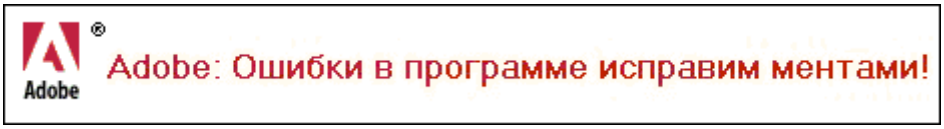
exploits review
12h выпуск

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

текущий обзор посвящен компании Abode, в продуктах которой обнаружено множество знаменательных дыр, допускающих возможность удаленного захвата управления, чем уже с успехом воспользовались черви, вызвав очередную волну эпидемии.



Acrobat Reader – удаленное выполнение кода через mailto

brief: 20 сентября 2007 года хакер по кличке pdp <pdp.gnucitizen@googlemail.com> (создатель сайта www.gnucitizen.org) обнаружил серьезную дыру в Acrobat Reader'e, обнародовав ее в своем блоге (www.gnucitizen.org/blog/0day-pdf-pwns-windows). Abode отреагировала на эту предъяву лишь 22 октября (при том что первый публичный exploit появился 16 октября!), а на следующий день, 23 октября, корпорация Symantec изловила живого червя Trojan.Pidief.A (он же EXPL_PIDIEF.B [Trend], он же Troj/PDFex-A [Sophos]), написанного китайкой Elia Florio и распространяющегося по электронной почте вместе с вложениями invoice.pdf, your_bill.pdf, bill.pdf и statemet.pdf (см. symantec.com/business/security_response/writeup.jsp?docid=2007-102310-3513-99&tabid=2 и symantec.com/enterprise/security_response/weblog/2007/10/when_pdfs_attack_again.html). Это довольно необычная дыра, возникающая на стыке Adobe Acrobat Reader'a с MS IE7, в процессе определения приложения, ответственного за обработку протокола mailto. Чтобы форсировать автоматический запуск исполняемого приложения достаточно создать: "<</URI(mailto:test%../../../../../../../../windows/system32/calc.exe".cmd)/S/URI>>" и тогда на XP|Server 2003/IE7 при локальном открытии pdf-документа автоматически запустится "Калькулятор" без каких бы то ни было дополнительных действий со стороны жертвы (кликать по ссылке необязательно). Открытие документа через pdf-ActiveX plug-in обойти сложнее: seclists.org/bugtraq/2007/Oct/0213.html). Более подробную информацию по дыре можно получить: www.securityfocus.com/bid/25748/, www.symantec.com/avcenter/attack_sigs/s22634.html и www.symantec.com/security_response/vulnerability.jsp?bid=25748;

targets: уязвимости подвержены только системы с XP|Server 2003/IE7 и стандартным почтовым клиентом по умолчанию. Под W2K дыра неактуальна (см. рис. 1). Версия Acrobat'a не имеет значения и шиза косит все билды подряд от 3.x до 8.x;

exploit: публичная версия exploit'a, запускающего "Калькулятор", выложена на security.fedora-hosting.com/0day/pdf/pdf_poc.pdf, а рядом с ней лежит и ее описание: security.fedora-hosting.com/0day/pdf/pdf_poc.txt. Все это добро на всякий случай скопировано на мой мыщъхиный сервер: http://nezumi.org.ru/souriz/hack/ pdf_poc.7z;

solutions: пользователи Acrobat Reader'a версий 7.x-8.x могут установить заплатку от Abode или, следуя предписаниям последней, отключить обработку протокола mailto в реестре: www.adobe.com/support/security/bulletins/apsb07-18.html. Остальным же рекомендуется просто установить любой другой мыльник, назначив его почтовым клиентом по умолчанию;

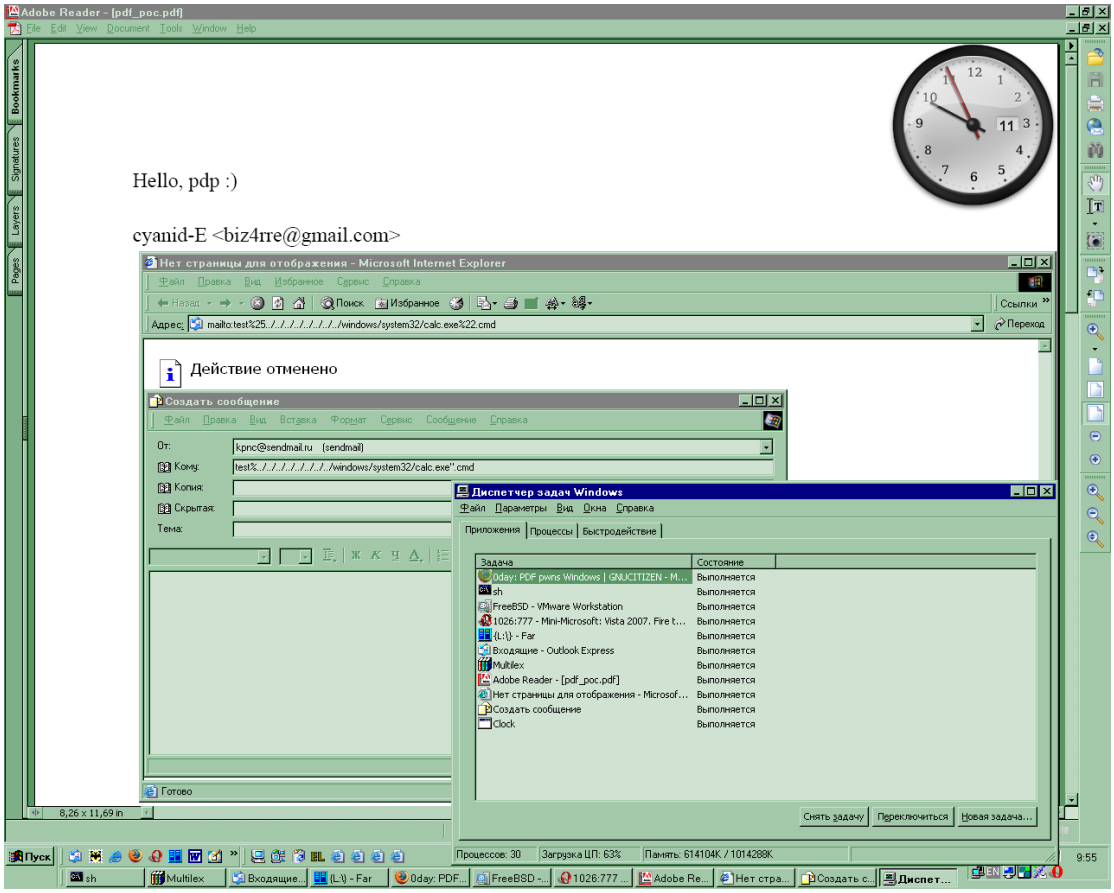


Рисунок 1 под W2K дыра в Acrobat'е не функцилирет

Adobe Flash Prayer — удаленное выполнение кода

brief: 15 июля 2007 года клевые китайские хакеры Stefano DiPaola, Elia Florio (создательница вируса Trojan.Pidief.A) и Giorgio Fedon с Ph4nt0m Security Team (www.ph4nt0m.org) лихо подломали Adobe Flash Player 9.0.45.0, используемый кучей браузеров (в том числе и на игровых консолях!), а наши браться по разуму luoluo и yunshu заточили под это дело нехилый exploit: http://www.icylife.net/yunshu/attachment.php?id=5 с shell-кодом внутри. История поимела продолжение 17 июля, когда Henke37 обнаружил множественные signed/unsigned ошибки переполнения в flv-парсере, а yunshu заточил под это дело еще один exploit. 31 октября эхо атаки докатилось и до Оперы, а точнее до Adobe Flash Player'a, работающего в ее контексте. Фрагмент кода, ответственный за переполнение, приведен ниже:

```
3019CE18 C1E9 02       SHR ECX, 2             ; // ECX := ECX * 4
3019CE1B 83E2 03       AND EDX, 3
3019CE1E 83F9 08       CMP ECX, 8             ; // IF (ECX < sizeof(DWORD))
3019CE21 72 29         JB Flash9c.3019CE4C    ;              GOTO somewhere
3019CE23 F3:A5         REP MOVSD              ; копируем буфер двойными словами
```

Листинг 1 уязвимый код

как видно, парни из Abode решили копировать буфер двойными словами (это быстрее) и потому тщательно проверили, что на кассе есть по меньше мере 8 байт, а вот о больше мере впопыхах как-то позабыли, позволив инструкции REP MOVSD чесать по куче по весь аллюр. А что у нас интересного в куче? Ну вот, например, указатели на виртуальные функции (Flash Player написан на плюсах), одна из которой вызывается вслед за переполнением:

```
300ED77D 85C9          TEST ECX, ECX                ; ECX - указ. на объект
300ED77F 0F84 00070000 JE Flash9c.300EDE85          ; проверка что объект есть
300ED785 8B01          MOV EAX, DWORD PTR DS:[ECX]  ; eax=указатель на объект
```

```
300ED787 FF50 20        CALL DWORD PTR DS:[EAX+20]    ; вызываем вирт. функцию
```

Листинг 2 вызов виртуальной функции после переполнения

мудрый китаец yunshu, покурив хорошей травы, написал Java-скрипт, выделяющий до фига динамической памяти (техника, известная под кодовым именем "heap spray"), заполнив все адресное пространство двойными словами 0C0C0C0Ch (ну и shell-кодом), при этом по адресу 0C0C0C0Ch (и дальше) оказывается записано 0C0C0C0Ch, поэтому, MOV EAX, DWORD PTR DS:[ECX] грузит в EAX значение 0C0C0C0Ch, а CALL DWORD PTR DS:[EAX+20] вызывает код, расположенный по адресу "0C0C0C0Ch + 20h", где приютилась вереница 0C0C0C0Ch за которой следует shell-код. Поскольку, на языке x86 ЦП последовательность 0C0Ch означает 0R AL,0Ch (т.е. имеет "нулевой побочный эффект"), то выполнение благополучно "докатывается" до shell-кода, независимо от его точного расположения в памяти, в результате чего мы получаем универсальный exploit. Слава китайцам!!!

targets: уязвимость подтверждена в следующих версиях Flash Player'a: 7.0.69.0, 8.0.34.0, 9.0.28.0, 9.0.31.0, 9.0.45.0, а так же некоторых других. Затронуты практически все платформы: Windows, Linux, Mac OS X и даже игровая консоль Nintendo Wii!

exploit: реально работающий exploit (вместе с объяснениями принципов его работы) лежит на packetstorm.linuxsecurity.com/0707-exploits/07162007-flash_flv_9.0.45.0_exp.zip, а на http://www.securityfocus.com/data/vulnerabilities/exploits/24856.zip — его зеркало;

solution: установить обновление от Abode (adobe.com/support/security/bulletins/apsb07-12.html) или же от конкретного производителя браузера, например, для Оперы: http://www.opera.com/support/search/view/868/;

Рисунок 2 а клево быть китайцем!

Photoshop — множественные переполнения буфера

brief: 10 октября 2007 года хакер Marsu <Marsupilamipowa@hotmail.fr> обнаружил очередную ошибку переполнения буфера в Photoshop'e/Illustrator'e. На этот раз — в парсере BMP, DIB и RLE-файлов. Вполне классическая ситуация. Тривиальное стековое переполнение, допускающее возможность передачи управления на shell-код, расположенный здесь же, в переполняемом буфере. Под W2K атака реализуется на ура без каких бы то ни было осложнений. XP SP2 с аппаратной поддержкой DEP, задействованной для всех приложений (по умолчанию DEP включен только для системных процессов и некоторых "родных" Windows-программ типа IE) ломается чуть-чуть сложнее и для обхода защиты от неисполняемого стека нам приходится прибегать к атаке типа "return2libc" (см. http://nezumi.org.ru/zq-nx.uncensored.zip), а вот Висла и Server 2008 уже представляют перед хакером серьезную преграду, поскольку рандомизуют адресное пространство и приходится не по-детски извращается, чтобы передать управление на shell-код (хинт: исполняемые файлы Photoshop'a и Illustrator'a собраны без перемещаемых элементов, а потому их местоположение в памяти всегда постоянно). Перспективы атаки в целом представляются не слишком обнадеживающими — Photoshop с Illustrator'ом не самые распространенные программы среди пользователей… Зато для целенаправленных атак на различные издательства и дизайн-студии разворачивается совсем нехилое поле для активной деятельности, тем более что расширение файла не играет никакой роли и для маскировки BMP можно переименовать в JPG (рассылка BMP вызывает определенные подозрения, поскольку это не самый популярный формат среди профи). Более подробную информацию о дыре можно нарыть на: www.securityfocus.com/bid/23621;

targets: дыра подтверждена в Adobe Photoshop CS2/CS3, Adobe Illustrator CS3 и Adobe GoLive 9;

exploit: proof-of-concept exploit, заточенный хакером Marsu и протестированный под XP SP2, лежит на www.securityfocus.com/data/vulnerabilities/exploits/23621.c. В случае успешной работы он запускает "Калькулятор";

solution: установить обновление для соответствующего продукта, загрузив его с сервера www.adobe.com;

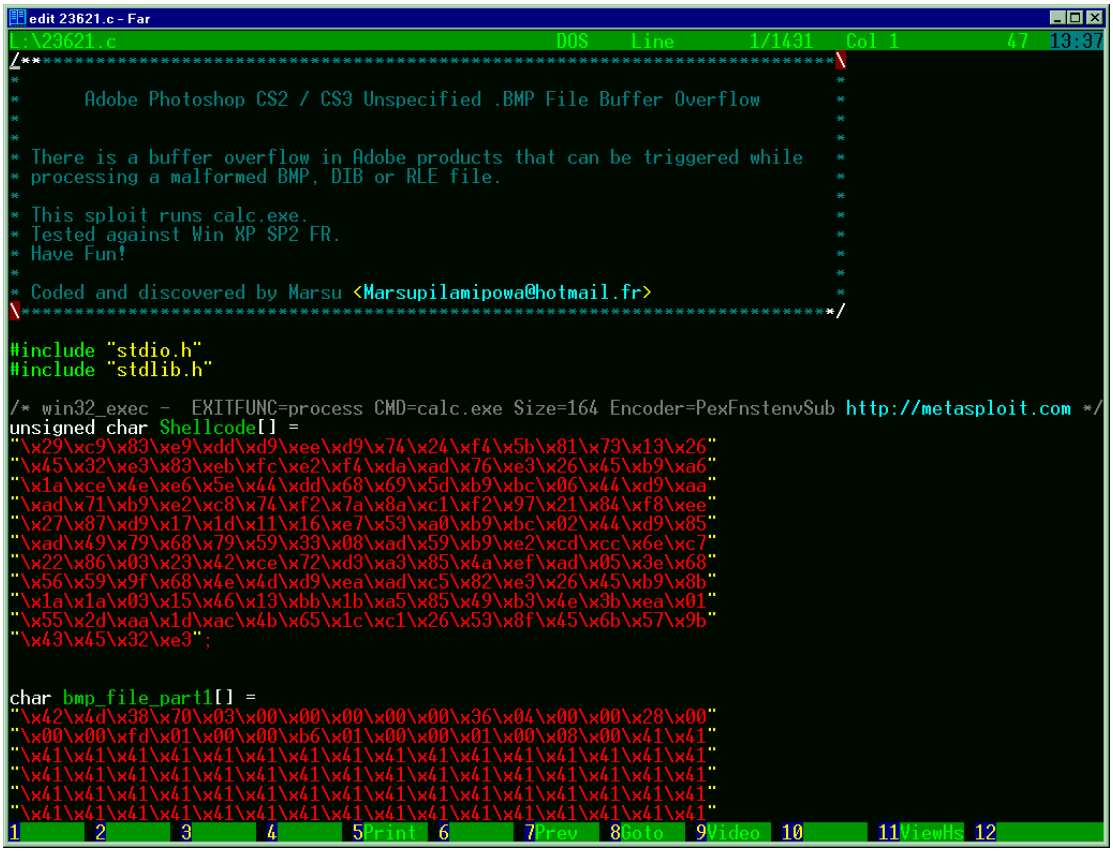


Рисунок 3 exploit от Marsu в FAR'e

full disclose
PageMaker — переполнение длинными именами фонтов

brief: 9 октября 2007 года китаец… нет не китаец, а японец! Японец Tan Chew Keong обнаружил в PageMaker'e ошибку переполнения стека, вызываемую длинными именами шрифтов, сохраненными в pmd-файлах и обнародовал exploit (запускающий "Калькулятор"), но спустя короткое время убрал его из публичного доступа (см. рис. 4). Вот су… суровый человек, однако! Впрочем, мыщъх написал свой собственный (но об этом — ниже). Дефект сидит в динамическое библиотеке MAIPM6.dll и не представляет из себя ничего интересного. Ну, подумаешь, забыли проверить длину фонта перед его копированием в буфер. PageMaker так же не относится к широко распространенным программам и об этой ошибке можно было бы и забыть, по ходу дела списав ее в утиль, если бы не одно важное обстоятельство: PageMaker широко используется в издательствах и других крупных фирмах, обменивающихся макетами и файлами презентаций в формате pmd, причем количество не залатанных машин, на которых установлен дырявый PageMaker, не поддается никакому учету! Поскольку, нарыть в сети техническую информацию об уязвимости не удалось, мыщъх (вовлеченный в очередной виток пен-тестинга), скрутил штопором хвост и занялся исследовательской деятельностью самостоятельно. (общее представление о проблеме можно получить на http://www.securityfocus.com/bid/25989/ или другом информационном ресурсе подобного рода, перепечатывающим обрывки информации);

targets: в настоящее время уязвимость подтверждена в Adobe PageMaker 7.0.1/7.0.2, другие версии не проверялись, но есть все основания полагать, что дыра присутствует и там;

exploit: демонстрационный exploit, вызывающий инструкцию INT 03h, лежит на: http://nezumi.org.ru/souriz/hack/nezumi-pagemaker-MAIPM6-bug.7z. Он протестирован под W2K и должен работать под XP в конфигурации по умолчанию (если задействован

аппаратный DEP для всех приложений, exploit следует доработать, задействовав атаку типа return2libc);

solution: установить обновленную версию динамической библиотеки MAIPM6.dll, выпущенную производителем: http://www.adobe.com/support/security/bulletins/downloads/MAIPM6.zip;

Рисунок 4 суровый японский самурай Tan Chew Keong удалил свой exploit из публичного доступа

full disclose:

Для экспериментов нам понадобиться PageMaker 7.x, 30-дневную испытательную версию можно бесплатно скачать у Adobe, предварительно зарегистрировав там халявный аккаунт (https://www.adobe.com/cfusion/tdrc/index.cfm?loc=en%5Fus&product=pagemaker) или найти его на диске, прилагаемом к журналу, сэкономив тем самым 64 Мегабайт трафика (см. рис. 5).

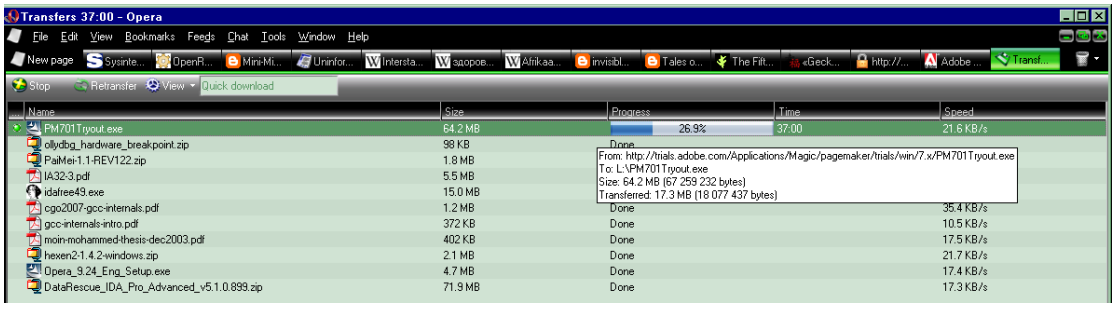


Рисунок 5 тянем-потянем Abode Mage Maker и никак не утянем

Установка не вызывает никаких проблем и проходит без сучка и задоринки (см. рис. 6). Чтобы не засирать машину всяким барахлом, рекомендуется использовать VM Ware или другой эмулятор подобного типа, особенно, если мы не собираемся работать с PageMaker'ом в дальнейшем.

Рисунок 6 устанавливаем PageMaker на свою машину (ну, или, если не на свою, так на чужую, — в общем-то никакой разницы нет)

И вот этот монстр установлен! Запускаем Pm70.exe и выбираем какой-нибудь шаблон проекта, немного поиздевавшись над которым (см. рис. 7), сохраняем на диск с расширением *.pmd (например, "nezumi.pmd"). На этом подготовительные мероприятия можно считать законченными. Мы получили файл над которым будем медитировать на протяжении всей последующей статьи.

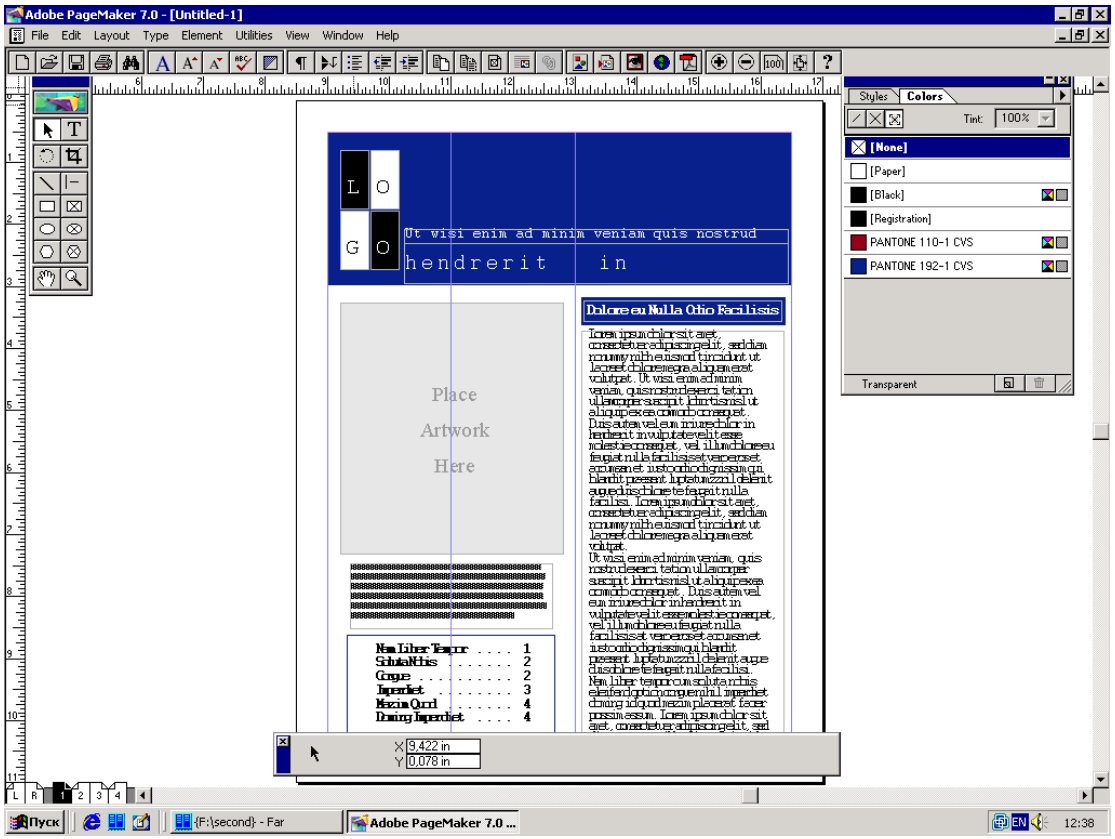


Рисунок 7 создаем pmd-файл для экспериментов (не путать с экскрементами!)

Открываем "nezumi.pmd" в hiew'e и ищем там какое-нибудь имя шрифта, например, "Courier" и ведь находим его по смещению 24B0h (см. рис. 8). Естественно, в зависимости от структуры файла это смещение может и другим, но это неважно. Пока лишь просто запомним его (запишем на бумажке), попутно обратив внимание на непонятную, но очень интересную структуру а'ля "1 :) :) :) :) :)" расположенную чуть ниже. Позже она нам _очень_ понадобиться!

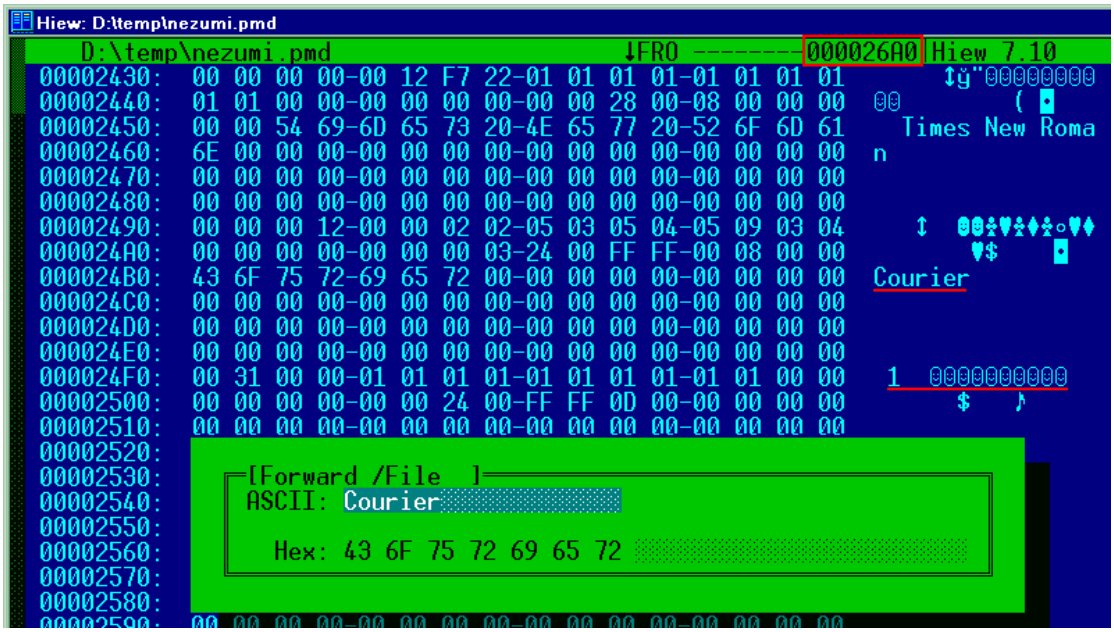


Рисунок 8 поиск имени шрифта в pmd-файле

Впрочем, не будем забегать вперед и вернемся в настоящее время. А в настоящем времени мы нажимаем <F3> (Edit) и дописываем к Курьеру длинный ряд шестерок, затирающий своим хвостом структуру "1 :) :) :) :) :)" – это чтобы если переполнять, то уж наверняка (см. рис. 8). Почему именно шестерки? Ну, нравятся они мне! Число зверя и поза 69 — навевают приятные ассоциации! (А еще — шестой день суббота, шестой месяц – начало каникул, etc). Большинство хакеров используют для переполнения последовательность "AAA…AAA", что работает ничуть не хуже, так что никакого произвола в конечном счете нет. Каждый выбирает тот сорт пива, который ему больше всего нравится.

Кстати, прежде чем затирать файл, его рекомендуется скопировать ну, например, в "nezumi-3.pmd", чтобы сохранить оригинал для возможности отката. Почему "nezumi-3"? Потому, что четное. Следовательно, счастливое ;) Ладно, не буду грузить поверьями (особенно теми, в которые сам ни хвоста не верю), а лучше сразу покажу конечный результат (см. рис. 9). Сохраняем изменения по <F9> и выходим.

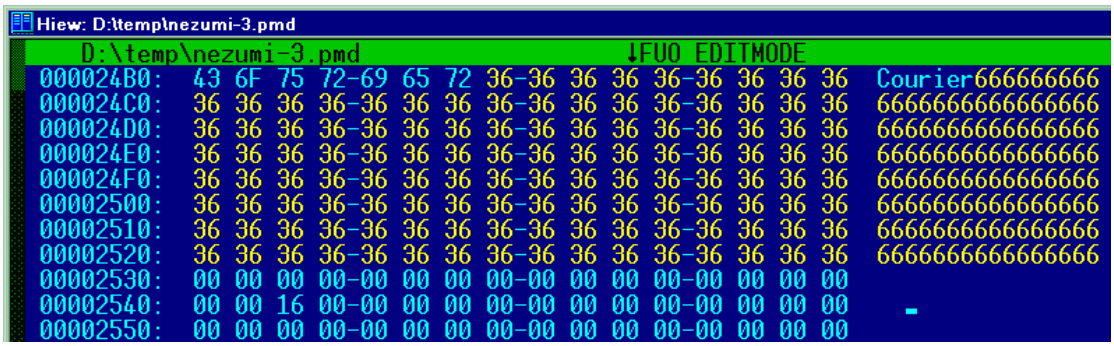


Рисунок 9 pmd-файл после затирания хвоста имени шрифта

Загружаем "nezumi-3.pmd" в PageMaker (если нас спросят "open the most recent version?" отвечаем "no, fuck you") и получаем крэш: "инструкция по адресу 0x36363636 обратилась к памяти по адресу 0x36363636. память не может быть read" (см. рис. 10). Что это значит?!

Офигеть — вот что это значит! Это как серпом по яйцам! 36h – hex-код ASCII-символа "6". Инструкция по адресу 36363636h не собиралась читать память по адресу 36363636h, это она сама не может быть прочитана процессором, потому что такой памяти нет, точнее она не выделена. Тот факт, что регистр EIP (указатель команд) принял значение 36363636h говорит о том, что затираемый код перекрыл собой адрес возврата из функции на который мы можем воздействовать, передавая управление туда, куда нам заблагорассудиться!

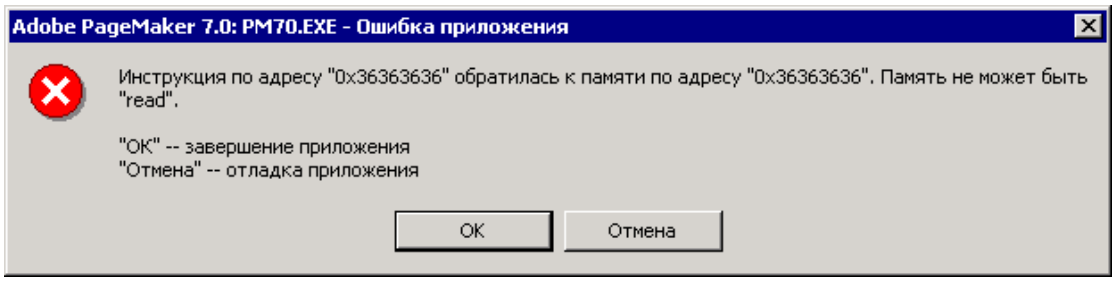


Рисунок 10 крэш PageMaker'а, вызванный файлом nezumi-3.pmd

Для дальнейших экспериментов нам понадобится какой-нибудь отладчик, например, бесплатный Olly Debugger (http://www.ollydbg.de) или его хакнутый клон Immunity Debugger с поддержкой скриптов, написанных на Питоне (http://debugger.immunityinc.com/register.html). Мыщъх ненавидит Питон еще с того дня, когда обнаружил, что тот относится к форматированию листинга так же строго, как Си — к соблюдению регистра. Впрочем, при желании можно обойтись и штатным Доктором Ватсоном (путь самурая).

Устанавливаем OllyDbg и назначаем его Just-in-Time Debugger'ом, так же называемом Post-Morten отладчиком, то есть, отладчиком, вызываемом после краха приложения (Options → Just-In-Time Debugging, Make Olly Just-In-Time-Debugger, Confirm before attaching), после чего повторяем открытие файла "nezumi-3.pmd" вновь, нажимая "отмену" для вызова отладчика (см. рис. 11).

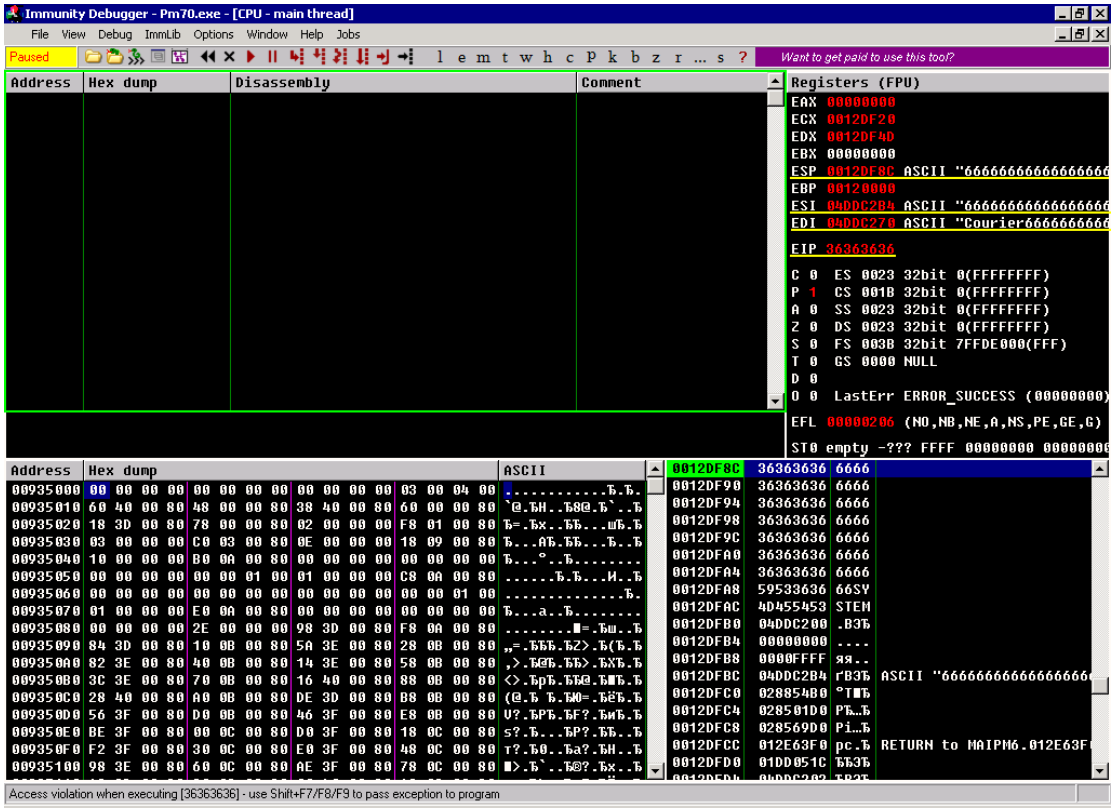


Рисунок 11 сразу же после крэша

Окно "CPU" пусто и это нормально, поскольку памяти, на которую указывает регистр EIP (равный 363636363h) как уже говорилось нет и показывать здесь нечего. Смотрим на стек.

Регистр ESP (равный 0012DF8Ch) указывает на строку "6666…", которой забит верх стека. Регистр ESI также указывает на "666…", но судя по его значению (04DDC2B4h) эта копия строки расположена не в стеке, а в куче (выяснить это нам поможет карта памяти, вызываемая по <Alt-M>) — это очень важное обстоятельство, которое нас выручит в дальнейшем. Регистр EDI также смотрит в кучу (04DDC270h), но указывает непосредственно на само имя шрифта "Courier66666…", тогда как ESI – куда-то на его середину.

Переходим к следующей фазе атаки — попробуем выяснить какой именно по счету байт перекрывает адрес возврата. Один из способов сделать это — воспользоваться "бегущей строкой", изменяющейся на всем своем протяжении. Для грубого определения дислокации адреса возврата поверх шестерок запишем "AAA…BBB….CCC…" (см. рис. 12).

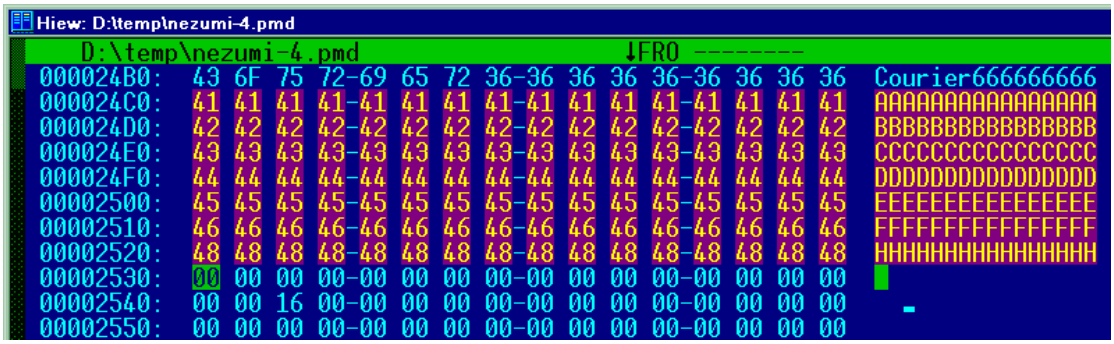


Рисунок 12 бегущая строка

Загружаем обновленный файл в PageMaker и… крэш мистическим образом исчезает. Как же это так?! Очень просто — по-видимому буквенная комбинация, перекрывшая адрес возврата, совершенно случайно указала на вменяемую область памяти! Рехнуться можно! Но такова наша хакерская жизнь. Сплошные сюрпризы. Ладно, меняем "AAA…BBB…CCC…" на "111…222…333…" (см. рис. 13), надеясь, что на этот раз нам повезет.

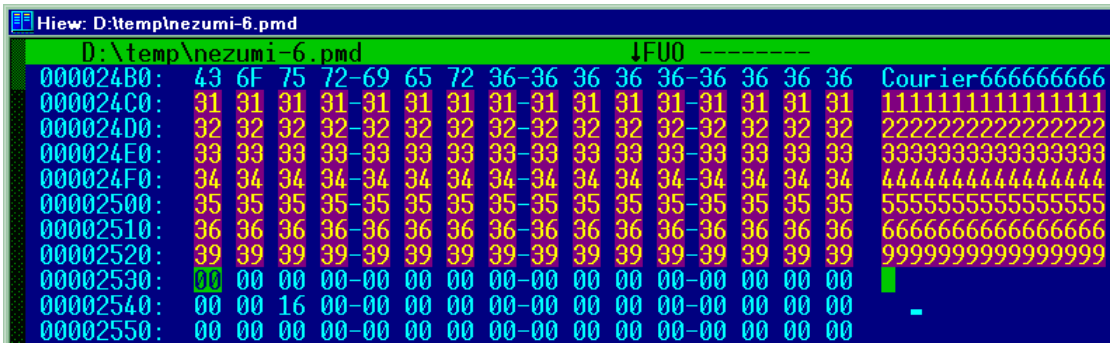


Рисунок 13 бегущая строка — вторая попытка

И точно! Теперь исключение выпрыгивает по адресу 32323232h, что соответствует последовательности "222…". Очень хорошо! Самое время для сужения сектора поиска. Меняем "222…" на "FF FE FF FD… FF F8 FF F7" (см. рис. 14). Естественно, эта последовательность может быть и другой. Никакого произвола тут нет. Главное, чтобы мы точно смогли определить локацию двойного слова, попадающего в адрес возврата.

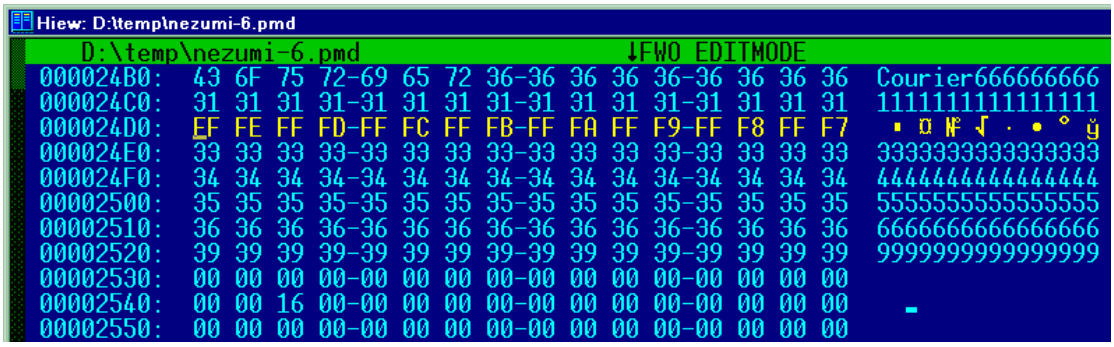


Рисунок 14 бегущая строка тонкой настройки

На этот раз исключение выпрыгивает по адресу F7FFF8FFh (см. рис. 15) из чего можно заключить, что в адрес возврата попадают последние четыре байта.

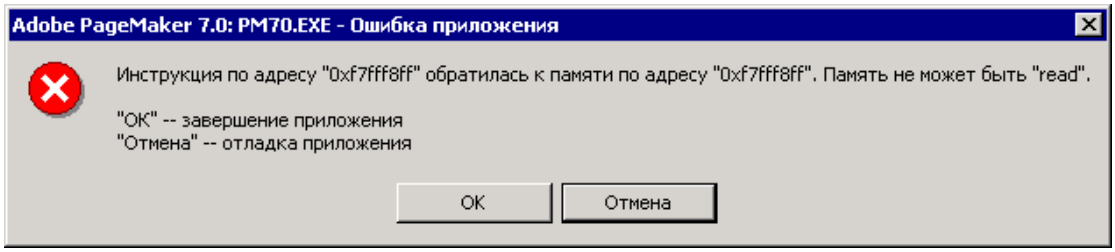


Рисунок 15 дислокация четырех байт, попадающих в адрес возврата

Нажимаем "отмену" для вызова отладчика и смотрим содержимое регистров и стека (см. рис. 16), а в стеке у нас содержится "444…555…". Приехали! В стек попадает буферная память, находящаяся _за_ подменяемым адресом возврата и если в адресе возврата окажется хотя бы один нулевой символ, он будет воспринят как завершитель строки и остаток переполняемого буфера в стек просто не попадет!

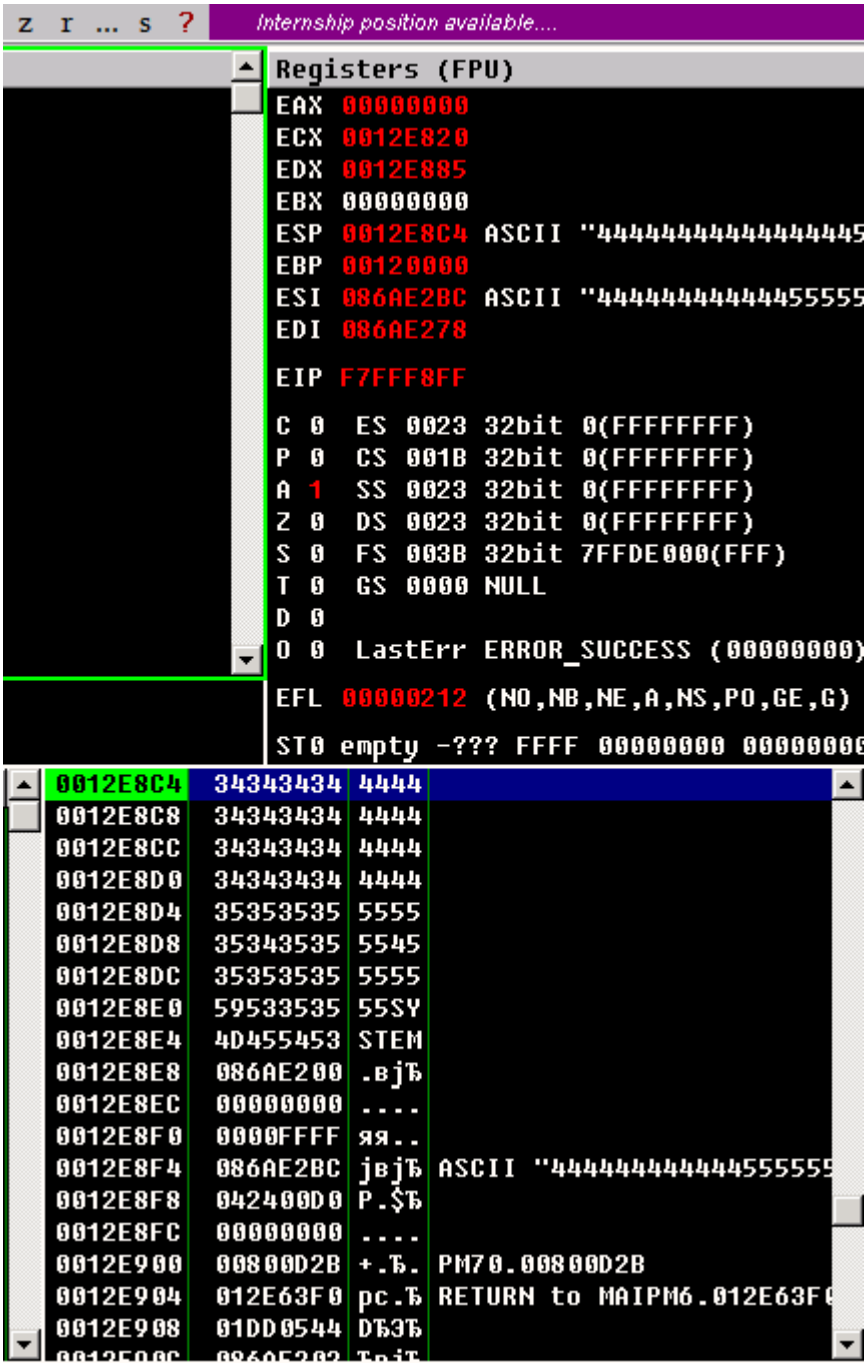


Рисунок 16 значение регистров на момент переполнения

Не попадет — и черт с ним! В конце концов, можно выбрать такой адрес возврата, чтобы без нулей. Например, передать управление на API-функцию TerminateProcess, чтобы аккуратно захлопнуть PageMaker при открытии файла без сохранения данных. Не велика подлость, но для начала сгодиться и это. Находясь в отладчике, давим <Alt-E> (Executable Modules), находим там KERNEL32.DLL, которая и экспортирует TerminateProcess (см. Platform SDK или кури Рихтера), и пишем "TerminateProcess" для быстрого поиска обозначенной функции, в W2K расположенной по адресу 77E8225Ch.

Открываем наш многострадальный файл в hiew'е и меняем FFh F8h FFh F7h на 5Ch 22h E8h 77h (адрес 77E8225Ch записанный задом наперед с учетом порядка следования байт в двойном слове). Сохраняем изменения по <F9>, выходим из hiew'а, грузим файл в PageMaker и… получаем неожиданный крэш по адресу 57E8225Ch. Чтобы мы сдохли! И не воскресли! Что это за ботва такая и откуда она вообще приплыла?! Смотрим — ага, совпадают все байты, только 77h магическим образом превратилось в 57h.

Курим. Медитируем. Потом опять курим и медитируем. И так вплоть до полного просветления. Hex-код 77h соответствует символу "w", а 57h – "W", то есть наш подопытный PageMaker перед копированием шрифтов переводит их имя в верхний регистр – ну чтобы искать легче было (как утверждает Platform SDK от MS, шрифты не чувствительны к регистру и потому, чтобы исключить неоднозначность, регистр должен быть один).

Таким образом, наша задача очень сильно осложняется — необходимо выбрать такой адрес возврата, чтобы в нем не было символов нижнего регистра. Но это ерунда. Самый главный вопрос — а где располагать shell-код?! Смотрим на значение остальных регистров. Так-так-так… Регистр ESI указывает на "444…555…", расположенную в куче. Отсюда возникает идея — поместить shell-код, начиная со строки "444…", а в качестве адреса возврата подсунуть адрес инструкции JMP ESI с опкодом FF E6, который легко отыскать в памяти процесса.

Какая по счету четверка попадает в ESI? На этот вопрос можно ответить сразу, просто подсчитав кол-во четверок, расположенных до первой пятерки. Их будет ровно 12. А мы вбивали — 16. Следовательно, ESI указывает на третью четверку, считая от нуля. Между прочим, именно по этому адресу располагалась та загадочная структура "1 :) :) :) :)…", назначение которой мы не выяснили до сих пор, но которая нас очень здорово выручает, поскольку копируется в кучу независимо от наличия нулевых символов в имени шрифта.

Короче, кончаем болтать и приступаем к поиску JMP ESI. Для этого в списке исполняемых модулей выбираем "PM70.EXE", нажимаем на <ENTER> для перехода в его начало и давим <Ctrl-B> для форсирования двоичного поиска последовательности "FFh F6h". Клавиша <Ctrl-L> – означает продолжить поиск. В общем, нам подойдет любой адрес, главное, чтобы он: а) не содержал символов нижней строки; б) находился в пределах модуля PM70.EXE. Последнее обстоятельство обеспечивает переносимость exploit'a между системами, поскольку, базовый адрес загрузки PM70.EXE всегда постоянный — где бы он ни был, а это значит, что постоянным будет и адрес команды JMP ESI, найденный внутри его.

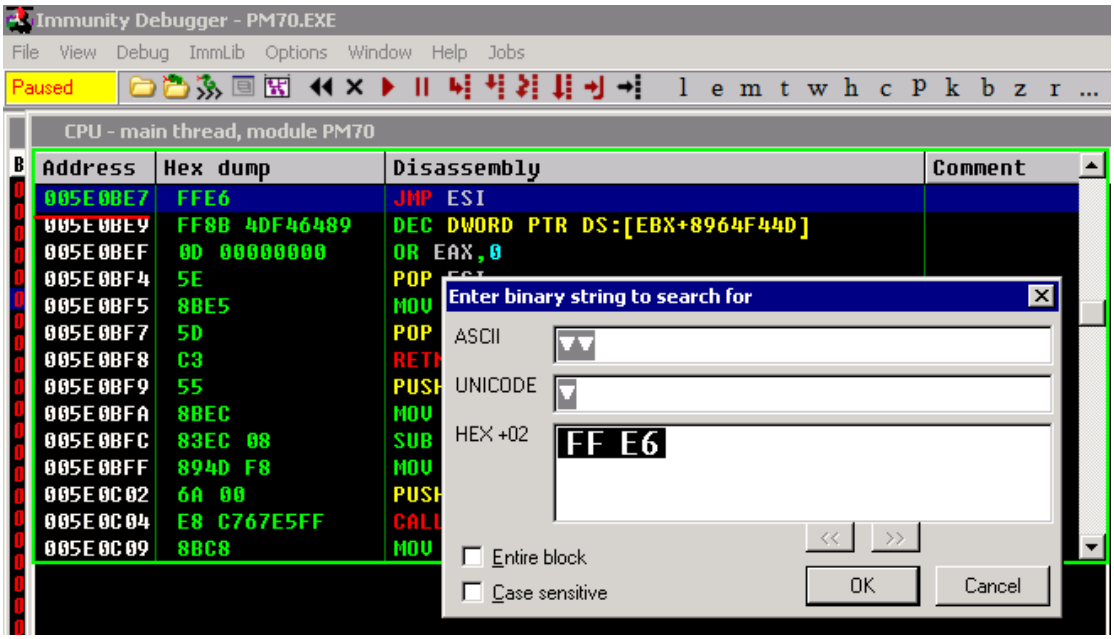


Рисунок 17 поиск JMP ESI

Вполне подходящий вариант расположен по адресу 005E0BE7h, который в символьном виде выглядит как: "ч♂^". И хотя "ч" находится в нижнем регистре, будем надеяться, что американский PageMaker с особенности русского хакера не знаком и вообще не в теме.

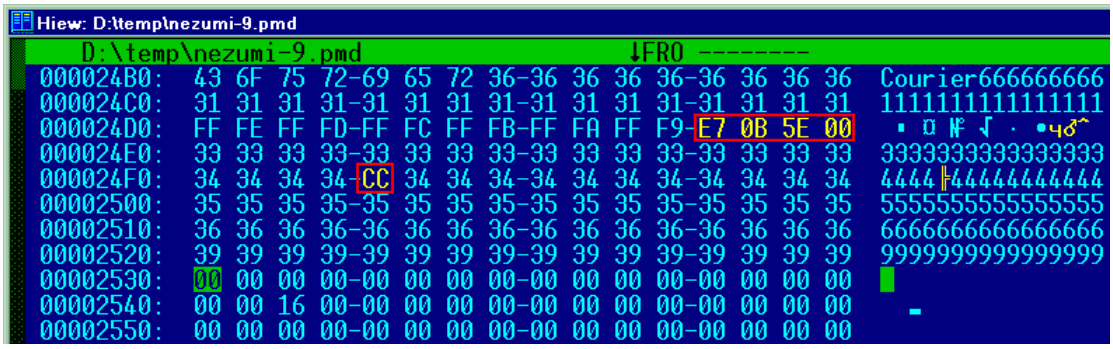


Рисунок 18 финальный вариант pmd-файла

Ну, понеслись! Заменяем FFh F8h FFh F7h на E7h 0Bh 5Eh 00h, а в третий (считая от нуля) байт последовательности "444…" внедряем наш shell-код, которым для простоты будет машинная инструкция INT 03h (опкод CCh), которая при выполнении должна вызывать исключение 80000003h, что послужит доказательством корректности работы exploit'a. В общем, финальный вариант должен выглядеть так (см. рис. 18).

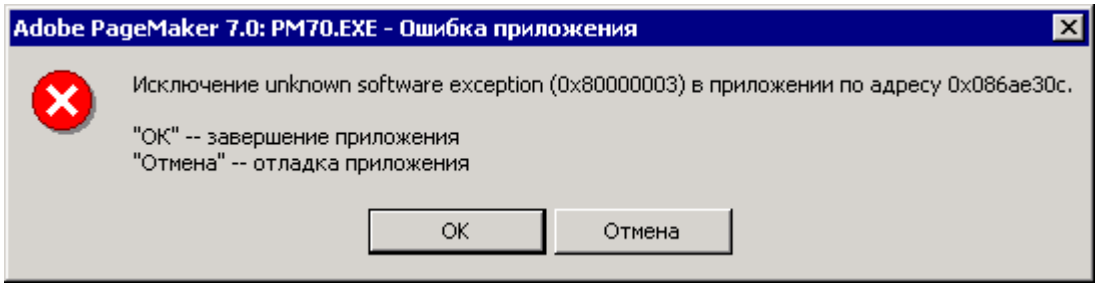


Рисунок 19 ура! это работает!

Загружаем сформированный файл (его можно взять с http://nezumi.org.ru/souriz/hack/nezumi-pagemaker-MAIPM6-bug.7z) в PageMaker и ловим исключение 80000003h, как и предполагалось ранее. Естественно, INT 03h – очень простой shell-код и его можно заменить любым другим боевым зарядом, взятым с http://www.milw0rm.com или написанным самостоятельно. В сети валяется куча shell-кодов устанавливающий back-door или делающих разные прикольные дела, остающиеся на совести их реализаторов. Но мыщъх'а это не интересует. Мыщъх подготовил публичный exploit, объяснив с какой позиции туда внедрять shell-код, чем и доволен.