAXBITS; len++)
{
       left <<= 1; left -= count[len];
       if (left < 0) return -1;             /* over-subscribed */
}
if (left > 0 && (type == CODES || max != 1))
return -1;                                  /* incomplete set */
```

Листинг 2 та же самая функция в библиотеки zlib-1.2.3

И хотя рабочего exploit'a в сети обнаружить не удалось, приведенной выше информации вполне достаточно, чтобы его написал любой грамотный хакер.

Это настоящая катастрофа! Библиотека ZLIB используется огромным количеством программ как с динамической, так и со статической компоновкой. А это значит, что для устранения уязвимости обновить файл zlib1.dll/libz.so будет явно недостаточно и потребуется перекомпилировать _все_ программное обеспечение, слинкованное с ZLIB статическим способом. А если программа включает в себя фрагменты исходных текстов компрессора, "вживляя" их в свое тело, то… починить ее сможет только разработчик.

Полный список уязвимых систем можно найти на www.securityfocus.com/bid/14162/info, а заплатки к ним на http://www.securityfocus.com/bid/14162/solution. Как и следовало ожидать, под угрозой оказались практически все платформы: Apple Mac OS X, Conectiva Linux, Debian Linux, FreeBSD, Gentoo Linux, HP-UX, Mandrake Linux, RedHat Fedora, S.u.S.E. Linux, SCO Unixware, Slackware Linux, Sun Solaris, Trustix Secure Linux, Ubuntu Linux, а так же ряд прикладных программ: XFree86 X11R6, IPCop, Sun Java Enterprise System, MandrakeSoft Multi Network Firewall, IPCop, MySQL, OpenPKG, CVS и т. д.

```
--- inftrees.c 2005-07-10 13:38:37.000000000 +0100
+++ ../zlib-1.2.2.orig/inftrees.c 2004-09-15 15:30:06.000000000 +0100
@@ -134,7 +134,7 @@
left -= count[len];
if (left }
- if (left > 0 && (type == CODES || (codes - count[0] != 1)))
+ if (left > 0 && (type == CODES || max != 1))
return -1; /* incomplete set */
```

Рисунок 1 патч, накладываемый на библиотеку ZLIB 1.2.2 для устранения уявзвимости

mozilla firefox, seamonkey, thunderbird множественные удаленные уязвимости

До сих пор главным мотивом использования **горящего лиса** (и его производных) была уверенность в его безопасности. Чем больше дыр обнаруживалось в IE, тем охотнее пользователи переходили к аутсайдеру. Когда популярность лиса достигла некоторой критической отметки, хакеры взялись за него всерьез и дыры полились полноводной рекой, подмочив лису его огненно-рыжий хвост. Если так будет продолжаться и дальше, то движок Mozilla (кстати говоря, расшифровываемый как Mosaic Killer – движок, на котором основан IE), не только догонит, но и перегонит IE!

За последнее время было обнаружено огромное количество дыр в Mozill'e, позволяющих выполнять произвольный код на атакуемой машине, повышать уровень привилегий JavaScript вплоть до исполнения машинного кода, запускать JavaScript даже когда он отключен, обрушивать браузер, предоставлять доступ к личным данным пользователя и т. д.

Все ошибки перечислять было бы слишком утомительно, вот только некоторые из них: хакер по кличке moz_bug_r_a4 обнаружил, что JavaScript, запущенный через компонент **EvalInSandbox** (используемый главным образом для автоматической настройки proxy), может вырваться за пределы "песочницы" и повысить свои привилегии простым вызовом **valueOf**(), обращаясь к объекту созданному вне "песочницы" и "затягивая" его внутрь. Другой исследователь Mikolaj J. Habryn обнаружил переполнение буфера в функции **crypto.signText**() из-за неправильной обработки индексов в массивах. Так же команда разработчиков столкнулась с трудновоспроизводимыми разрушениями памяти, создающими угрозу засылки shell-кода со всеми вытекающими отсюда последствиями.

Уязвимости подвержены следующие продукты: Mozilla Thunderbird 1.5.2, Mozilla SeaMonkey 1.0.1, Mozilla Firefox 1.5.3, Netscape Browser 8.0.4 (кстати говоря, более ранние версии неуязвимы). Proof-of-concept exploit'ы можно найти в базе данных "Mozilla Bugzilla", доступной только разработчикам и закрытой для публичного доступа (к счастью, присоединиться к команде может практически любой желающий).

```
<html>
<body>
	<iframe src="javascript:alert('Found by www.sysdream.com !')"></iframe>
</body>
</html>
```

Листинг 3 JavaScript выполнения, даже если он отключен

```
<html>
<body>
<iframe src="javascript:parent.document.write('Found by www.sysdream.com!')"></iframe>
</body>
</html>
```

Листинг 4 HTML-код, вызывающие крах приложения

За более подробной информацией обращайтесь по ссылкам: securityfocus.com/bid/18228, www.securityfocus.com/bid/16770/ и www.securityfocus.com/bid/17516.

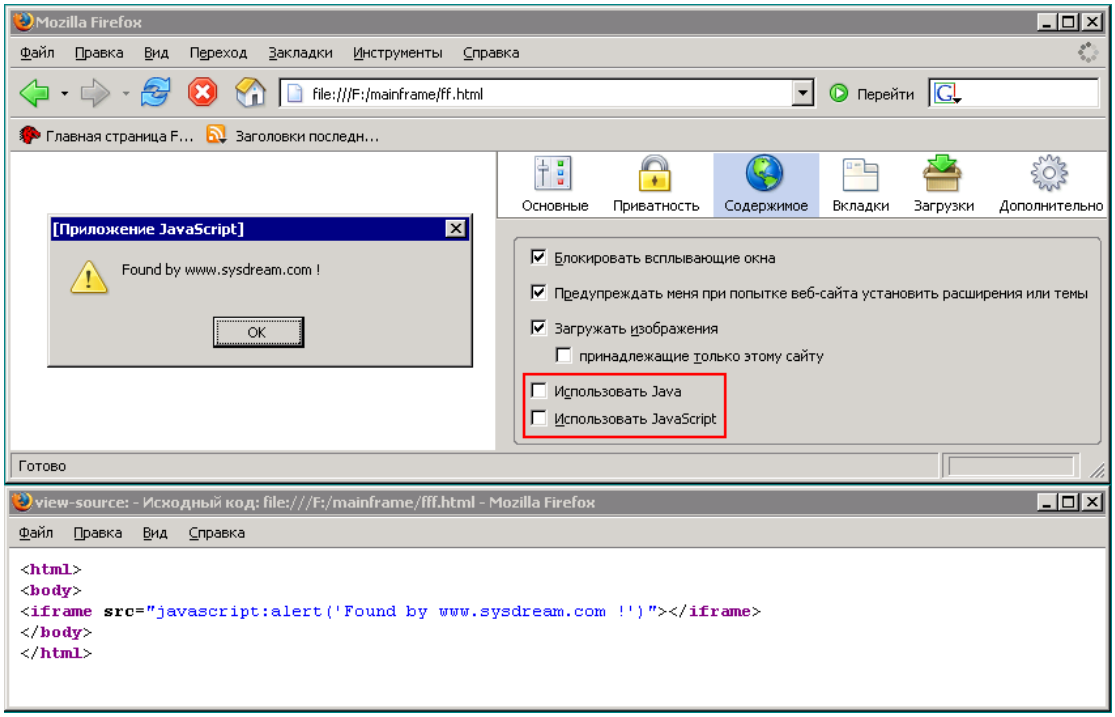

Рисунок 2 FireFox исполнят JavaScript в плавающий фреймах, даже если они запрещены

переполнение буфера в IE MHTML URI

Microsoft прилагает большие усилия по защите и вылизыванию кода **IE**, однако, поток дыр не прекращается и вот 31 мая 2006 года, два хакера Mr Niega и Hariharan обнаружили переполнение локального стекового буфера в функции **inetcomm!CActiveUrlRequest::ParseUrl**, принадлежащей библиотеке INETCOMM.DLL.

Исходный код последних версий IE транслировался компилятором Microsoft Visual Studio .NET с ключом /GS, активирующим примитивную защиту стека от переполнения, представляющую собой некоторую разновидность Stack-Guard'a причем в далеко не лучшей его "инаугурации". Никогда не разрабатывающая собственных продуктов, а только "ворующая" уже готовые (авторитетный товарищ Берзуков в своей софт-панораме об этом только и говорит, сходите на www.softpanorama.org/Bulletin/News/Archive/news078.txt, почитайте — там много интересного), Microsoft, как это часто и бывает, сама не поняла, что стащила и у кого.

В практическом плане это значит, что при затирании секретного cookie, расположенного перед адресом возврата, управление получает недокументированная функция `inetcomm!__report_gsfailure`, завершающая приложение в аварийном режиме. Короче, грохает IE. Однако, передать управление на shell-код все-таки возможно и в статье "**переполняющиеся буфера - активные средства защиты**" показано как это сделать (электронная копия лежит на моем мыщъхином сервера ftp://nezumi.org.ru/pub/stack-guards.zip).

Сами же exploit'ы выглядят довольно тривиально:

```
<html>
        <a href="mhtml://mid:AAA...AAAA">example</a>
</html>
```

Листинг 5 фрагмент exploit'a, поражающего IE (полный текст находится на www.securityfocus.com/data/vulnerabilities/exploits/18198.htm)

```
[DEFAULT]
BASEURL=
[InternetShortcut]
URL=mhtml://mid:AAA...AAA
```

Листинг 6 фрагмент exploit'a, поражающего IE (полный текст находится на www.securityfocus.com/data/vulnerabilities/exploits/18198.url_

Уязвимости подвержены следующие версии IE: 6.0, 6.0 SP1, 6.0 SP2, 7.0 beta1, 7.0 beta2, а так же, возможно и более младшие версии (версия 6.0.2800.1106, установленная у мыщъха неуязвима — сам проверял).

За дополнительной информацией обращайтесь на: www.securityfocus.com/bid/18198/.

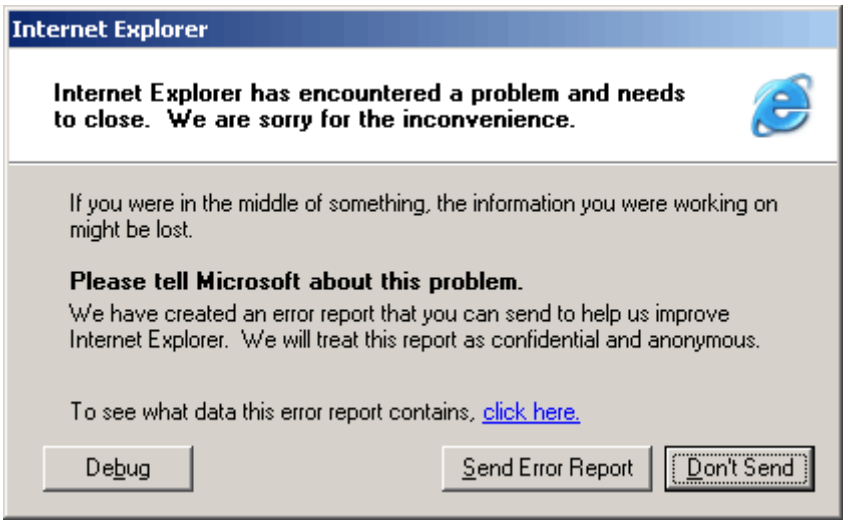

Рисунок 3 IE не выдержал атаки и весь раскрошился

переполнение буфера в Опере

Опера — это быстрый, надежный, относительно безопасный и во многом культовый бразузер (не такой, конечно, культовый как Рысь, но все-таки, сформировавший свое, особое сообщество. в частности мыщъх любит Оперу за развитый клавиатурный ввод, позволяющий вообще отказаться от мыши, что значительно ускоряет серфинг). Самое главное, Опера — это _единственный_ независимый браузер, созданный с нуля и лишенный тяжелого наследия прошлого. FireFox, основанный на движке Mozilla, и IE все еще содержащий фрагменты кода древнего Mosaic, представляют собой настоящее "кладбище" программистских технологий всех времен и народов. "Осадочные" слои кода взаимодействуют друг с другом очень сложным образом и потому ошибки вылезают то тут, то там. Добавление новых свойств требует глобального пересмотра всего кода, поскольку он уже давно превратился в сплошной клубок…

В отличии от этого, Опера сначала проектировалась (с учетом всех требований современности), а потом кодировалась с четким разделением функций каждого модуля. Такой подход упрощает отладку продукта и ликвидирует целый пласт ошибок, но... не страхует от них полностью. Программ без ошибок, увы, не бывает. В Опере они тоже встречаются. Последняя была обнаружена 13 апреля 2006, исправлена, и заново "переоткрыта" 7 июня, поскольку проблема оказалась намного серьезней, чем ожидалось.

Речь идет о классическом ***знаковом переполнении***, в после время находящимся под прицелом хакеров всего мира. Рассмотрим следующий код и попробуем найти в нем ошибку (только, чур, не подглядывать):

```
demo_singed_overflow(char *s)
{
        // объявление переменных
        int len; char buf[MAX_LEN];

        // определение длины строки
        len = strlen(s);

        // если строка влезает в буфер, копируем ее
        // в противном случае возвращаем ошибку
        if (len < MAX_LEN) strncpy(buf, s, len); else return 0;

        // тем или иным образом обрабатывает строку
        printf("%s\n",buf);
}
```

Листинг 7 пример демонстрирующий простейший случай знакового переполнения

На первый взгляд все написано правильно — мы тщательно проверяем длину строки перед копированием в буфер… О каком переполни тут можно говорить?! А вот о каком: о знаковом. Переменная `len` имеет тип `signed int` (в большинстве компиляторов `int` имеет знаковый тип по умолчанию), в то время как прототип функции `strncpy` выглядит так: `strncpy(char *dst, char *source, unsigned int count)`.

Предположим, что длина строки s превышает 2 Гбайта, тогда знаковый бит переменной `len` будет установлен в единицу и выражение (`len < MAX_LEN`) окажется ***истинным***, поскольку `len` – отрицательный, а всякое отрицательное число, как известно, меньше любого положительного. В то же самое время, функция `strncpy` трактует `len` как беззнаковый аргумент и копирует в буфер `buf` очень-очень много байт.

Чтобы избежать переполнения необходимо явно объявить переменную `len` как **`unsigned int`**, но разработчики сплошь и рядом об этом забывают. В том числе и разработчики Оперы.

Итак, значит, Опера. Возьмем английскую версию 8.52 и будем ее пытать (http://www.opera.com/download/index.dml?opsys=Windows&lng=en&ver=8.52&platform=Windows&local=y). Это последняя уязвимая версия и Opera 8.54 уже исправлена. Точнее, _как_бы_ исправлена. Беглый просмотр под дизассемблером показывает, что знаковое сравнение там по-прежнему встречается и всего лишь остается разобраться какие именно входные параметры подвержены переполнению. Короче, надо копать от забора до обеда.

Начнем с того, что файл opera.exe упакован ASPack'ом — в hex-редакторе хорошо видны секции .aspack, .adata, а PEiDE даже определяет даже версию упаковщика 2.12. Однако, при своем размере 78 Кбайт, ничего интересного он содержать не может и весь функционал сосредоточен в opera.dll с размером 2,3 Мбайт который так же упакован все тем же ASPack'ом.

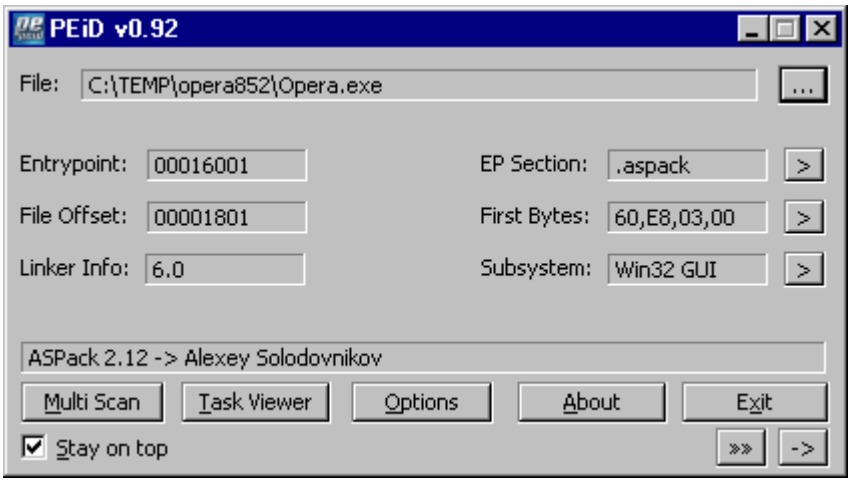

Рисунок 4 утилита PEiD определила, что opera.exe упакована ASPack'ом

Чтобы не искать готовый распаковщик (не все распаковщики умеют распаковать динамические библиотеки), воспользуемся утилитой PE-TOOLS и снимем дамп с opera.dll. Таблица импорта останется искаженной, но зачем она нам?! Мы же ведь не crack собрались писать, а проводить исследование на предмет безопасности. Главное, чтобы полученный дамп было можно загрузить в IDA Pro или hiew, а все остальное — уже дело техники!

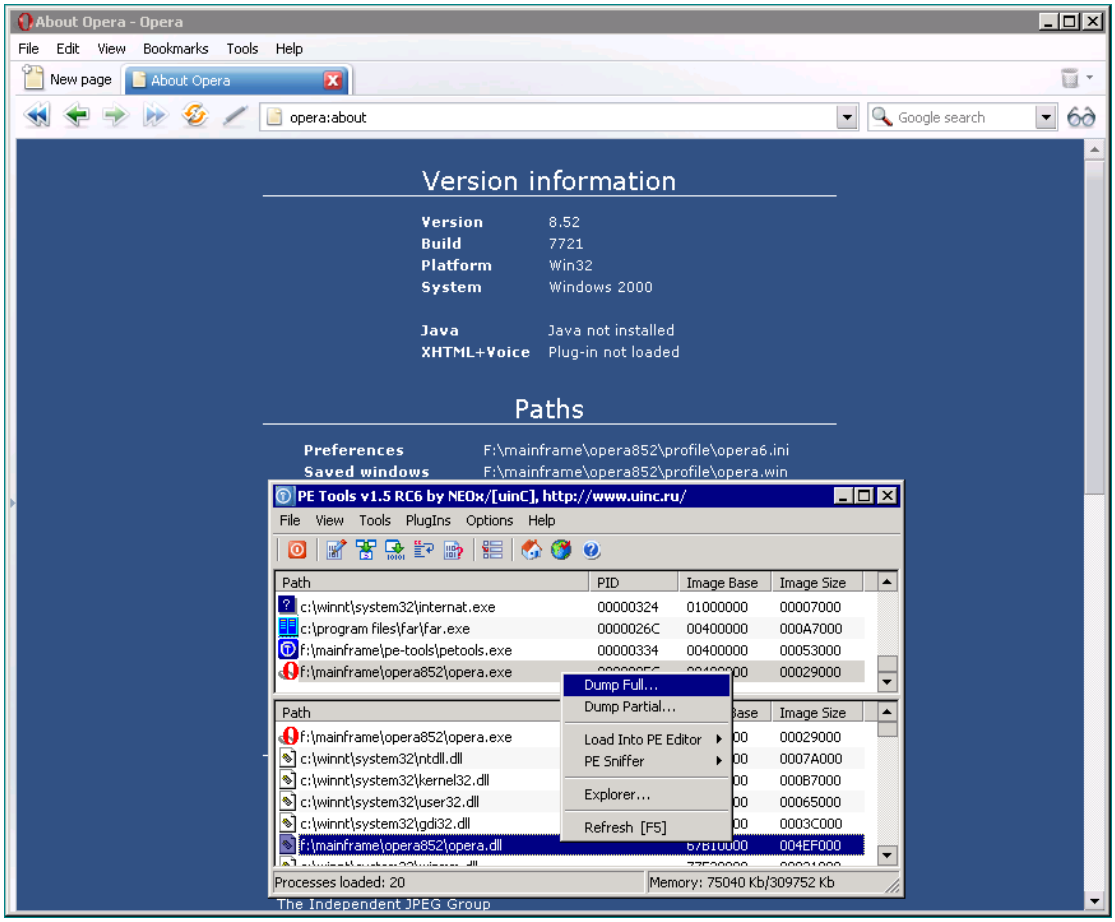

Рисунок 5 снятие полного дампа памяти с opera.dll утилитой PE Tools

Поскольку стартовый код точки входа в dll (dllentry) искажен, IDA Pro не может опознать компилятор и загрузить сигнатуры, оставляя нас без библиотечных имен, что значительно усложняет анализ. Впрочем, отождествить компилятор можно и вручную по текстовым строкам, оставленным из патриотических соображений поборников авторский прав. Просмотр дампа в hiew'e убеждает нас в том, что Опера была скомпилирована ничем иным как Microsoft Visual C++:

```
.67F3DFBC: 4D 69 63 72-6F 73 6F 66-74 20 56 69-73 75 61 6C  Microsoft Visual
.67F3DFCC: 20 43 2B 2B-20 52 75 6E-74 69 6D 65-20 4C 69 62  C++ Runtime Lib
.67F3DFDC: 72 61 72 79-00 00 00 00-52 75 6E 74-69 6D 65 20  rary    Runtime
```

Листинг 8 поиск текстовых строк в opera.dll позволяет легко и быстро отождествить компилятор

Остается только загрузить соответствующие сигнатуры. Это делается так: в меню File IDA Pro выбираем пункт Load file → FLIRT signature file, в появившимся списке имеющихся сигнатур находим строку "vc32rtf Microsoft VisualC 2-7/net runtime" и с удовлетворением жмем <ENTER>. Вот теперь с файлом можно по-настоящему работать!

Как найти места потенциального знакового переполнения? Существует множество путей. Например, можно искать все команды `JL`, `JLE` или (если этих команд окажется _слишком_ много) перебирать все вызовы строковых функций типа `_strlen`, `_strcpy`, `_wcsncpy`, обращая внимая на те из них, что соседствуют со знаковым сравнением длины копируемой строки. Это уже почти переполнение! "Почти" потому что, для совершения атаки необходимо, чтобы копируемые данные были как-то связаны с пользовательским вводом и нигде не "усекались" по пути. В принципе, задачу можно автоматизировать, написав специальный скрипт, но его разработка займет довольно продолжительно время, поскольку придется учитывать слишком много ситуаций, так что "ручная" работа все же окажется эффективнее.

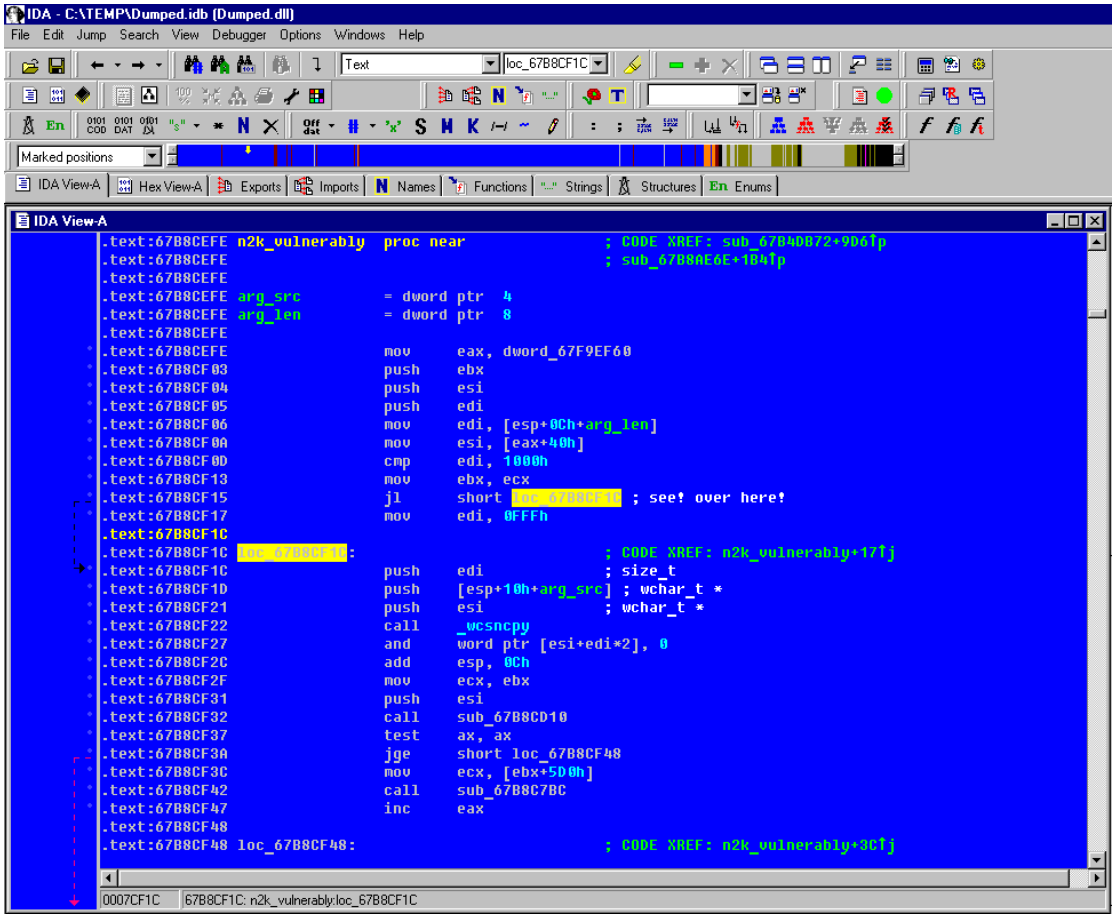

Рисунок 6 уязвимая функция глазами дизассемблере IDA Pro

Bernhard Mueller, работающий в австрийской компании SEC Consult Unternehmensberatung GmbH нашел одно из таких мест (о чем и рапортовал разработчикам Оперы, немедленно исправившим версию 8.54 и текущую 9.0), однако, в программе по-прежнему присутствует большое количество ошибок подобного типа, которые ждут своего хакера, поэтому нелишне присмотреться к уже заткнутой дырке поподробнее.

```
.text:67B8CEFE n2k_vulnerably  proc near               ; CODE XREF: sub_67B4DB72+9D6↑p
.text:67B8CEFE                                         ; sub_67B8AE6E+1B4↑p
.text:67B8CEFE
.text:67B8CEFE arg_src         = dword ptr 4
.text:67B8CEFE arg_len         = dword ptr 8
.text:67B8CEFE
.text:67B8CEFE A1 60 EF F9 67  mov     eax, dword_67F9EF60    ; pObj
.text:67B8CF03 53              push    ebx                    ; сохраняем ebx
.text:67B8CF04 56              push    esi                    ; сохраняем esi
.text:67B8CF05 57              push    edi                    ; сохраняем edi
.text:67B8CF06 8B 7C 24 14     mov     edi, [esp+0Ch+arg_len] ; edi := arg_len
.text:67B8CF0A 8B 70 40        mov     esi, [eax+40h]         ; pDestination
.text:67B8CF0D 81 FF 00 10 00  cmp     edi, 1000h             ; проверка длины arg_len
.text:67B8CF13 8B D9           mov     ebx, ecx               ; this call
.text:67B8CF15 7C 05           jl      short loc_67B8CF1C     ; see! over here!
.text:67B8CF17 BF FF 0F 00 00  mov     edi, 0FFFh             ; "усечение" arg_len
.text:67B8CF1C
.text:67B8CF1C loc_67B8CF1C:                           ; CODE XREF: n2k_vulnerably+17↑j
.text:67B8CF1C 57              push    edi                    ; size_t
.text:67B8CF1D FF 74 24 14     push    [esp+10h+arg_src]      ; wchar_t *
.text:67B8CF21 56              push    esi                    ; wchar_t *
.text:67B8CF22 E8 E9 C0 31 00  call    _wcsncpy               ; копируем имя шрифта
.text:67B8CF27 66 83 24 7E 00  and     word ptr [esi+edi*2],0 ; ставим завершающий нуль
.text:67B8CF2C 83 C4 0C        add     esp, 0Ch               ; выталкиваем аргументы
.text:67B8CF2F 8B CB           mov     ecx, ebx               ; this call
.text:67B8CF31 56              push    esi                    ; скопированный arg_len
.text:67B8CF32 E8 D9 FD FF FF  call    sub_67B8CD10           ; обрабатываем имя шрифта
.text:67B8CF37 66 85 C0        test    ax, ax                 ; все ок?
.text:67B8CF3A 7D 0C           jge     short loc_67B8CF48     ; имеем хэндл
```

```
.text:67B8CF3C 8B 8B D0 05 00 mov     ecx, [ebx+5D0h]        ; обработчик ошибки
.text:67B8CF42 E8 75 F8 FF FF call    sub_67B8C7BC           ; eax := [ecx]
.text:67B8CF47 40             inc     eax                    ; eax++
.text:67B8CF48
.text:67B8CF48 loc_67B8CF48:                              ; CODE XREF: n2k_vulnerably+3C↑j
.text:67B8CF48 5F             pop     edi                    ; восстанавливаем edi
.text:67B8CF49 5E             pop     esi                    ; восстанавливаем edi
.text:67B8CF4A 5B             pop     ebx                    ; восстанавливаем ebx
.text:67B8CF4B C2 08 00       retn    8                      ; выталкиваем аргументы
.text:67B8CF4B n2k_vulnerably endp                           ; конец процедуры
```

Листинг 9 дизассемблерный листинг уязвимой функции n2k_vulnerably со знакомым переполнением

Как мы видим, эта процедура (назовем ее **n2k_vulnerably**) принимает два аргумента– указатель на строку и длину этой строки, которая затем сравнивается _знаковой_ операцией с константой 1000h, в результате чего допустимый диапазоны длин строки оказывается равны: [0, 1000h) и (7FFFFFFFh, FFFFFFFFh]. Затем эта строка копируется внутрь какой-то структуры (по всей видимости — объекта, поскольку присутствуют вызовы типа `this call`), который передается функции `sub_67B8CD10` для обработки.

Очевидно, что передав строку размером 2 Гбайта или выше, мы затрем добрую половину адресного пространства приложения, отчего ему станет очень нехорошо. В лучшем случае дело кончится крэшем, в худшем - передачей shell-кода и захватом управления. Вот только передать такую длинную строку по сети очень проблематично. Даже если жертва сидит на DSL атака растянется на несколько часов и пользовать, скорее всего, просто закроет Оперу или соединение будет выбито по тайм-ауту.

Но... к счастью для хакеров, разработчики Оперы допустили _двойную_ ошибку, использовав расширение слова до двойного слова. Со знаком разумеется. Куда же без него! (По правде говоря, за разработчиков это сделал компилятор, разработчики лишь использовали неправильное преобразование типов, но пользователям Оперы от этого ничуть не легче). Причем, это преобразование осуществляется в функции, вызывающей `n2k_vulnerably`!

Ниже приведен ее ключевой фрагмент с некоторыми сокращениями:

```
.text:67B8AF5D loc_67B8AF5D:                              ; CODE XREF: sub_67B8AE6E+F8↓j
.text:67B8AF5D 89 46 2C       mov     [esi+2Ch], eax
.text:67B8AF60 EB 16          jmp     short loc_67B8AF78
.text:67B8AF62
.text:67B8AF62 loc_67B8AF62:                              ; CODE XREF: sub_67B8AE6E+E2↑j
.text:67B8AF62 0F BF 45 EC    movsx   eax, [ebp+ var_length_ovfl]
.text:67B8AF62                ; вот оно! расширение слова, хранящего длину копируемой
.text:67B8AF62                ; строки, до двойного слова _со_ _знаком_
.text:67B8AF62                ; если длина строки  превышает 7FFFh байт,
.text:67B8AF62                ; то и результат превышает 7FFFFFFFh
.text:67B8AF62
…
.text:67B8B012 FF 76 08       push    dword ptr [esi+8]      ; передаем arg_src функции
.text:67B8B015 8B 0D 18 EF F9 mov     ecx, dword_67F9EF18    ; this
.text:67B8B01B 8D 85 B0 FD FF lea     eax, [ebp+var_250]     ; получаем arg_len
.text:67B8B021 50             push    eax                    ; передаем arg_len функции
.text:67B8B022 E8 D7 1E 00 00 call    n2k_vulnerably         ; вызываем функцию
```

Листинг 10 уязвимый код, вызывающий функцию n2k_vulnerably и передающий ей в качестве arg_len знаковое слово, расширенное (со знаком!) до двойного слова

Код функции довольно громоздкий и потому приведен не полностью. Под сокращение, в частности, попала передача расширенного EAX в переменную `[EBP+var+250]`, но тот факт, что она передается, сокращает длину переполняемой строки всего до 8000h байт или 32 Кбайт, что вполне приемлемо для атаки.

Остается выяснить — что же это за строка такая и как она связана с пользовательским вводом (и связана ли вообще). Ответ дает функция **sub_67B8CD10**, вызываемая из `n2k_vulnerably`.

Вот ее дизассемблерный текст:

```
.text:67B8CD10 sub_67B8CD10   proc near
.text:67B8CD5C FF 74 24 14    push    [esp+0Ch+arg_0]        ; передаем "свой"аргумент…
.text:67B8CD60 E8 D3 FE FF FF call    sub_67B8CC38           ; …подфункции sub_67B8CC38
…

; // дизассемблерный листинг подфункции sub_67B8CC38
```

```
.text:67B8CC38 sub_67B8CC38   proc near      ; CODE XREF: sub_67B8CB3C:loc_67B8CBA5↑
.text:67B8CC38                               ; sub_67B8CD10+50↓p ...
.text:67B8CC38
.text:67B8CC38 arg_0   = dword ptr  8
.text:67B8CC38
.text:67B8CC38 56              push    esi
.text:67B8CC39 8B 74 24 08     mov     esi, [esp+arg_0]
.text:67B8CC3D 68 94 24 F5 67 push    offset aSerif         ; "SERIF"
.text:67B8CC42 56              push    esi
.text:67B8CC43 E8 3D 30 0A 00 call    sub_67C2FC85          ; обработка шрифта
.text:67B8CC48 59              pop     ecx
.text:67B8CC49 85 C0           test    eax, eax
.text:67B8CC4B 59              pop     ecx
.text:67B8CC4C 74 04           jz      short loc_67B8CC52
.text:67B8CC4E 33 C0           xor     eax, eax
.text:67B8CC50 5E              pop     esi
.text:67B8CC51 C3              retn
.text:67B8CC52
.text:67B8CC52 loc_67B8CC52:                 ; CODE XREF: sub_67B8CC38+14↑j
.text:67B8CC52 68 88 24 F5 67 push    offset aSansSerif_0   ; "SANS-SERIF"
.text:67B8CC57 56              push    esi
.text:67B8CC58 E8 28 30 0A 00 call    sub_67C2FC85          ; обработка шрифта
.text:67B8CC5D 59              pop     ecx
.text:67B8CC5E 85 C0           test    eax, eax
.text:67B8CC60 59              pop     ecx
.text:67B8CC61 74 05           jz      short loc_67B8CC68
.text:67B8CC63 6A 01           push    1
.text:67B8CC65
.text:67B8CC65 loc_67B8CC65:                 ; CODE XREF: sub_67B8CC38+43↓j
.text:67B8CC65                               ; sub_67B8CC38+58↓j
.text:67B8CC65 58              pop     eax
.text:67B8CC66 5E              pop     esi
.text:67B8CC67 C3              retn
.text:67B8CC68
.text:67B8CC68 loc_67B8CC68:                 ; CODE XREF: sub_67B8CC38+29↑j
.text:67B8CC68 68 80 24 F5 67 push    offset aFantasy       ; "FANTASY"
.text:67B8CC6D 56              push    esi
.text:67B8CC6E E8 12 30 0A 00 call    sub_67C2FC85          ; обработка шрифта
```

Листинг 11 переполняющая строка передается функции, обрабатывающей шрифты, это позволяет предположить, что строка представляет собой имя шрифта

Имена шрифтов сразу же бросаются в глаза. Ага! Значит, эта функция управляет стилем оформления страницы, загружая соответствующий шрифт. И это хорошо, потому что шрифты мы можем принудительно менять через CSS. Главное, чтобы длина имени шрифта (необязательно реально существующего, вполне сойдет и фиктивный), превышала 32 Кбайта. Вместе с именем может быть передан shell-код, который пойдет гулять по куче (динамической памяти), основательно ее затирая. Ну, а технику переполнения кучи мы уже неоднократно рассматривали в прошлых номерах хакера.

Ниже приведен код простейшего exploit'a, "роняющего" Оперу версий 8.52 и более младших (впрочем, атака может не сработать, если CSS отключен):

```
<STYLE type=text/css>A { FONT-FAMILY: 35000x'A' } </STYLE>
```

Листинг 12 код простейшего CSS-exploit'a вызывающий обрушение Оперы

Для ликвидации уязвимости необходимо скачать новую версию Оперы (благо, она бесплатна) или… ликвидировать дыру собственными руками! Действительно, зачем перекачивать несколько мегабайт по модему, переустанавливать и т. д., когда нас и текущая версия вполне устраивает. Как известно, с каждой новой версией программное обеспечение все толстеет и толстеет, становится неповоротливым, не принося никаких существенно новых фич.

Всего-то и нужно — заменить 67B8CF15: JL loc_67B8CF1C (7Ch 05h) на JB loc_67B8CF1C (72h 05h). Если бы файл был неупакован, это можно было бы сделать прямо в hiew'e, а так… нет, конечно, распаковать opera.dll вполне возможно, тем более, что ASPack – упаковщик вполне известный, но… не всегда распаковка проходит успешно и потом начинают вылезать баги в разных местах.

Мы пойдем другим путем, прибегнув к on-line patch'y, то есть запустим opera.exe, дождемся распаковки opera.dll и исправим байты прямо в оперативной памяти! Аналогичный подход может быть применен и к другим программам, не только к Опере, поэтому ниже приводится исходный текст простейшего универсального on-line patcher'a.

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

#define MAX_SIZE              16      // длина буфера для патча
main(int argc, char **argv)
{

        // объявляем переменные
        STARTUPINFO si; PROCESS_INFORMATION pi; DWORD N,FL;

        // имя исполняемого файла для запуска
        unsigned char                 name[]="opera.exe";
        unsigned char                 buf[MAX_SIZE];

        // указываем что и где мы будем патчить
        unsigned char                 jmp_from[] = "\x7Ch\x05h";    // old
        unsigned char                 jmp_to[]   = "\x72\x90";      // new
        void*                         jmp_off    = (void*)0x67B8CF1C;// address

        printf("loading & patching...\n");

        // инициализация всех структур данных
        si.cb = sizeof(si); memset(&si,0,sizeof(si)); memset(buf,0,MAX_SIZE);

        // запускаем оперу на выполнение, для простоты не передавая
        // ей аргументы командой строки
        CreateProcess(0, name, 0, 0, 0, NORMAL_PRIORITY_CLASS, 0, 0, &si, &pi);

        // ждем 1 сек, в течении которых opera.dll должна загрузиться и распаковаться
        // возможно, на медленных ЦП это значение придется увеличить
        Sleep(1000);

        // проверка версии оперы перед патчем
        ReadProcessMemory(pi.hProcess, jmp_off, buf, strlen(jmp_from), &N);
        if (N != strlen(jmp_from)){printf("-ERR:reading memory\x7\n"); return -1;}
        if (strcmp(jmp_from,buf)){printf("-ERR:incorrect version!\x7\n");return -1;}

        // падчим условный знаковый переход JL на беззнаковый JB
        WriteProcessMemory(pi.hProcess, jmp_off, jmp_to, strlen(jmp_to), &N);
        if (N != strlen(jmp_to)){printf("-ERR:writing memory\x7\n");return -1;}

        // говорим OK и сваливаем
        printf("OK\nall ok \n");
}
```

Листинг 13 исходный текст on-line patcher'a opera_loader.c

Естественно, теперь каждый раз придется запускать не opera.exe, а opera_loader.exe. Впрочем, чтобы не напрягаться достаточно всего лишь сменить ярлыки и файловые ассоциации. Правда, в силу небольшой конструктивной недоработки on-line patcher'a он не передает Опере аргументов командой строки, поэтому если она установлена основным браузером по умолчанию, то htm-файлы открываться не будут! Используйте drag-n-drop или доработайте patcher "напильником" до законченной конструкции.

Главное, что мы заткнули дыру своими собственными силами и нам не понадобились никакие обновления. Но если захочется получить дополнительную информацию по уязвимости, то милости просим посетить следующие ресурсы: http://www.securityfocus.com/bid/17513, http://www.greymagic.com/security/advisories/gm008-op/, а мыщъх тем временем идет спать, сожрав феназепам, подтолкнув под себя хвост и нацепив наушники в которых разворачивается aus der tiefe.