# переполнение буфера на системах с неисполняемым стеком

крис касперски ака мыщъх

не ошибается то, что не работает.
(c) Windows

**отчаявшись справиться со своими же собственными ошибками, компания Microsoft совместно с Intel и AMD реализовала технологию DEP, призванную покончить с удаленными атаками раз и навсегда, но ни хрена она с ними не покончила и сейчас я расскажу почему… этого не произошло и защиту удалось обойти...**

## *введение*

Несмотря на все усилия, вырытые рвы и воздвигнутые защитные сооружения, интенсивность удаленных атак не снижается и отражать их становится все труднее. Хакеры научились маскировать вредоносные процессы под LINUX/BSD и NT, разобрались с брандмауэрами и освоились с распределенными системами — сотни тысяч зараженных машин, управляемых через IRC — это настоящая армия, разрушительная как монгольская орда. Интересы отдельных пользователей, пострадавших от атаки, отходят на задний план, уступая место вопросам безопасности всей инфраструктуры в целом. А инфраструктура — это уже серьезно.

Анализ показывает, что подавляющее большинство атак используют ошибки переполнения (buffer overflow), фундаментальную природу которых мы уже обсуждали (см. "ошибки переполнения буфера извне и изнутри как обобщенный опыт реальных атак"). Львиная доля таких ошибок находится в Internet Explorer'e, что в частности и явилось причиной перехода NASA на FireFox (http://www.securitylab.ru/news/242844.php?R1=RSS&R2=allnews), что можно только приветствовать, однако, FireFox (как и все остальные браузеры) не свободен от ошибок и в нем присутствует большое количество критических уязвимостей, позволяющих злоумышленнику выполнить свой код (далее по тексту называемый shell-кодом). Вот только одна из таких дыр: http://www.securitylab.ru/vulnerability/240254.php.

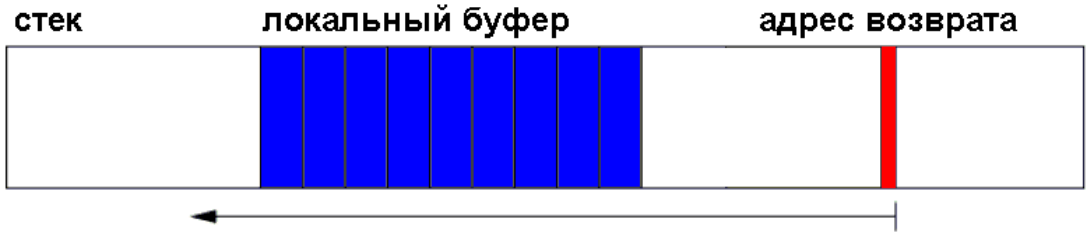


**Рисунок 1 схематичное изображение стека. стрелка показывает направление в котором растет стек. локальный буфер "растет" в противоположном направлении и при определенных обстоятельствах может затирать адрес возврата**

Никакое программное обеспечение небезопасно! Даже если своевременно устанавливать свежие заплатки всегда существует риск, что хакер найдет новую дыру о которой еще никто не знает и с успехом воспользуется ею. Кстати говоря, разработка атакующих программ за последние несколько лет поднялась с колен чистого энтузиазма и встала на коммерческих поток с большими вложениями. Стоимость гарантированно работающего эксплоита зачастую доходит до $2.000. Можно только догадываться кому и зачем это нужно…

В очередной раз возникло понимание, что "дальше так жить нельзя" (c) и нужно что-то решать. Попытки покончить с удаленными атаки неоднократно предпринимались еще с 80'х годов, особенно после Червя Морриса, но все безуспешно. Как говорят в этих случаях американцы: серебренной пули нет! Оборотня не убить! Тем не менее, Microsoft решилась отчаянный шаг и возродила старые идеи, отрытые на помойке истории и реализованные без учета понимания ситуации наших дней.

Технология **DEP** (***D**ata **E**xecution **P**rotection* — Защита от Выполнения Данных), реализованная в Windows XP SP2 и Server 2003 SP1, делает секцию данных, стек и кучу неисполняемыми, что (теоретически) предотвращает засылку shell-кода и отсекает целый класс удаленных атак, основанных на переполнении.

В чем же революционность такого решения? Ведь еще со времен Windows 95 (не говоря уже про стандарт POSIX, принятый в 1985 году) x-атрибут (от e**X**ecutable – исполняемый) имеется только у кодовой секции и отсутствует у всех остальных. Достаточно взять любой ELF/PE-файл и посмотреть! Так-то оно так, но… у x86 процессоров на этот счет имеется свое собственное мнение: на уровне страниц процессор поддерживает всего лишь два атрибута защиты: -**a**- (от **a**ccessed – страница доступна для чтения/исполнения) и -**w**- (от **w**rite – запись разрешена). Никакого x-атрибута в PTE нет и он присутствует только в таблице селекторов, а это значит, что мы не можем выборочно разрешать/запрещать исполнение кода для отдельных страниц, а только для всего сегмента целиком. Вот потому, Windows с ее плоской моделью памяти, вынуждена трактовать атрибут -r- как -x-, а -x- как -r-. То есть, право на чтение страницы дает неявное право на ее исполнение и наоборот.

На самом деле, никакого произвола со стороны процессора здесь нет: проблему можно решить и в рамках плоской модели. Достаточно "всего лишь" перегруппировав сегменты и установить правильные лимиты (см. рис. 2). Естественно, это требует больших телодвижений со стороны разработчиков ядра (необходимо использовать раздельные ITLB/DTLB и т. д.), что их совсем не вдохновят.

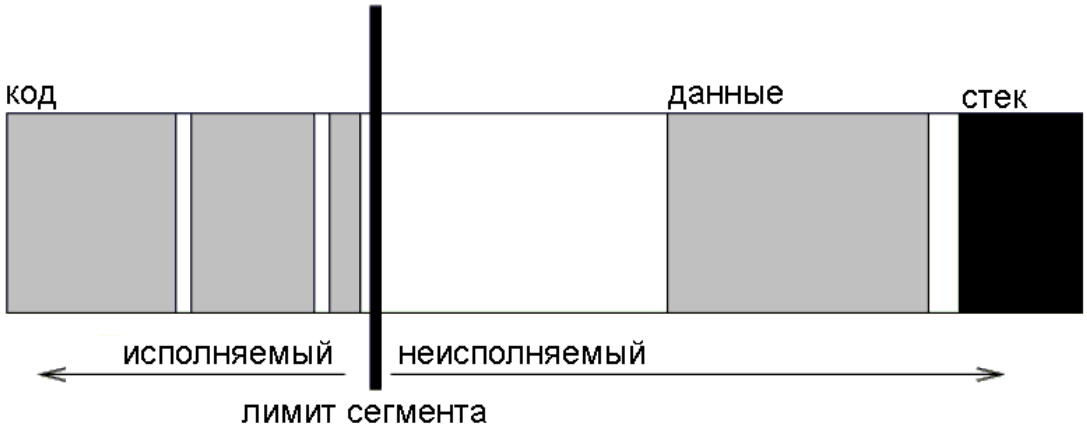


**Рисунок 2 "эмуляция" NX/XD битов на x86 процессорах**

Почему же в x86 не была предусмотрена возможность задания x-атрибута на уровне страниц? Риторический вопрос… Скорее всего, во времена проектирования 80386 это было никому не нужно, вот и решили не усложнять логику процессора без необходимости. А вот в Itanium'e этот атрибут присутствует изначально и специальный бит, известный под аббревиатурой XD (от eXecute Disable – выполнение запрещено), определяет: разрешено ли выполнение кода в данной странице или нет.

Аналогичный бит имеется и в процессорах AMD-64 (Opteron и Athlon-64), только там он называется NX (от No-eXecute — не выполняемый). Кому-то в компании пришла в голову "здравая" мысль объявить x-атрибут "технологией" и вокруг NX-бита тут же развернулась рекламно-маркетинговая акция "Enhanced Virus Protection" (Расширенная Вирусная Защита) или сокращенно EVP. На стайте компании выложено множество красочных роликов, демонстрирующих как AMD борется с вирусами на уровне процессора (!). Неудивительно, что 64-битная редакция NT от рождения имеет неисполняемую кучу и стек! Microsoft просто подхватила брошенный ей атрибут защиты и встроила его в систему, чтобы лишний раз продемонстрировать, что она не отстает от прогресса. В этом-то и заключается сущность аппаратного (hardware-enforced) DEP.

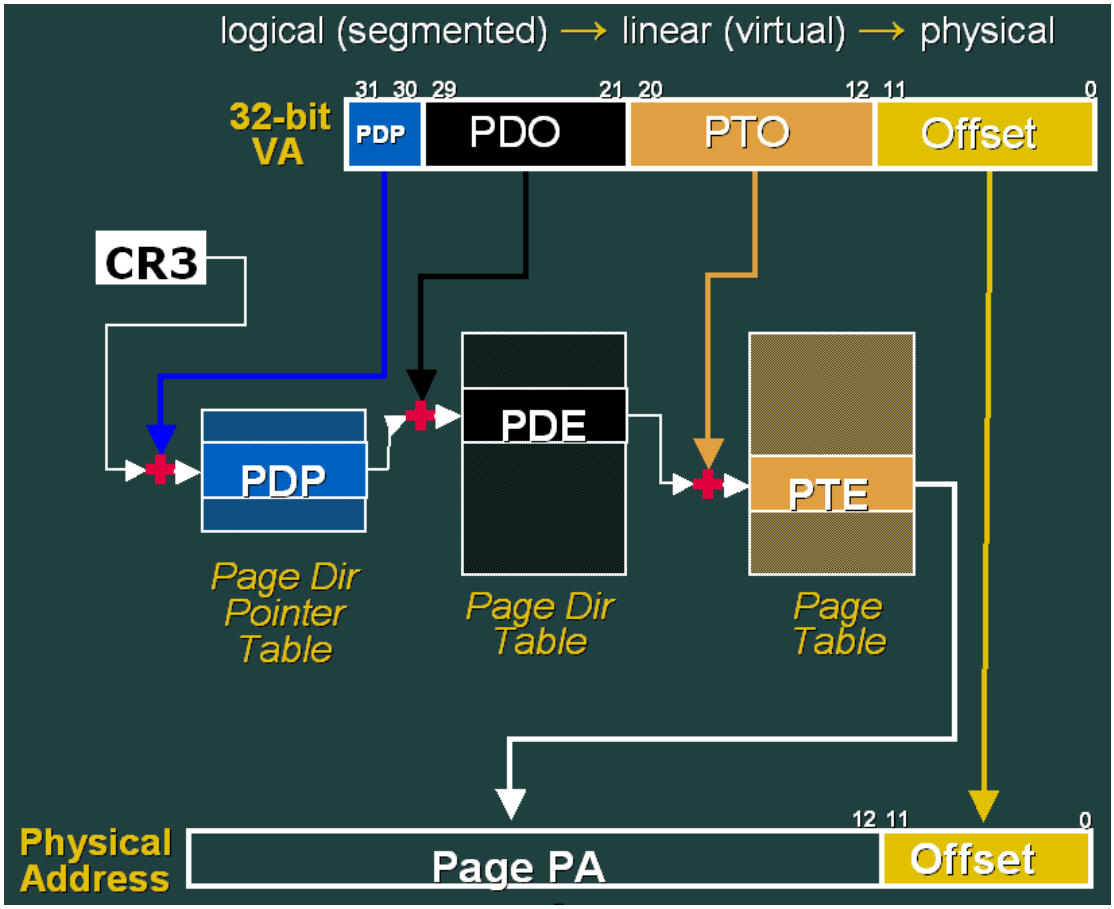


**Рисунок 3 организация виртуальной памяти**

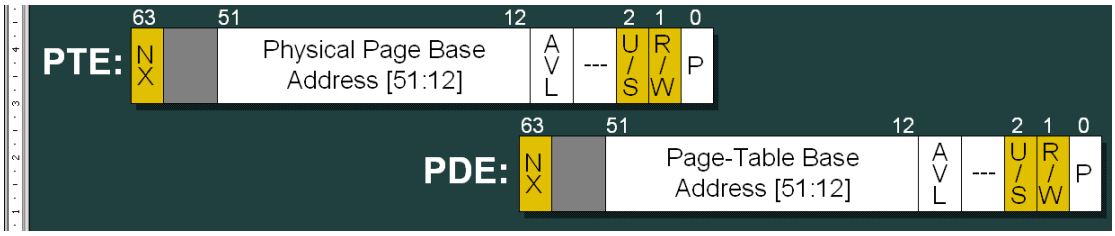


**Рисунок 4 NX бит — новый атрибут защиты в PDE/PTE**

Воспользовавшись шумихой, развернутой AMD, компания Intel внедрила поддержку XD-бита в 32-разрядные Pentium'ы, сделав эту "технологию" доступной всем и каждому (желающих пересесть на AMD-64 ради одно лишь DEP'а набралось не так уж и много). Строго говоря, Nx присутствует не только в "настоящих" 64-битных процессорах от AMD, но и в собранных на 64-битном ядре 32-битных (т. е. работающих в состоянии перманентной эмуляции i386) процессорах – например Sempron (около 99 долларов за штуку). Достаточно лишь воткнуть последний Pentium-4 и обновить ядро, чтобы Windows могла задействовать новые аппаратные возможности, и тогда при попытке выполнения прикладного кода в куче, секции данных или на стеке возбуждается исключение типа STATUS_ACCESS_VIOLATION (C0000005h) и, если только программист не установил свой обработчик SEH, выполнение программы аварийно завершается с выдачей сообщения "to help protect your computer, Windows has closed this program" (см. рис. 5).


**Рисунок 5 реакция DEP на попытку выполнения кода на стеке**

В режиме ядра при попытке выполнения кода в неисполняемой области памяти возбуждается исключение ATTEMPTED_EXECUTE_OF_NOEXECUTE_MEMORY с BugCheck-кодом FCh, обрушивающим систему в "синий экран". 64-битные версии NT защищают от исполнения стек ядра, paged и session pool, а 32-битные — только стек.

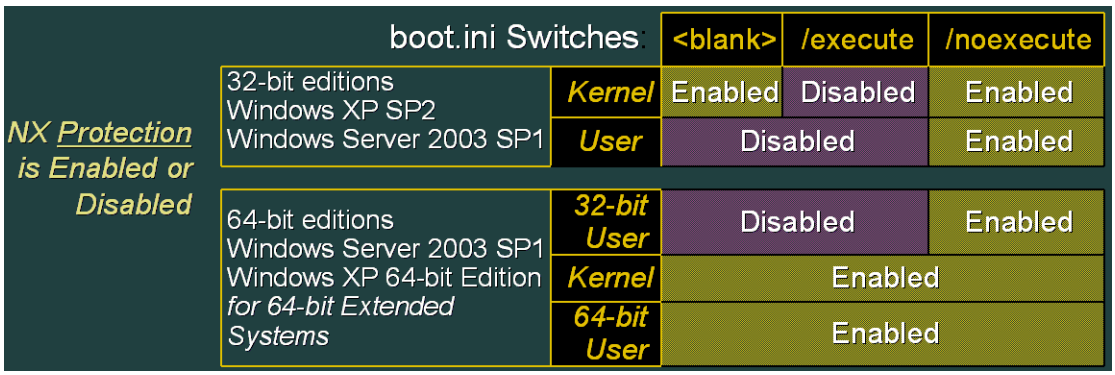


**Рисунок 6 регионы памяти, защищенные от выполнения кода**

Если процессор не поддерживает NX/XD-битов, система активирует программный (software-enforced) DEP, который даже и не пытается эмулировать неисполняемый стек/кучу (а ведь мог бы!). Технология, которую продвигает Microsoft, в действительности представляет собой примитивную защиту обработчика структурных исключений, ранее известную под именем SafeSEH. Она никак не препятствует выполнению shell-кода, но затрудняет использование структурных исключений shell-кодом, если тот вдруг решит их использовать. Подробнее об этом можно прочитать в статье "SEH на службе контрреволюции", электронная копия которой лежит на сервере ftp://nezumi.org.ru.

Мы будем обсуждать только аппаратный DEP, поскольку его сложнее всего обойти. Некоторые даже считают, что это вообще невозможно: "*...на сегодняшний день не существует ни одного хотя бы концептуального эксплойта, на практике доказывающего возможность поражения Windows XP с установленным пакетом обновления Service Pack 2*" (www.computerra.ru/softerra/36652). Если бы DEP проектировался головой, все так бы и было. Однако, Microsoft идет своим особым путем, который умом не понять, и DEP обходится без труда, а вот у легальных пользователей возникают множество проблемы, о которых мы еще поговорим.

Неправильно называть DEP защитным механизмов. До "защитного механизма" ему так же далеко как одиноко торчащему рулю до самосвала. DEP – это всего лишь поддержка атрибутов защиты страниц и ничего более!

## >>> врезка кто поддерживает XD-бит

По данным компании Intel, XD-бит поддерживают следующие операционные системы, предотвращая ***непреднамеренное*** исполнение кода в неисполняемых областях памяти (однако, преднамеренных хакеров эта мера не остановит и все системы могут быть атакованы, Windows – проще, Red Hat – сложнее):

- Microsoft Windows* Server 2003 with Service Pack 1
- Microsoft Windows* XP* with Service Pack 2
- SUSE Linux* 9.2
- Red Hat Enterprise Linux 3 Update 3

## конфигурирование DEP

64-битные редакции NT при работе в native-режиме всегда задействуют DEP и не позволяют его отключать. Если разработчику хочется выполнить код на стеке или в куче (а хочется это достаточно часто), он должен ***явно*** назначить атрибуты доступа к данному региону памяти путем вызова API-функции VirtualAlloc или VirtualProtect. Никаких прав для этого не нужно, так что мы получаем лишь видимость безопасности – защиту от непреднамеренного доступа, но не более того и чуть позже мы покажем как ее обойти.

С 32-битными приложениями все намного сложнее. Существует огромное количество уже написанного ПО, трактующего атрибут -r- как -x-, и отказывающегося работать, если это не так. Поэтому, для сохранения обратной совместимости Microsoft предусмотрела возможность отключения DEP для 32-битных редакций NT и 32-битных приложений, исполняющихся в 64-битных редакциях NT.

Чтобы задействовать механизм DEP, необходимо иметь процессор, поддерживающий NX/XD-биты, причем 32-битные процессоры поддерживают NX-бит только в режиме PAE (Physical Address Extension – режим расширение физических адресов). 32-битные редакции NT автоматически распознает тип процессора, при необходимости добавляя ключ /PAE в файл boot.ini. 64-битные редакции не имеют отдельного PAE-ядра (они работают через технологию расширения окна адресации Address Windowing Extension — AWE), поэтому добавлять ключ /PAE для них не нужно.

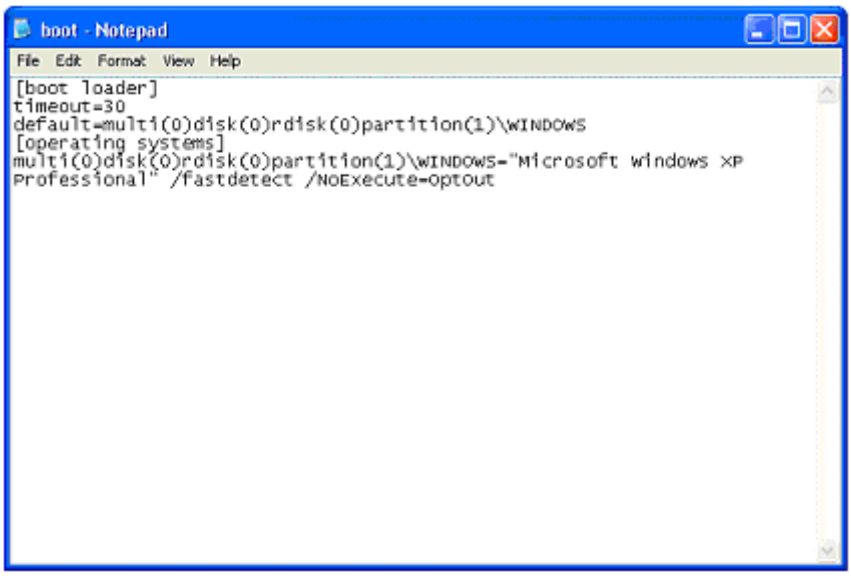


**Рисунок 7 конфигурирование DEP через редактирование boot.ini файла в Notepad**

Если не указывать никаких дополнительных ключей, то в 32-битных редакциях NT будет запрещено выполнение кода на стеке, в ядре и в некоторых системных службах прикладного уровня. Все остальные приложения будут исполняться в обычном режиме. 64-битные версии по умолчанию разрешают выполнение кода на стеке/куче только для 32-битных приложений, а для 64-битных они запрещены (см. рис. 6)

Ключ **/execute** полностью отключает DEP для 32-битных редакций NT и 32-битных приложений, исполняющихся под 64-битными редакциями NT, но на "родные" 64-битные приложения его влияние не распространяется и они по-прежнему остаются защищены.

Ключ **/noexecute=*уровень_политики*** позволяет конфигурировать DEP по своему усмотрению, выбирая требуемый уровень защищенности. Параметр "уровень_политики" может принимать одно из следующих значений: *AlwaysOn*, *AlwaysOff*, *OptIn и OptOut*, описанных в таблице 1.

| параметр | значение |
|---|---|
| **OptIn**(по умолчанию) | DEP включена только для ограниченного системных процессов Windows и ядра |
| **OptOut** | DEP включена для всех процессоров и ядра, однако, можно сформировать список приложений, на которые защита не распространяется |
| AlwaysOn | DEP включена для всех процессоров и ядра. Отключить защиту для выборочных приложений нельзя |
| AlwaysOff | DEP отключена для всех процессов и ядра |

**Таблица 1 влияние уровня политики на безопасность**

Если редактирование boot.ini приводит вас ужас (что в общем-то вполне закономерно, поскольку после этого система может перестать загружаться), воспользуйтесь интерактивным конфигуратором:

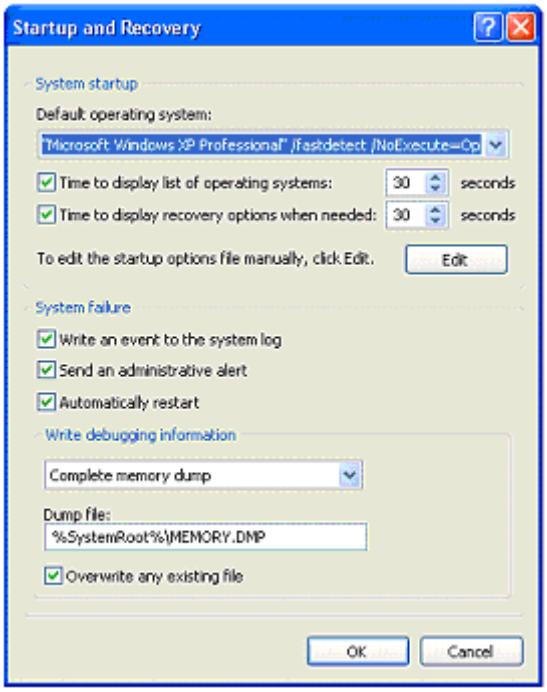


**Рисунок 8 редактирование boot.ini файла через интерактивный конфигуратор**

- в командной строке введите "start sysdm.cpl" или нажмите <win>-<break>;
- на вкладке "дополнительно" в группе "быстродействие" нажмите кнопку "параметры";
- перейдите на вкладку "предотвращение выполнения данных";
- выберите пункт "включить DEP только для основных программ и служб Windows" чтобы добавить в boot.ini параметр OptIn или выберите "включить DEP для всех программ и служб, кроме выбранных ниже", чтобы добавить параметр OptOut. другие параметры через интерактивный конфигуратор добавить невозможно;
- если был выбран параметр OptOut нажмите кнопку "добавить" и укажите программы, на которые DEP распространятся не должна;

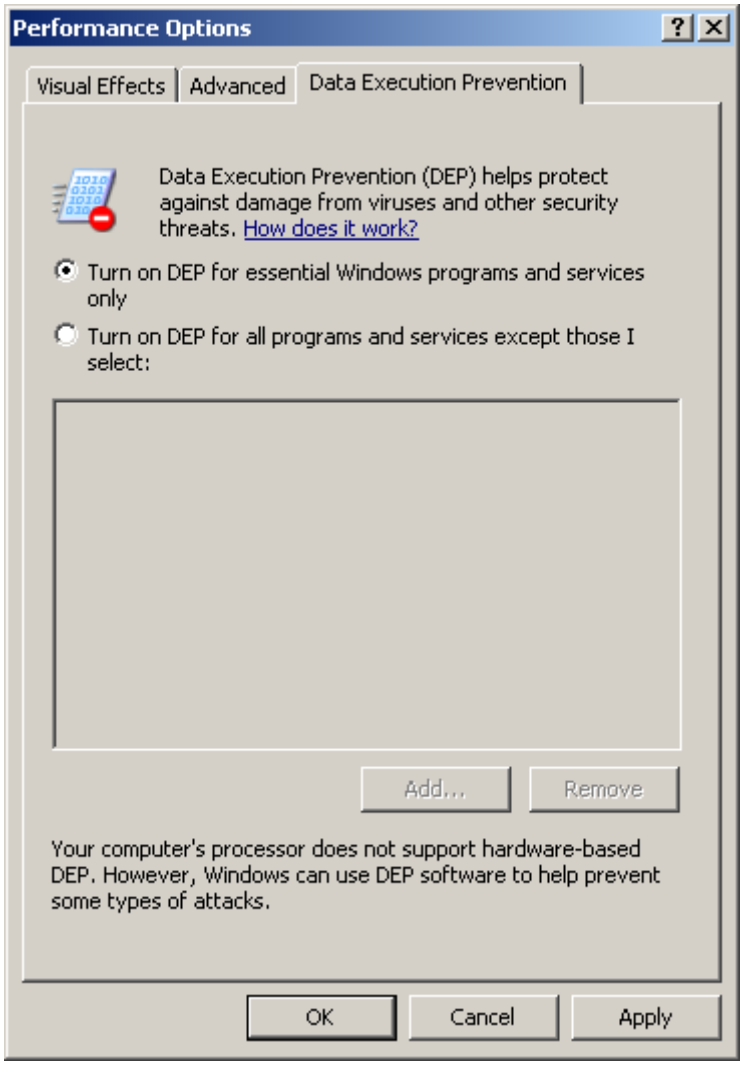


**Рисунок 9 интерактивное конфигурирование DEP**

Список программ, для которых отключен DEP, можно сформировать и через реестр (чисто хакерский путь, экономящий массу времени). Просто откройте ключ HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers и создайте новый ключ типа "строка". Его имя должно содержать полный путь к exe-файлу, защиту которого мы хотим отключить, поместив внутрь ключа значение "DisableNXShowUI". Только не перепутайте их местами — иначе ничего не получится.

## *проблемы совместимости*

Исполняемый стек необходим очень многим приложениям: защитным механизмам, эмуляторам, just-in-time компиляторам и т. д. Попытки сделать стек неисполняемым неоднократно предпринимались еще в 70-80 годах, когда никакого Windows'a и в проекте не существовало, но… они как-то не прижились. А все из-за проблем совместимости. Кому нужна операционная система, если на ней нельзя запускать свои любимые приложения? Выхода нет. Или безопасность, или совместимость. Компромисс невозможен. Разве только что… рассказывают (врут кончено), что давным-давно стали греки строить баню. И озадачились они вопросом: стоит ли строгать доски или нет? Ведь если не построгать, можно легко занозиться, а если построгать… попробуй тут не поскользнуться мыльной пяткой. Поразмыслив и курнув хорошей травы, они поступили так: построгали и прибили обстроганным на низ. Аналогичным путем пошла и Microsoft: разработала некоторую пародию на защиту и тут же ее отключила, чтобы у пользователей не возникло лишних проблем.

Тем не менее, проблемы все-таки возникли. Наберите "DEP" в "Базе Знаний", чтобы посмотреть какие конфликты обнаруживаются только с продуктами самой Microsoft, которая по

всем понятиям, должна была обеспечить надлежащую преемственность и совместимость. Вот, например:

- **Hardware (DEP)-enabled computer may unexpectedly quit after you resume from standby or from hibernation in Windows XP Service Pack 2** (при активном аппаратном DEP компьютер может неожиданно зависать при выходе из спящего или hibernation-режима): http://support.microsoft.com/default.aspx?scid=kb;en-us;889673;
- **You receive a "Data Execution Prevention" error message when you start Live Meeting 2005** (при запуске Live Meeting 2005 появляется сообщение об ошибке типа "Data Execution Prevention") http://support.microsoft.com/default.aspx?scid=kb;en-us;894643;
- **You receive error messages when you install Windows CE 4.x Emulator on a computer that is running Windows XP Service Pack 2 and the computer has DEP hardware** (при установке эмулятора Windows CE 4.x, возникает сообщение об ошибке): http://support.microsoft.com/default.aspx?scid=kb;en-us;891667;

Со сторонними производителями дело и вовсе труба. Навесные упаковщики и протекторы, использующие самомодифицирующийся код и другие антихакерские трюки, не могут работать с неисполняемым стеком, а это значит, что защита DEP для них должна быть отключена. Кое-кто может возразить, что все проблемы от "неправильного" стиля программирования и использования "недокументированных" особенностей системы (которые на самом деле очень даже документированные), однако, проблемы встречаются не только у кустарей, но и весьма именитых фирм, например, Borland. Заходим в "Базу Знаний", набираем "DEP" и тут же получаем множество ссылок, типа:

- http://qc.borland.com/wc/qcmain.aspx?d=21249
- http://qc.borland.com/wc/qcmain.aspx?d=10827
- http://support.borland.com/entry.jspa?externalID=1522&categoryID=108

Таким образом, ***DEP очень конфликтная вещь, создающая множество проблем и ни от чего не защищающая***. Microsoft допустила множество серьезных просчетов, позволяющих проникать сквозь DEP даже на максимальном уровне защищенности.

Все дальнейшие рассуждения применимы как к 32- так и к 64-битным редакциям NT, и не зависят от настойки системы. Будем для определенности считать, что ключ /noexecute установлен в положение AlwaysOn.

## *атака на DEP*

Microsoft подтвердила возможность обхода DEP еще в январе 2005, когда на maxpatrol'e появилась статья Александра Анисимова "Defeating Microsoft Windows XP SP2 Heap protection and DEP bypass", однако, не придала этому большего значения, заявив, что реализовать удаленную атаку все равно не удастся: "An attacker cannot use this method by itself to attempt to run malicious code on a user's system. There is no attack that utilizes this, and customers are not at risk from the situation" (*Атакующий не может использовать этот метод для запуска зловредного кода на целевой системе. До сих пор не было продемонстрировано ни одной таки, использующей этот трюк и потребители находятся вне риска* — http://www.eweek.com/article2/0,1759,1757786,00.asp).

Это — наглая ложь! Механизм DEP легко пробивается с любого расстояния, хоть из Антарктики. Результатом атаки становится переданный и успешно выполненный shell-код, выполняющий команды заранее подготовленные злоумышленником. Чтобы понять как это делается, сначала необходимо разобраться с классическими методами переполнения, подробно разобранных в статье "ошибки переполнения буфера извне и изнутри как обобщенный опыт реальных атак".

Напомним читателю основные положения: отсутствие контроля границ локальных буферов позволяет затирать адрес возврата из функции, помещая сюда указатель на shell-код, который находится здесь же, в стеке. Другой тип переполнения связан с кучей. С его помощью хакер может модифицировать любую writable ячейку в адресном пространстве уязвимого процесса (например, подменить указатель на виртуальную функцию или подделать адрес возврата). Имеются и другие возможности (в частности, атакующий может изменить обычный порядок выделения блоков памяти из кучи, разместив следующий выделяемый блок поверх ключевых структур данных), но для простоты изложения мы ограничимся модификацией адреса возврата.

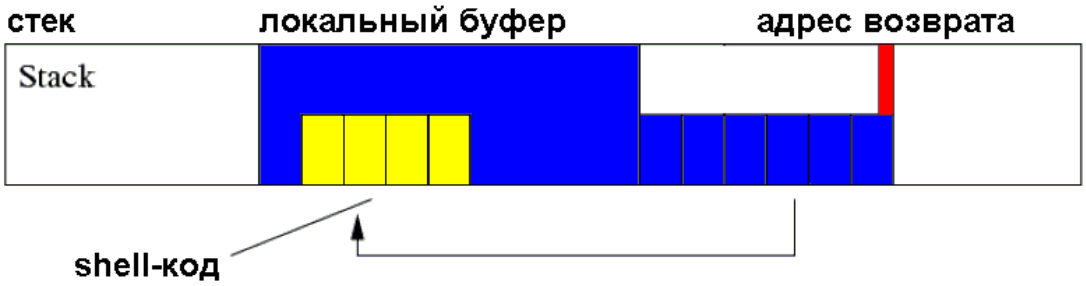


**Рисунок 10 классическая удаленная атака — засылка shell-кода в стек с последующей передачей на него управления путем модификации адреса возврата.**

Существует множество защитных механизмов, контролирующих целость кучи и адреса возврата, но… со своей задачей они не справляются. Это отдельная большая тема, никак не связанная ни с NX/XD-битами, ни с технологией DEP. В следующих статьях мы подробно рассмотрим ее, но… это будет потом. Сейчас же ограничимся тем, что DEP никак не препятствует модификации адреса возврата и существует множество программ действительно позволяющих это делать (Internet Explorer, FireFox и др.).

Манипулируя адресом возврата, хакер может вызывать произвольные функции уязвимой программы (в том числе и API-функции операционной системы) передавая необходимые параметры через стек. Конечно, при этом он будет очень ограничен в своих возможностях, поскольку среди готовых функций полезных для хакерствования не так уж и много, однако, если покурить хорошей травы и как следует подумать головой, попутно напрягая и другие части тела, проблему удается решить путем вызова API-функции CreateProcess или функции System из библиотеки CRT, запустив штатную программу типа tftp.exe и закачав на атакуемый компьютер двоичный файл, который можно исполнить с помощью CreateProcess/System (атака типа return-to-libc). Механизм DEP этому сценарию ***никак*** не препятствует, поскольку в этом случае shell-код передается не через стек/кучу, а через легальный исполняемый файл. На стеке содержаться лишь аргументы вызываемых функций и "поддельные" адреса возврата, указывающие на них.

Таким образом, ***даже при включенном DEP у хакера сохраняется возможность забрасывать ~~передать~~ на атакуемую машину свой код и передавать на него управление***. Наличие tftp.exe не является необходимым условием для атаки. Даже если его удалить, хакер может вызвать cmd.exe и… нет, удалять все файлы с диска необязательно (хотя и возможно), достаточно перенаправить вывод в файл и с помощью ECHO создать крохотный com, делающий что-то "полезное". Да и в самой уязвимой программе наверняка содержаться функции, через которые можно загрузить файл из Интернета… Словом, возможностей — море! Атаки этого типа хорошо изучены хакерами и описаны в литературе. Только специалисты из Microsoft похоже об этом ничего не знают (они не в курсе — они в танке), иначе как можно объяснить тот загадочный факт, что успешность атаки данного типа никак не зависит от активности DEP и старые эксплоиты полностью сохраняют свою работоспособность. Распространение червей останавливается только потому, что вместе с DEP пакет обновлений включает в себя заплатки на все известные дыры. Однако, стоит хакерам найти еще одну ошибку переполнения (а это всего лишь вопрос времени), как Интернет захлестнет новая эпидемия и никакой DEP ее не остановит. (Вообще-то, нет, не захлестнет, ведь автоматическое обновление теперь включено по умолчанию).

Некоторые могут сказать, что это "не настоящая" атака, поскольку в ней отсутствует явная передача shell-кода через стек, а ведь именно на это DEP и ~~нацелен~~рассчитан! Мысль, конечно, умная, но забавная. И какой это хакер броситься на амбразуру если можно сходить в обход? Неужели в Microsoft всерьез считают, что атакующий играет по "правилам" и идет по пути наибольшего сопротивления, карабкаясь по извилистой горной тропинке, напичканной патрулями, когда столбовая дорога никем не охраняется?!

В качестве разминки для мозгов рассмотрим альтернативный сценарий атаки, передающий shell-код через стек. Засунуть shell-код в локальный буфер — не проблема, подменить адрес возврата тоже, но… при попытке передачи управления на shell-код при активном DEP будет возникать исключение, ведь x-атрибута у нас нет, хотя… это смотря, что мы переполняем. Как уже говорилось выше, некоторые приложения (в том числе и браузеры) нуждаются в исполняемом стеке, в который они складывают откомпилированный java-код. Но не будем смягчать себе условия. Давайте считать, что никаких исполняемых страниц в нашем

стеке нет. Чтобы их заполучить, необходимо либо сбросить NX/XD-бит (но это можно делать только система), либо… вызывать функцию VirtualProtect, назначая атрибуты защиты по своему желанию! Но как же мы вызовем VirtualProtect без возможности выполнения shell-кода? Да очень просто — скорректируем адрес возврата так, чтобы он указывал на VirtualProtect, тогда при выполнении команды return она передаст на нее управление!

На самом деле, это только идея. До практической реализации ей еще далеко. В жизни все ~~происходит~~ намного сложнее и… элегантнее. Допустим, вызвали мы VirtualProtect. А дальше что? Куда она возвратит управление? И как узнать адрес выделенного блока? Это великолепная головоломка на решение которой мыщъх потратил целый день и был очень разочарован, когда узнал, что он не один такой умный, и все загадки ~~уже~~ разгадали еще до него. Взгляните на рис. 11. Специалист~~ам~~ы поймут идею с первого взгляда, не~~-~~специалистам мы сейчас все объясним.

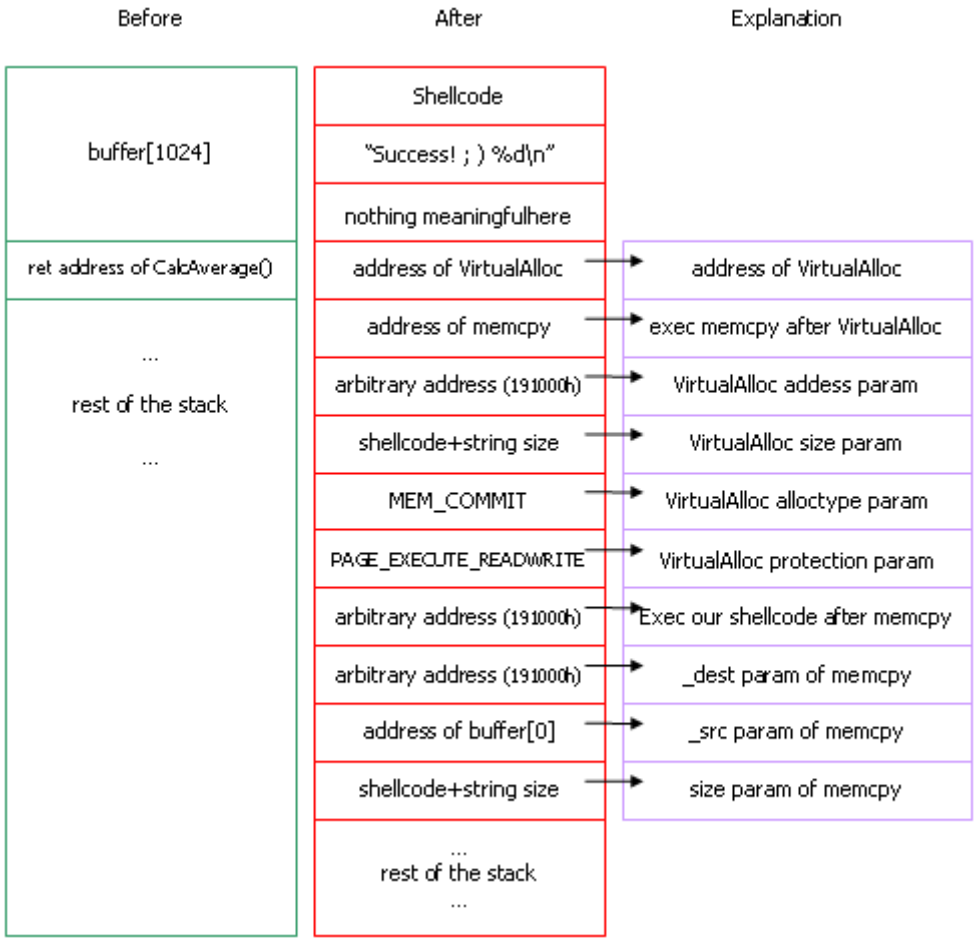


**Рисунок 11 подготовка стека для реализации атаки типа commit-n-copy**

Итак, все по порядку. Начнем с ответа на вопрос: куда возвращает управление VirtualProtect? Ответ очевиден: по указателю, который лежит за модифицированным адресом возврата! В момент выхода из VirtualProtect, процессор стакливает текущий адрес возврата с вершины стека и удаляет переданные ей аргументы. Так происходит потому, что VirtualProtect (как и все API-функции) придерживается соглашения о передаче параметров типа stdcall, при котором аргументы удаляются самой вызываемой функцией.

Таким образом, мы можем вызывать столько stdcall-функций, сколько захотим. Процессор будет послушно стягивать их со стека, поступательно двигаясь от вершины вглубь. Техника вызова cdecl-функций выглядит чуть сложнее. Они не очищают аргументы при выходе, и это атакующему приходится это делать самостоятельно. Проще всего перенаправить адрес возврата на код типа `ADD ESP,n/RET`, расположенный где-то внутри уязвимой программы, где n – количество байт, занятых аргументами. Такую комбинацию можно встретить практически в любом оптимизированном эпилоге. Ну а нужное n подобрать совсем не сложно!

Теперь мы знаем как вызывать функцию для изменения атрибутов доступа и вернуться обратно в shell-код, но это еще не все. Функция VirtualProtect требует, чтобы ей передали адрес уже выделенного региона, а shell-коду он неизвестен. К тому же, по слухам Microsoft собирается встроить в VirtualProtect дополнительную проверку, запрещающую назначать x-атрибут, если он не был присвоен еще при выделении. Тупик? Не совсем, ведь мы можем выделить новый регион, назначить нужные права, скопировать туда свой shell-код и передать ему управление. С выделением никаких проблем нет — берем VirtualAlloc и вперед. Как прочитать возвращенный указатель — вот в чем вопрос! Функция передает его в регистре EAX, и заранее предсказать его значение невозможно. А ведь хакер должен сформировать указатель и затолкать его в стек еще на стадии проектирования shell-кода, то есть задолго до вызова VirtualAlloc. Ну теперь уж точно тупик… А вот и нет! Внимательное чтение SDK (или Рихтера) показывает, что Windows позволяет COMMIT'ить (то есть передавать) уже переданную память по заданному адресу и хотя последствия такого выделения могут быть очень печальными (для кучи) — кого это волнует?! Главное, чтобы shell-код получил управление и он его получит! Оторвать мыщъх'у хвост, если это не так! Выбираем произвольный адрес, который с высокой степенью вероятности не ~~будет~~ занят ничем полезным (например, 191000h), и передаем его функции VirtualAlloc вместе с флагом MEM_COMMIT и атрибутами PAGE_EXECUTE_READWRITE. Все! Первая стадия атаки благополучно завершилась и теперь мо~~ж~~жно курнуть травы или глотнуть пива.

На втором шаге мы вызываем функцию memcpy и копируем shell-код в только что выделенный регион памяти, целевой адрес которого заранее известен. После этой операции, shell-код оказывается в области памяти, где разрешено выполнение и нам остается только заснуть в стек еще один подложный адрес возврата, который будет указывать на него (**внимание**! ***функция memcpy в отличии от VirtualAlloc придерживается cdecl соглашения, поэтому после нее мы уже не можем выполнять никакие другие функции, предварительно не удалив аргументы со стека***).

Последовательность вызовов, реализующих атаку выглядит так (напоминаем: это не shell-код, это именно *последовательность вызовов API-функций*, осуществляемая путем подмены адресов возврата):

```
VirtualAlloc(REMOTE_BASE, SHELLCODE_LENGTH, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
memcpy(REMOTE_BASE, SHELLCODE_BASE, SHELLCODE_LENGTH);
GOTO shell_code;
```

**Листинг 1 последовательность вызова функций, реализующих атаку типа commit-n-copy**

Маленький нюанс — 64-битные редакции NT передают аргументы API-функциям через регистры и потому вызывать VirtualAlloc на них уже не удастся (разумеется, речь идет только о 64-битных приложениях). То есть, вызывать-то удастся, а вот передать аргументы — нет, поэтому этот сценарий уже не сработает, однако, вызов функций уязвимой программы через подмену адреса возврата будет действовать по-прежнему (то есть, запустить tftp.exe мы все-таки сможем, вызывая ее через функцию System, которая по-прежнему принимает аргументы через стек).

Подведем итог: стечение ряда неблагоприятных для DEP обстоятельств делает эту технологию практически полностью бесполезной. Да, она отсекает целый класс атак, основанных на переполнении, однако, дает пишу для новых и в целом ситуация никак не меняется. Забавно, но большинство людей (в том числе и администраторов!) совершенно не понимают, что такое DEP, зачем он нужен, какие цели преследует и в чем заключается взлом. Достаточно почитать дискуссию, развернувшуюся на http://www.mastropaolo.com/?p=13, чтобы убедится, что оба предложенных сценария обхода DEP не считаются взломом, поскольку x-атрибут присваивается "легальным" способом через вызов VirtualAlloc. На самом деле, суть взлома вовсе не в том, чтобы выполнить код в области памяти без x-атрибута (это действительно невозможно), а в том, чтобы использовать уязвимое ПО в своих хакерских целях. В идеале, DEP должен представлять собой целый комплекс защитных мер, предотвращающих это, но ничего подобного он не делает, ограничиваясь простой формальной поддержкой NX/XD-атрибутов. Перечислим те обстоятельства, которые ему мешают:

- параметры API-функций передаются через стек;
- адреса API-функций и положение вершины стека легко предсказуемы;
- всякий процесс может пометить любой регион памяти как "исполняемый";
- функция VituallAlloc позволяет выделять/передавать уже переданную память;

## >>> врезка прототип функции VirtualAlloc

Функция VirtualAlloc занимается выделением (allocate) и передачей (commit) виртуальной памяти, с одновременной установкой прав доступа на чтение/запись/выполнение:

```
LPVOID VirtualAlloc
(
        LPVOID lpAddress,
        SIZE_T dwSize,
        WORD flAllocationType,
        DWORD flProtect
);
```

**Листинг 2 функция VirtualAlloc и ее прототип**

Назначение аргументов следующее:

| аргумент | значение |
|---|---|
| lpAddress | указатель на передаваемый регион памяти |
| dwSize | размер передаваемого региона |
| flAllocationType | тип запроса, MEM_COMMIT (1000h) — передача памяти |
| flProtect | атрибуты защиты, PAGE_EXECUTE_READWRITE (40h) – rwx |

**Таблица 2 аргументы функции VirtualAlloc**

## в лагере UNIX

Ошибки переполнения не являются "собственностью" Windows. Это общая проблема для всех Си/Си++ программ, родиной которых был и остается UNIX. Уязвимые приложения в изобилии встречаются и там. С ними активно борются путем хитроумных извращений и оригинальных технических решений, о которых мы и собираемся сейчас рассказать.

Начнем с того, что существуют процессорных архитектуры (SPARC, SPARC64, Alpha, HPPA), на которых UNIX имеет неисполняемый стек еще от рождения. Существуют архитектуры, использующие раздельные стеки для хранения адресов возврата и локальных переменных. Подменить адрес возврата на них невозможно, но легко затереть указатель на функцию, при вызове которой управление получит shell-код (правда, передача аргументов представляет собой большую проблему, особенно если они передаться через регистры, тем не менее атаковать такую систему все-таки возможно, пусть и на концептуальном уровне).

На IBM-совместимых машинах LINUX/BSD по умолчанию ведут себя точно так же как XP без SP2. То есть трактуют r-атрибут как -x-, позволяя исполнять код там, где вздумается. Первой ласточкой, ринувшийся на встречу буре, стал патч от Solar'a Designer'a, делающий стек неисполняемым. Хакеров это ничуть смутило (ведь осталась возможность вызова функций через адрес возврата return-to-libc), но зато помешало многим "честным" программам. В общем, большой популярности это решение не получило.

Неисполняемый стек — это всего лишь одна из защитных мер, которая оправдывает себя только в совокупности с целым комплексом остальных. Так, например, бессмысленно ставить бронированную дверь, если можно залезть в окно. Рассмотрим возможности, которые предоставляет популярный пакет **PaX**, распространяемый на бесплатной основе вместе с исходными текстами и подробной технической документацией, из которой можно почерпнуть массу интересного.

Первое и главное — **PaX не требует специальной поддержи со стороны оборудования и не нуждается в битах NX/XD**. Вместо этого он группирует сегменты как показано на рис 2, и устанавливает лимиты так, чтобы они не пересекались. Несомненный плюс такого решения в том, что для защиты от хакеров нам не нужно приобретать новый процессор — это раз. shell-код не сможет выделить исполняемый регион в области кучи или стека — это два. Тоже самое, впрочем, относится и к just-in-time компиляторам, поэтому с совместимостью будут проблемы, которые PaX обходит довольно элегантным путемспособом. При аппаратной поддержке со стороны ЦП (биты NX/XD), он может защищать не только весь процесс целиком, но и его отдельную часть. Допустим, мы имеем приложение, нуждающееся в исполняемом стеке, но не присваивающее x-атрибут явно. Под Windows мы будем вынуждены занести его в список программ, на которые механизм DEP не распространяется, со всеми вытекающими отсюда последствиями. А PaX позволяет отключить защиту лишь для части стека! Конечно, это снижает иммунитет системы, но не так радикально, как полное отключение

DEP. К тому же рандомизация адресов не позволяет shell-коду добраться до этой исполняемой области и заюзать ее в своих целях.

Вторым китом является **технология рандомизации адресного пространства — Address Space Layout Randomization или сокращенно ASLR**. Если при каждом запуске программы непредсказуемым образом менять положение всех сегментов, хакер не сможет определить ни расположение shell-кода, ни адреса API-функций (это утверждение справедливо даже для тех клонах UNIX'a, в которых вызов API-функций происходит через прерывание, например в LINUX/i386, поскольку прикладные процессы общаются с ядром не напрямую, а через разделяемые библиотеки, придерживающиеся соглашения stdcall, и являющиеся своеобразным аналогом KERNEL32.DLL), следовательно, подмена адреса возврата не дает ничего кроме DoS. Пакет PaX позволяет рандомизовать основные сегменты ELF-файла (code, data, bss), кучу, библиотечные функции, стек потока, разделяемую память и стек ядра, короче устраивает настоящий хаос, в котором не остается ничего постоянного, за что можно было бы уцепиться. Ну или практически ничего.

Третий кит – PaX "дорабатывает" функцию mprotect() так, чтобы выдавать x-атрибут могла только система. Никакой прикладной процесс не может изменить атрибуты региона памяти с -x на +x. Так же, никакой регион памяти не может иметь -x- и -w- атрибуты одновременно.

Пакет PaX портирован на множество систем, в том числе и… под Windows. Существует по меньшей мере два приличных порта для NT – BufferShield (см. одноименную врезку) и stackdefender (http://www.ngsec.com/ngproducts/stackdefender/), которые в отличии от штатного DEP'a действительно защищают компьютер от вторжения и преодолеть их ну очень трудно.

Другой популярный пакет — **Exec Shield** (http://people.redhat.com/mingo/exec-shield/), входящий в состав Red Hat Enterprise Linux v.3, update 3, так же использует неисполняемую кучу/стек, замечательно работая на всем семействе x86 процессоров без NX/XD-битов, и частично рандомизует адресное пространство, произвольным образом изменяя базовый адрес стека, расположение разделяемых библиотек и начало области кучи. Остальная память остается нетронутой, однако, для отражения большинства атак и этого оказывается вполне достаточно. Фактически, Exec Shield представляет урезанный вариант PaX'a и не несет в себе ничего нового. А что еще можно ожидать от Red Hat?

Еще хуже дела обстоят в OpenBSD. Начиная с версии 3.3 ядро поддерживает механизм W^X (произносится как "W xor X"), который своему названию предотвращает одновременную установку атрибутов -x- и -w- на любую область памяти, что по замыслу разработчиков должно серьезно озадачить хакеров (http://marc.theaimsgroup.com/?l=openbsd-announce&m=105175475006905&w=2).

На самом деле, эта защита элементарно обходится множественным вызовом функции ~~m~~mprotect. Сначала shell-код вызывает mprotect, устанавливая один лишь атрибут записи (если он не был установлен ранее), затем копирует shell-код через memcpy и вызывает mprotect еще раз, сбрасывая атрибут записи и присваивая себе права исполнения. Кстати говоря, в версии 3.3 W^X не работал на x86, поскольку у того отсутствует возможность задания x-атрибута на уровне страниц, однако, начиная с версии 3.4 этот недостаток был исправлен.

Короче говоря, в штатной конфигурации, без установки пакета PaX, все UNIX-подобные системы потенциально уязвимы и допускают выполнение shell-кода по сценарию, описанному в разделе "атаки на DEP".

## *заключение*

DEP – это не защита! Это рекламный трюк! Если только Microsoft не создаст свой собственный клон PaX, реализованный по всем правилам (что очень навряд ли) черви и хакеры продолжат свое существование. Так что слухи об их кончине в очередной раз оказались преждевременными и сильно преувеличенными. По этому поводу вспоминается один анекдот "Недостаток у танка ровно один: через триплексы ни хрена не видно, куда едешь. Но когда ты едешь на танке, это не очень большой недостаток". Так вот, Microsoft едет в танке, прикрываясь толстой броней монополизма и ее совершенно не волнует куда она едет и что творит.

Не полагайтесь на Microsoft! Для надежной обороны своего компьютера (рабочей станции, сервера) используйте BufferShield или StackDefender, "пробить" который смогут только гуру, да и то с кучей ограничений, делающих червей практически нежизнеспособными.

## >>> BufferShield или PaX на NT

Пакет BufferShield это достойный конкурент штатному DEP'у, не требующий установки SP2, работающий без аппаратной поддержки со стороны процессора, использующий рандомизацию раскладки адресного пространства и задействующий NX/XD биты, если они есть. Он значительно превосходит DEP по защищенности и атаковать его очень сложно (тем не менее, некоторые лазейки все-таки есть, в частности из-за отсутствия GOT, рандомизация выполняется не полностью, оставляя в памяти значительное количество предсказуемых адресов).

Защищать можно как отдельные приложения (например, пресловутый Internet Explorer), так и ~~установленный~~ определенный диапазон памяти внутри конкретного приложения. Так же поддерживаются и многопроцессорные системы (правда, только для "однополых" процессоров, то есть если один процессор имеет поддержку NX/XD битов, а другой нет, NX/XD биты остаются незадействованными для всех процессоров).

Короче говоря, BufferShield реализует те же самые возможности, что и пакет PaX, фактически являясь его портом на NT. Однако, в отличии от бесплатного PaX'a, BufferShield распространяется на коммерческой основе, а для ознакомления предлагается только 30-дневная пробная версия: 30 days trail http://www.sys-manage.com/sites/D_BuffShld.html;

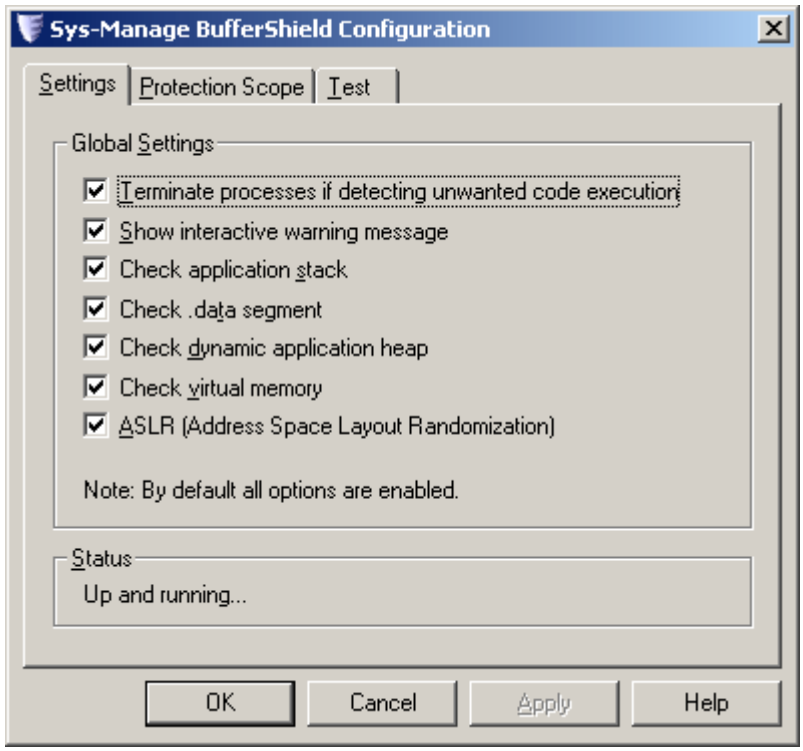


**Рисунок 12 окно настойки BufferShiled'a**

основные свойства продукта:

- распознает выполнение кода на стеке, куче, виртуальной памяти и сегменте данных;
- запрашивает подтверждение на завершение, если обнаружено переполнение буфера;
- ведет отчет обнаруженных переполнение в протоколе событий (Windows event log);
- позволяет включать защиту даже для отдельных частей выбранных приложений (!!!);
- поддерживает NXбит и задействует его, если доступен (но может работает и без него);
- поддерживает симметричные многопроцессорные системы (SMP) с однополыми ЦП;
- рандомизует раскладку адресного пространства (технология ASLR);

## >>> врезка
## ссылки на защитные пакеты, упомянутые в статье

- PaX (лучший бесплатный проектор для LINUX/BSD):
  - http://www.ngsec.com/ngproducts/stackdefender/
- BufferShield 1.01k (коммерческий порт PaX'a на NT):

- http://www.sys-manage.com/index10.htm
- stackdefender (еще один порт PaX'a на NT):
  - http://www.ngsec.com/ngproducts/stackdefender/
- Exec Shield (усеченный клон PaX'a для In Red Hat Enterprise Linux v.3, update 3)
  - http://people.redhat.com/mingo/exec-shield
- W^X (простой и бесполезный защитный пакет для OpenBSD):
  - http://marc.theaimsgroup.com/?l=openbsd-announce&m=105175475006905&w=2

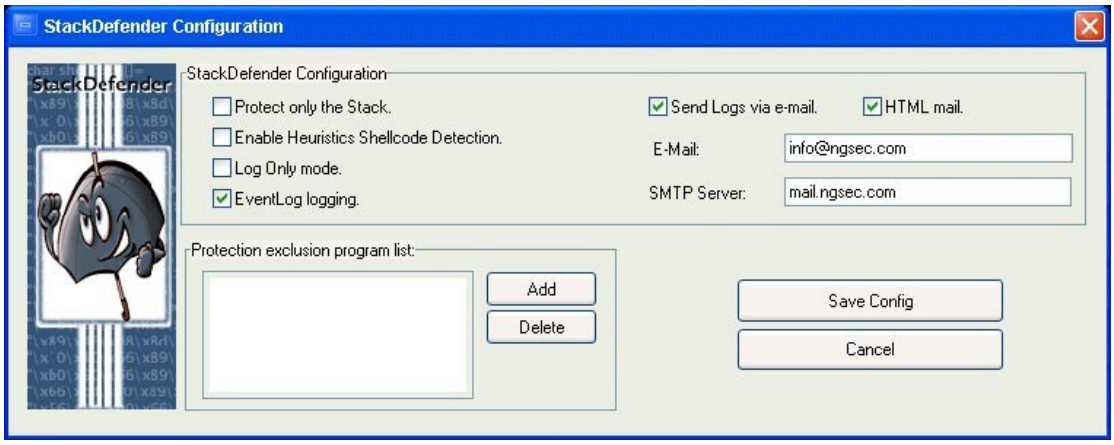


**Рисунок 13 окно конфигурации пакета StackDefender**

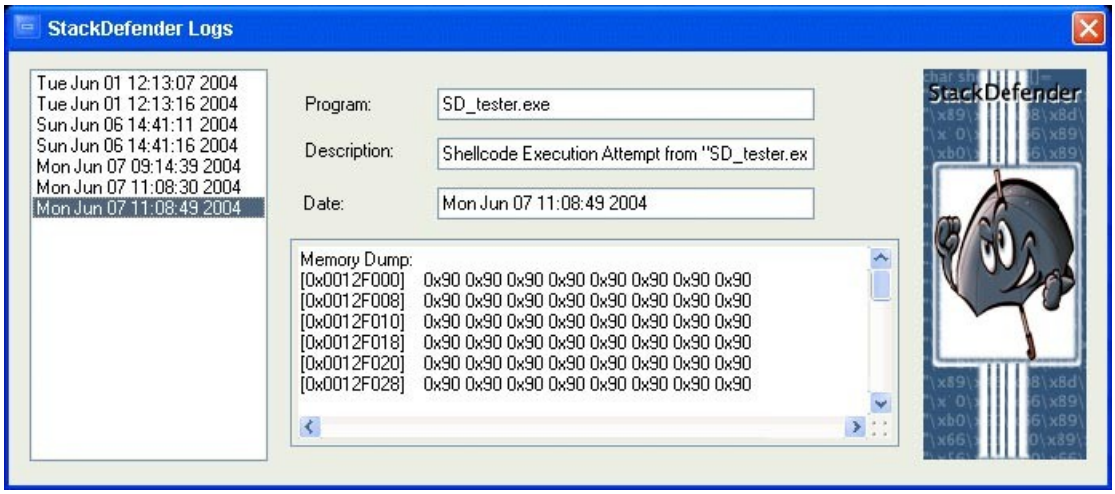


**Рисунок 14 окно протокола пакета StackDefender**

## *>>> глоссарий*

- ASLR (Address Space Layout Randomization - Рандомизация Раскладки Адресного Пространства):
  - мощная технология защиты против shell-кода, основанная на непредсказуемом изменении адресов системных функций, расположения стека, используется в пакете PaX и Exec Shield, но только не в DEP; http://en.wikipedia.org/wiki/ASLR;
- DEP (Data Execution Prevention – Предотвращение выполнения данных):
  - совокупность программных и аппаратных технологий, призванных предотвратить выполнение shell-кода, но в действительности с этим не справляющихся. DEP может быть как аппаратным (hardware-enforced), так и программным (software-enforced). аппаратный DEP фактически сводится к поддержке NX/XD бита, программный представляет собой SafeSEH;
- EVP (Enhanced Virus Protection – Расширенная Вирусная Защита):
  - маркетинговый трюк AMD, выдающий XD бит за "технологию защиты против вирусов"
- PAE (Physical Address Extension – режим расширения физических адресов):

- специальный режим 32-разряных x86 процессоров, в котором они могут адресовать больше памяти и использовать дополнительные возможности (в частности, атрибут NX);

- PTE (Page Table Entry – Элемент Каталога Страниц):
  - структура данных, описывающая свойства "своей" страницы виртуальной памяти и задающая права доступа;
- NX (Not executable – не исполняемый):
  - AMD64: специальный бит в PTE, разрешающий/запрещающий выполнение машинного кода в данной странице;
- XD (execution Disabled – исполнение запрещено):
  - Intel: специальный бит в PTE, разрешающий/запрещающий выполнение машинного кода в данной странице;
- SafeSEH (безопасный SEH)
  - набор защитных мер, разработанных Microsoft для предотвращения использования обработчика структурных исключений (SEH) shell-кодом, позднее был переименован в software-enforced DEP;

## *>>> врезка интересные ссылки*

- **A detailed description of the Data Execution Prevention (DEP)**:
  - официальное описание технологии DEP от Microsoft, крайне поверхностное и неконкретное, но почитать все-таки стоит (на русском и английском языках): http://support.microsoft.com/kb/875352/ru; http://support.microsoft.com/kb/875352;
- **AMD64 Enhanced Virus Protection**:
  - официальная презентация EVP "технологии" от компании AMD. куча роликов и никакой технической информации (на английском языке): http://www.amd.com/us-en/Weblets/0,,7832_11104_11105,00.html;
- **Execute Disable Bit Functionality Blocks Malware Code Execution**:
  - детальная техническая информация от Intel, касающаяся XD бита, ориентированная на системных программистов (на английском языке): http://cache-www.intel.com/cd/00/00/14/93/149307_149307.pdf;
- **Processor Number Feature Table**:
  - перечень процессоров от Intel, поддерживающих XDбит (на английском языке): http://support.intel.com/products/processor_number/proc_info_table072505.pdf;
- **nx: how well does it say NO to attacker's eXecution Attempts**:
  - презентация с Black Hat, раскрывающая принципы работы механизма DEP и перечисляющая его основные уязвимости (на английском языке): http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-maynor.pdf;
- **Buffer overflow attacks bypassing DEP (NX/XD bits) - part 2 : Code injection**:
  - обход DEP, основанный выделении региона памяти посредством вызова VirtualAlloc с последующим копированием shell-кода (на английском языке): http://www.mastropaolo.com/?p=13;
- **Defeating Microsoft Windows XP SP2 Heap protection and DEP bypass**:
  - обход DEP, основанный на переполнении кучи с последующей подменой адреса возврата из функции main на функцию crt!system (на английском языке): http://www.maxpatrol.com/defeating-xpsp2-heap-protection.htm;
- **Windows Heap Overflows**:
  - презентация с Black Hat, описывающая общую принципы переполнения кучи под NT (на английском языке): http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt;
- **DEP evasion technique**:
  - замечательный блог, демонстрирующий технику обхода DEP и сравнивающий его с аналогичными защитными механизмами из UNIX (на английском языке): http://woct-blog.blogspot.com/2005/01/review-of-microsofts-dep.html; http://woct-blog.blogspot.com/2005/01/dep-evasion-technique.html;
- **ошибки переполнения буфера извне и изнутри как обобщенный опыт реальных атак**:
  - уже устаревавшая статья мыщъх'а, подробно описывающая причины и следствия ошибок переполнения различных типов (на русском языке): ftp://nezumi.org.ru/pub/shellcoders.demo.zip;

- **SEH на службе контрреволюции**:
  - еще одна статья мыщъх'а, демонстрирующая технику обхода программного DEP (на русском языке): ftp://nezumi.org.ru/pub/zq-buf-SEH.zip;
- **eWEEK.com Special Report: Windows XP Service Pack 2**:
  - Microsoft подтверждает возможность обхода DEP (на английском языке): http://www.eweek.com/article2/0,1759,1757786,00.asp;