неинициализированные указатели под прицелом осенний сбор дыр в IE открывает новый класс атак

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

сегодня мы поговорим о сравнительно новом и малоизвестном семействе хакерских атак, направленных на неинициализированные указатели, используя в качестве подопытной лабораторной крысы свежие дыры в IE, допускающие удаленных захват управления с передачей управления на shell-код, а так же обсудим перспективы атак на другие приложения

введение

Идея использовать неинициализированные переменные для локальных/удаленных атак пришла в хакерские головы еще лет десять тому назад, однако, ни одного работоспособного exploit'a (не говоря уже о червях!) за минувшее время так и не появилось. Почему? В отличии от переполняющихся буферов (легко обнаруживаемых fuzzer'ами), поиск неинициализированных указателей требует кропотливого ручного анализа кода и творческого вдохновения, без которого разобраться в вековых наслоениях пластов различных модулей (по которым можно изучать хронологию развития Си++) не так-то просто, если вообще возможно.

Хуже того – подавляющее большинство найденных дыр данного типа не представляют никакой хакерской ценности, ведь помимо наличия одной или нескольких неинициализированных переменных нам необходимы еще и методы локального/удаленного воздействия на их содержимое, а так же выработать определенный сценарий атаки, заканчивающейся захватом управления или на худой конец крахом приложения. Но увы! Подобные подарки судьбы встречаются крайне редко и чаще всего неинициализированные переменные приводят к некорректному поведению жертвы, да и то при сочетании кучи достаточно маловероятных обстоятельств.

Какое-то время хакеры носились с этой идей, писали статьи в электронные журналы с полными математическими выкладками, теоретически обосновывающими возможность атаки на неинициализированные переменные, но сами атаки носили единичный характер, не выходящий за рамки лабораторных экспериментов. Тем временем, пока Microsoft (и другие производители) лихорадочно затыкали одни дыры, хакеры открывали другие, прорывая тоннели и заходя на посадочную позу с совершенно невероятных позиций, граничащих со срывом в плоский штопор. Но срыва не было. Вместо штопоры управление получал диверсионно-разведывательный код, открывающий удаленный shell. Как известно, прочность цепи определяется ее слабейшим звеном. Вот точно так и с безопасностью компьютерных систем.

Но цепи, кнуты и нагайки — все это извращ и голимый BDSM. Совсем другое дело — грибы, а грибы собирают по осени, к очередному сбору которых, Microsoft подогнала серию заплаток в IE, связанных с неинициализированными указатели и проходящих под статусом критических, что означает возможность удаленного захвата управления машиной.

Сами заплатки (вместе со скупой технической информацией) содержатся в августовском бюллетене безопасности MS08-45, выпущенным в свет 12 числа (http://www.microsoft.com/technet/security/Bulletin/MS08-045.mspx), однако, поскольку, на самом деле, в IE оказалось гораздо больше неинициализированных указателей, первый блин вышел комом, — не прошло и 9'ти дней, как MS выпустила исправленный набор заплаток, расположенный по тому же самому адресу. Заплатки латают заплатки! С ума сойти!!! Вот такие по осени грибы. Крышу срывает основательно.

Хотя, какие в Америке грибы? Там больше травой балуются, в результате чего мы имеем целых пять официально признанных ошибок:

- HTML Objects Memory Corruption Vulnerability - CVE-2008-2254;
- HTML Objects Memory Corruption Vulnerability - CVE-2008-2255;
- Uninitialized Memory Corruption Vulnerability - CVE-2008-2256;
- HTML Objects Memory Corruption Vulnerability - CVE-2008-2257 и CVE-2008-2258;

Попробуем разобраться, почему возникают неинициализированные переменные в современных программах и как их можно использовать в своих целях.

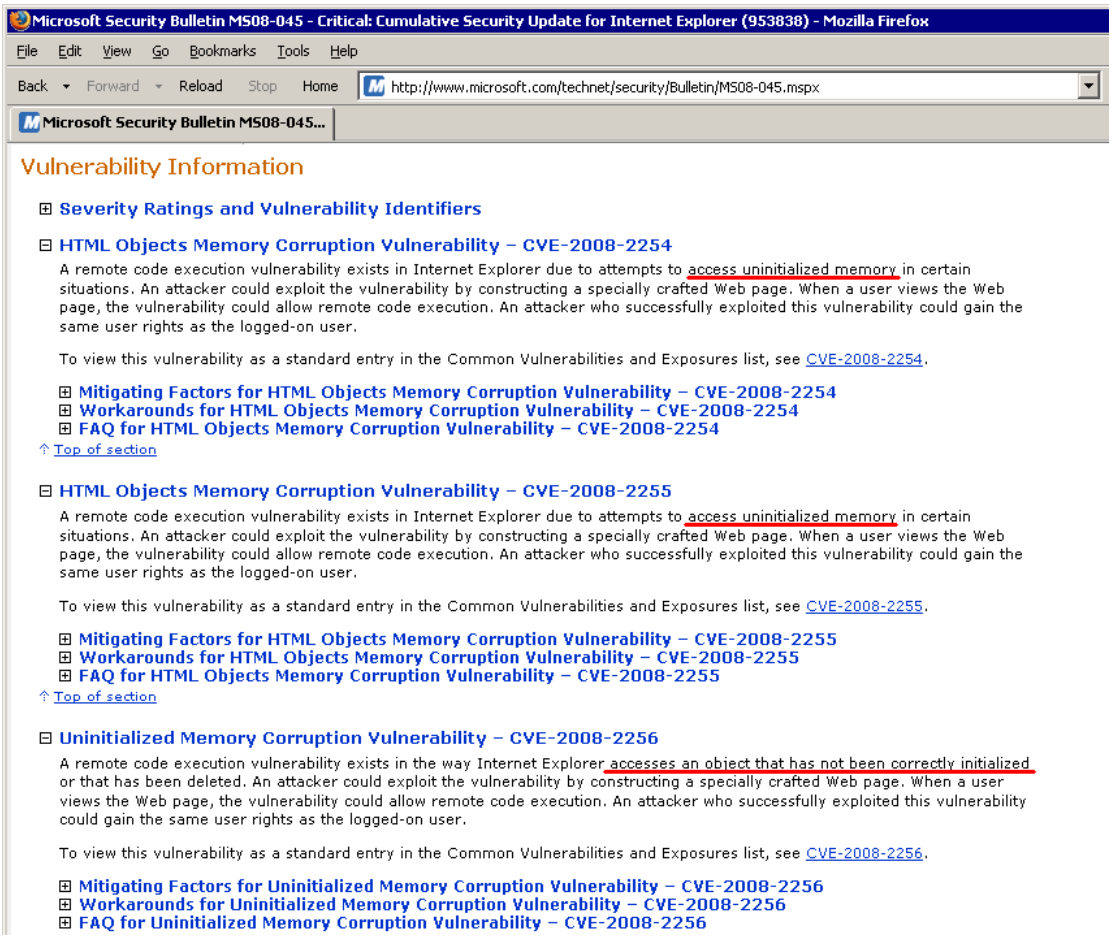

Рисунок 1 осенние ошибки в IE, связанные с доступом к неинициализированным указателям

недостроенные объекты на куче

Мыщъх не устает твердить, что плюсов у Си++ только два, а вот минусов…. Минусов у него _намного_ больше! И вообще… объективно-ориентированный подход создает намного больше проблем, чем их решает, а для борьбы с проблемами, порождаемыми парадигмами ООП, создаются сложные и громоздкие механизмы, пользоваться которыми не рекомендуют даже умудренные жизнью гугу.

Взять хотя бы два фундаментальных понятия: конструкторы и исключения. Конструктор может выбрасывать исключение, при этот _как_ _правило_ деструктор не вызывается и освобождается лишь память, выделенная под сам объект, но объекты, которые конструктор уже успел насоздавать, остаются (в общем случае) не уничтоженными и их поведение зависит от природы самих объектов. Для некоторых объектов (подчиняющихся парадигмам ООП) автоматически вызывается деструктор, удаляющий их по всем правилам, но вот открытые файлы, установленные сетевые соединения и еще куча всякого барахла, продолжает болтаться в памяти, если только обработчик исключения не позаботиться об их освобождении.

Коварство Си++ в том, что для временных объектов (создаваемых компилятором, например, во время передачи аргументов), транслятор _может_ выполнять раскрутку стека (stack unwind), удаляя недостроенные объекты оператором delete. В результате чего при возникновении исключения в конструкторе, вызов деструктора все же происходит, однако… никаких гарантий на этот счет у нас нет. Тут все от типа объекта, особенностей транслятора и ключей компиляции зависит!

Вообще, обработка ошибок в конструкторе — настоящая головная боль и потому программисты очень часто прибегают к разнообразным хакам, самый популярный из которых — забить на конструктор и выполнять инициализацию объекта в отдельном методе

(или даже нескольких методах), условно — Init(). Подобный подход нещадно эксплуатируется разработчиками библиотеки MFC, да и в других проектах — не редкость.

Не будем спорить, что хорошо, а что плохо (хватит разводить теоритический флейм, разработчики MFC не дураки и уж тем более не пионеры). Методы-инициализаторы были, есть и будут — это факт, от которого не уйти (миллионы строк кода не переписать за один день!) и этот факт приводит к возможности появления недостроенных объектов. Действительно, объект создали, а вызывать метод-инициализатор забыли. В результате чего в переменных-членах объекта — мусор. Как следствие, использования недостроенного объекта приводит к непредсказуемому поведению последнего. Чуть позже мы покажем, как можно использовать ошибки этого типа для направленной атаки, а пока обратим внимание на вторую причину возникновения неинициализированных переменных.

Для программ, написанных на смеси чистого и приплюснутого Си, характерна попытка имитации (а, точнее, "эмуляции") некоторых Си++ фич из Си кода, в том числе и виртуальных функций, которые и в самом Си++ реализованы не лучшим образом, а уж их ручная имплементация и вовсе становится источником ошибок. Испортить такую идею!!! А ведь идея и в самом деле очень красива! Создаем структуру (собственно говоря, приплюснутые классы являются типичными сишными структурами, только в Си++ все члены по умолчанию приватные, если только не оговорено обратное). Помещаем в структуру один или несколько указателей на функции, которые могут быть переопределены в любой момент. Очень удобно! Скажем, нужно нам в одном месте заменить системный обработчик правого клика мыши на наш собственный для вывода пользовательского меню или перехватывать вывод документа на печать… да все что угодно!!!

И тут мы плавно переходим к объектам с программируемыми свойствами (или, говоря английским языком, properties). Вместо того, чтобы перечислять в методе-инициализаторе _все_ свойства (большинство из которых остается в состоянии по умолчанию), архитектор проекта пишет объект, поддерживающий по одному методу-инициализатору на каждое свойство, — весьма популярный подход, обязанный своим рождением еще одному косяку в приплюснутом си, который формально поддерживает функции с аргументами по умолчанию, но не позволяет нам менять состояние произвольных аргументов, что на 99% обесценивает идею.

Анализ показывает, что IE написан на смеси классического и приплюснутого си, а так же использует большое количество методов-инициализаторов, вызываемых вручную. То есть, существует возможность вызова недостроенного объекта с неинициированными переменными. Несмотря на то, что большинство объектов написаны вполне корректны и просто отказываются работать без предварительной инициализации, честно возвращая ошибку, без ляпов здесь не обходится и ряд объектов содержат с своем чреве указатели на другие объекты, инициализируемые не конструктором (вызываемом автоматически), а отдельными методами-инициализаторами, вызываемых вручную или… вообще никем не вызываемых. Как следствие, при использовании недостроенного объекта происходит обращение к неинициализированному указателю, содержащему всякий мусор и приложение кончает исключением.

А теперь самое главное! Огромное количество IE-объектов (в том числе и уязвимых!) доступны из скриптовых языков (типа JavaScript или VBScript), что делает возможным направленную атаку на неинициализированные переменные, поскольку, объекты физически размещаются в динамической области памяти, используя единый менеджер кучи. Другими словами, скрипт может выделить блок памяти, записать туда все, что угодно, освободить его и… при последующих запросах памяти на размещение очередного создаваемого объекта, с некоторой степенью вероятности будет использован именно наш блок. Какова степень этой вероятности и как можно ее повысить — мы еще расскажем, а пока подведем краткий итог причин возникновения неинициализированных переменных.

- вынос инициализирующего кода из конструктора (вызываемого автоматически) в метод(ы)-инициализаторы, вызываемые вручную или не вызываемые вообще;
- гибридное Си/Си++ программирование с "эмуляцией" виртуальных функций посредством чистого Си;
- объекты с многочисленными properties, инициализируемыми из закрепленных за ними методов;
- разнотравье неклассифицируемых ошибок;

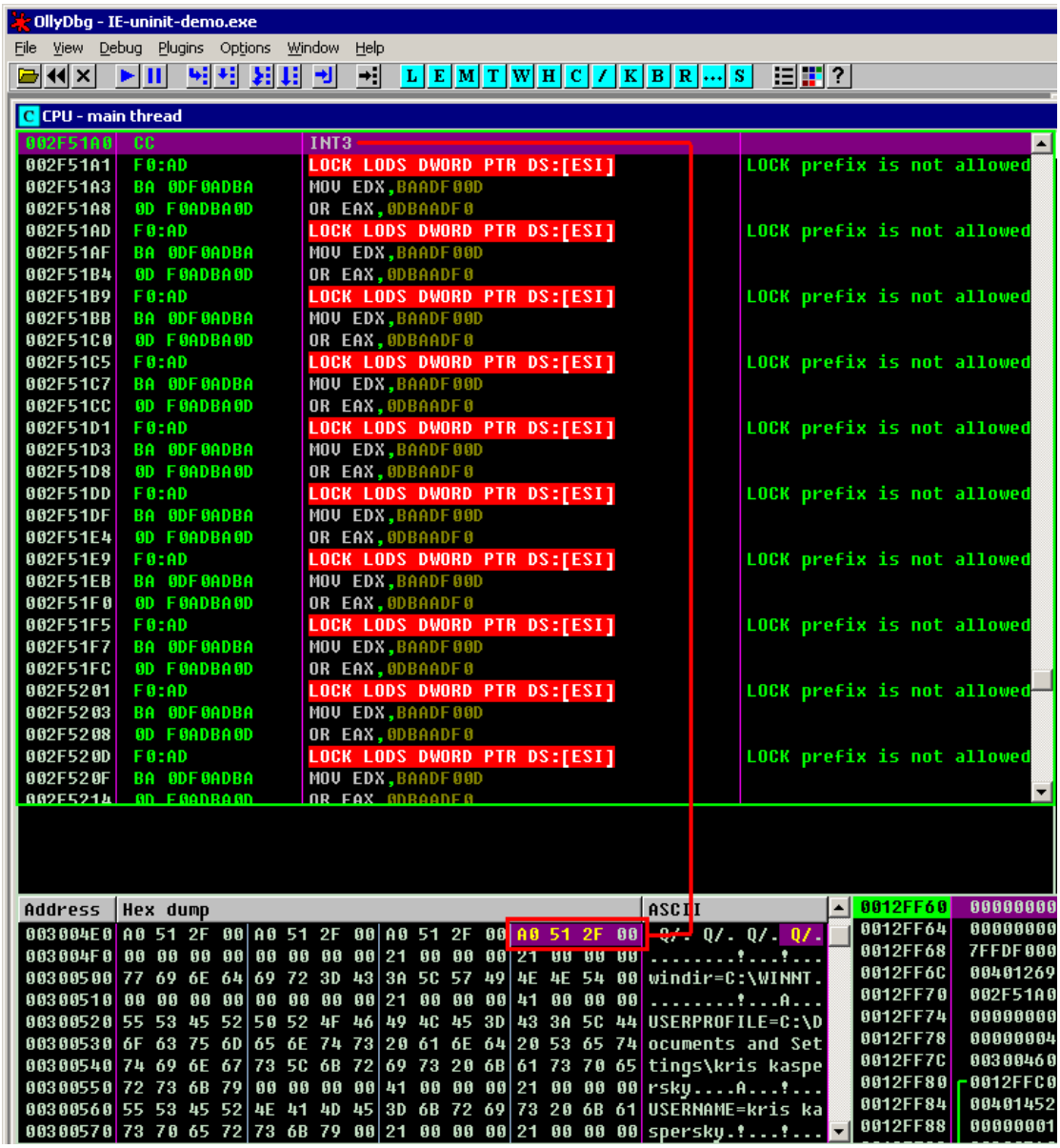

Рисунок 2 передача управления на shell-код путем выделения блоков из кучи, заполнением их указателями на shell-код с последующим возвращением их в пул свободной динамической памяти

недостроенные объекты на стеке

Неинициализированные переменные встречаются не только в куче, но и на стеке тоже. Взять хотя бы такое интересное явление как автоматическая инициализация глобальных и статических переменных, обращаемых в ноль еще не стадии загрузки исполняемого файла в память, а потому, "int a = 0;" и "static a;" содержат в себе одно и тоже значение. Писать "static a = 0;" совершенно необязательно.

Однако, если по каким-то причинам программист вдруг удалит ключевое слово static, превращая переменную "a" из статической в автоматическую, в ней тут же окажется мусор, оставленный на стеке кем-то еще и программа пойдет в разнос. Конечно, такое случается не так уж часто, да и компилятор начнет ругаться матом, выдавая warning'и, но… современные программисты на предупреждающие сообщения не смотрят. Компилируется — и ладно. Да и стековых переменных _намного_ больше, чем членов объекта, размещаемых в динамической памяти, а потому и вероятность нарваться на неинициализированную переменную здесь выше.

К сожалению (или к счастью — смотря с какой стороны смотреть) возможности направленного воздействия на стековые переменные очень ограничены, а сама атака весьма

специфична и заслуживает отдельной статьи, так что отложим этот разговор до лучших дней, а пока вернемся к куче и посмотрим: что там можно сделать.

реализация направленной атаки

Указатели составляют малую часть от всех типов переменных, но мы сосредоточимся именно на них, более того, из всех указателей нас будут интересовать только указатели на функции, поскольку они намного более уязвимы чем все остальные. Достаточно просто "дотянуться" до неинициализированного указателя на функцию, записав туда адрес своей функции (shell-кода), а все остальное за нас сделает автоматика. Но даже такое с виду простое мероприятие реализуется довольно экзотическим способом в стиле "недокументированные позиции кама-сутры".

Как мы уже говорили, стандартный аллокатор приплюснутого си размещает объекты в динамической памяти (куче), причем, для достижения максимальной производительности, обнуление выделенных блоков памяти не производится и там оказывается мусор. А что такое мусор с точки зрения программиста? Правильно, отходы жизнедеятельности предыдущих объектов. Другими словами, возможность (хотя бы теоретическая!) воздействия на неинициализированные указатели у нас есть, правда, практическая реализация наталкивается на множество подводных камней и прочих препятствий. Но все по порядку.

Идея: выделяем блок памяти (что можно сделать напрямую из JavaScript или VBScript), копируем туда shell-код, после чего отъедаем всю память, заполняя ее указателями на наш shell-код и когда память совсем подойдет к концу, освобождаем все блоки, кроме первого (с shell-кодом). Последующие запросы на выделения памяти возвратят уязвимому приложению блок, заполненный ссылками на shell-код и если хотя бы один из указателей объекта окажется неинициализированным, вместо вызова оригинальной функции управление получит shell-код. Описанная техника очень похожа на heap-spray – известный механизм атаки на переполняющиеся буфера, заканчивающийся передачей управления на shell-код, однако, присмотревшись повнимательнее, мы обнаружим существенное различие. В классическом heap-spray'е выделенные блоки не освобождаются, а в нашем — освобождаются все блоки, кроме первого, с таким расчетом, чтобы быть использованными повторно!

А вот теперь первые грабли. Памяти у современных компьютеров много и быстро откушать ее не удастся. Атака растягивается на минуты или даже десятки минут, демаскируя хакера и у жратвы есть хороший запас по времени, чтобы закрыть подозрительно ведущее себя приложение (IE при этом как бы "подвисает"). Но даже не это самое страшное. По умолчанию, начальный размер файла подкачки меньше конечного и в большинстве систем этот размер меньше 2х гигабайт — объема памяти, выделенного каждому процессу под прикладные нужды. Следовательно, в процессе пожирания памяти неизбежно наступает момент, когда система начнет увеличивать размер файла подкачки, что происходит не мгновенно и запросы на выделение памяти, осуществляемые в это время (даже поступающие от посторонних приложений), заканчиваются возвращением ошибки, а что касается стека — то система вообще выбрасывает исключение "исчерпание стековой памяти", которое мало кто обрабатывает. Короче говоря, в процессе увеличения файла подкачки начинают сыпаться совершенно посторонние приложения и пользователь, чертыхаясь, отправляет систему на перезагрузку, в результате чего хакерская атака накрывается медным тазом.

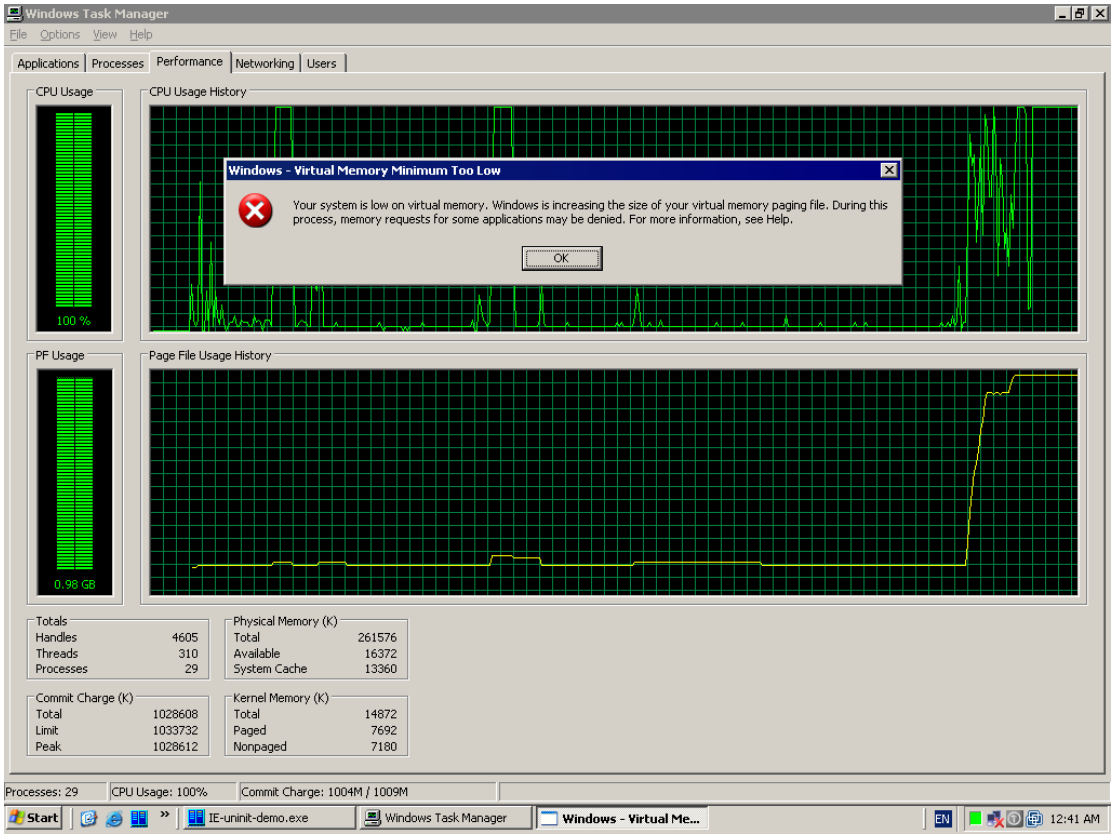

Рисунок 3 тупой сценарий атаки на неинициализированные указатели по методу а-ля heap-spray с колоссальным потреблением памяти

Да… поели грибочков… А как на счет более элегантного сценария? Тут по ходу дела выясняется одна очень интересная вещь. Даже если отъесть всю доступную память, а затем ее освободить, то… никаких гарантий перезаписать неинициализированный указатель у нас нет. Почему? А все потому, что мы выделяли память большими блокам. Если размер выделяемого блока превосходит размер уязвимого объекта, то… система, стремясь, подобрать блок наиболее адекватных размеров, выделит память совсем из других резервов — списка маленьких блоков. Ранее занятых, а теперь освобожденных, но так и не сумевших объединиться с остальными свободными блоками в единое целое, поскольку на пути между ними имеется один или больше занятых блоков, разрывающих единое адресное пространство на множество обособленных суверенных "островков".

Выходит, что для достижения успеха, необходимо выделять блоки памяти предельно компактного размера (4 байта), с таки расчетом, чтобы туда записать один-единственный указатель на shell-код? Ну это вообще дохляк. Дело в том, что размер выделяемого блока автоматически округляется до определенной величины, зависящей от особенностей реализации конкретного библиотечного аллокатора, опирающегося в свою очередь на аллокатор операционной системы. Обычно это 16, 32 или 64 байт. Причем, часть этой памяти (как минимум два двойных слова) расходуется под служебные нужды — указатели на следующий и предыдущий свободный (занятый) блок и потому первые два двойных слова для хакерских махинаций недоступны (на самом деле, доступны и они, но это опять-таки тема отдельной узко-специфичной статьи, заточенной под определенные версии определенных библиотек, а нам хотелось бы познакомиться с универсальным алгоритмом).

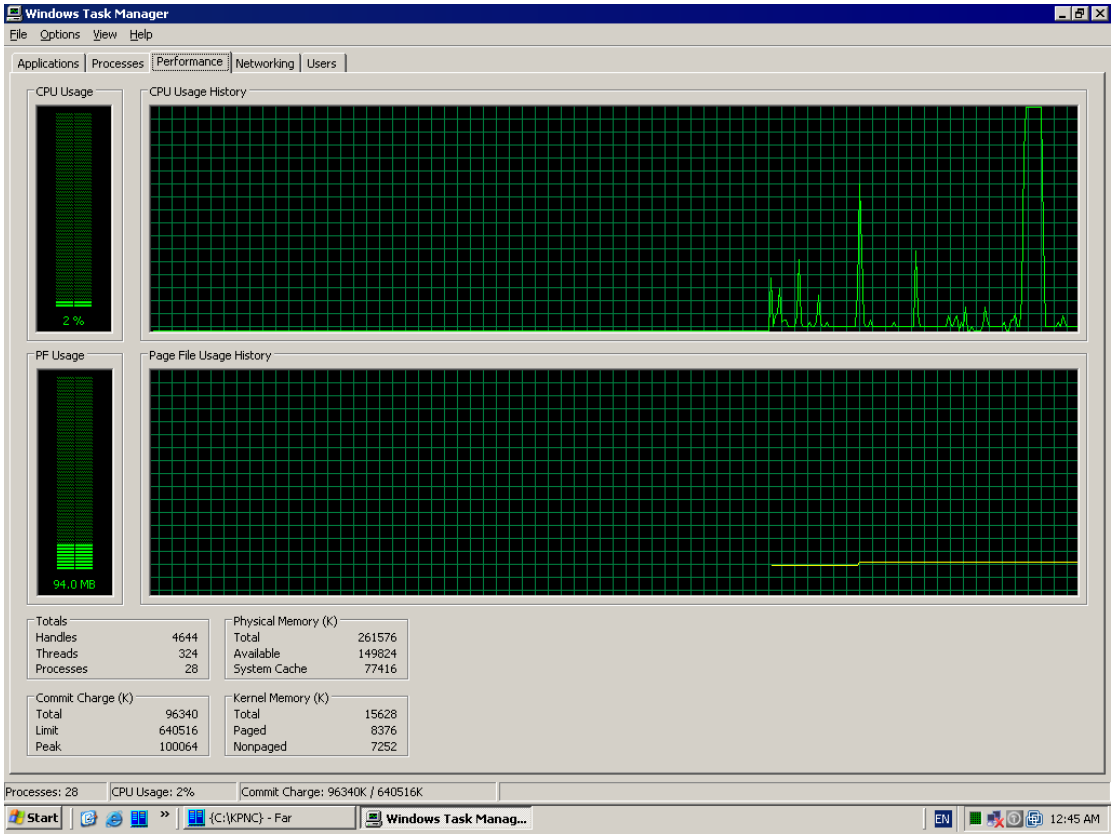

Рисунок 4 продвинутый алгоритм атаки потребляет ничтожное количество памяти

И такой алгоритм действительно есть!!! Достаточно, чтобы размер выделяемых нами блоков памяти совпадал с размером уязвимого объекта. Тогда отъедать всю доступную память уже не потребуется! Вполне хватит нескольких сотен (максимум — тысяч) выделенных блоков, что в общей совокупности (ну путать с совокуплением!) дает порядка одного-двух мегабайт памяти. Как говориться — меньше, чем совсем ничего. Атака совершается, быстро, надежно и незаметно. Жертва даже пикнуть не успевает, как ее уже щемят по полной программе.

Кстати, о программах. Ниже приведен тестовый стенд, демонстрирующий технику использования неинициализированных указателей в хакерских целях (см. листинг 1).

```
#define NNN     (2048)
// #define NSZ  (4096)                          // <-- bad
#define NSZ     (sizeof(struct object))         // <-- good

f_ok() { printf("+OK\n"); }

f_err() { printf("-ERR\n"); }

struct object
{
        char *s;
        int (*bar)();
        int (*foo)();

};

char buf[1023];
char** all_p[NNN];

main()
{
        struct object *Obj;
        int a, b; char *shell, **p;

        // attack
        shell = (char*) malloc(1024); *shell = 0xCC;

        // allocate heap blocks and fill them with pointers to our shell-code
```

```
        for (a = 0; a < NNN; a++)
        {
                p = (char**) malloc(NSZ);
                for (b = 0;b < NSZ / sizeof(char*); b++) p[b] = shell;
                all_p[a] = p;
        }

        // free all blocks to return them in the pool
        for (a = 0; a < NNN; a++) free(all_p[a]);

        // IE-like code
        Obj = (struct object*) malloc(sizeof(struct object));
        Obj->bar = f_ok;

        while(1)
        {
                fgets(buf, 1023, stdin);
                if (strlen(buf) < 8) { Obj->foo(); continue; }
                Obj->s = buf; Obj->bar();
                break;
        }
}
```

Листинг 1 исходный код, демонстрирующий реализацию направленной атаки на неинициализированные указатели

Конструктивно программа состоит из двух частей. Первая — реализует атакующий сценарий, вторая (начинающаяся с комментария "IE-like code") — имитирует уязвимый код, содержащийся, например, в IE. Что касается shell-кода, то он представляет собой однобайтовую машинную команду INT 03h с опкодом CCh – программная точка останова, отлавливаемая отладчиком типа OllyDbg или Soft-Ice (в последнем случае необходимо отдать команду "I3HERE ON", чтобы Soft-Ice реагировал на точки останова, установленные пользовательских приложениях, а не только драйверах, как это он делает по умолчанию).

Естественно, INT 03h – не очень интересный shell-код и потому с практической точки зрения намного полезнее стянуть готовый shell-код с metaspot.com, открывающий back-door или запускающий "Калькулятор". Но это уже несущественные технические детали, не имеющие к описываемому сценарию атаки ни малейшего отношения.

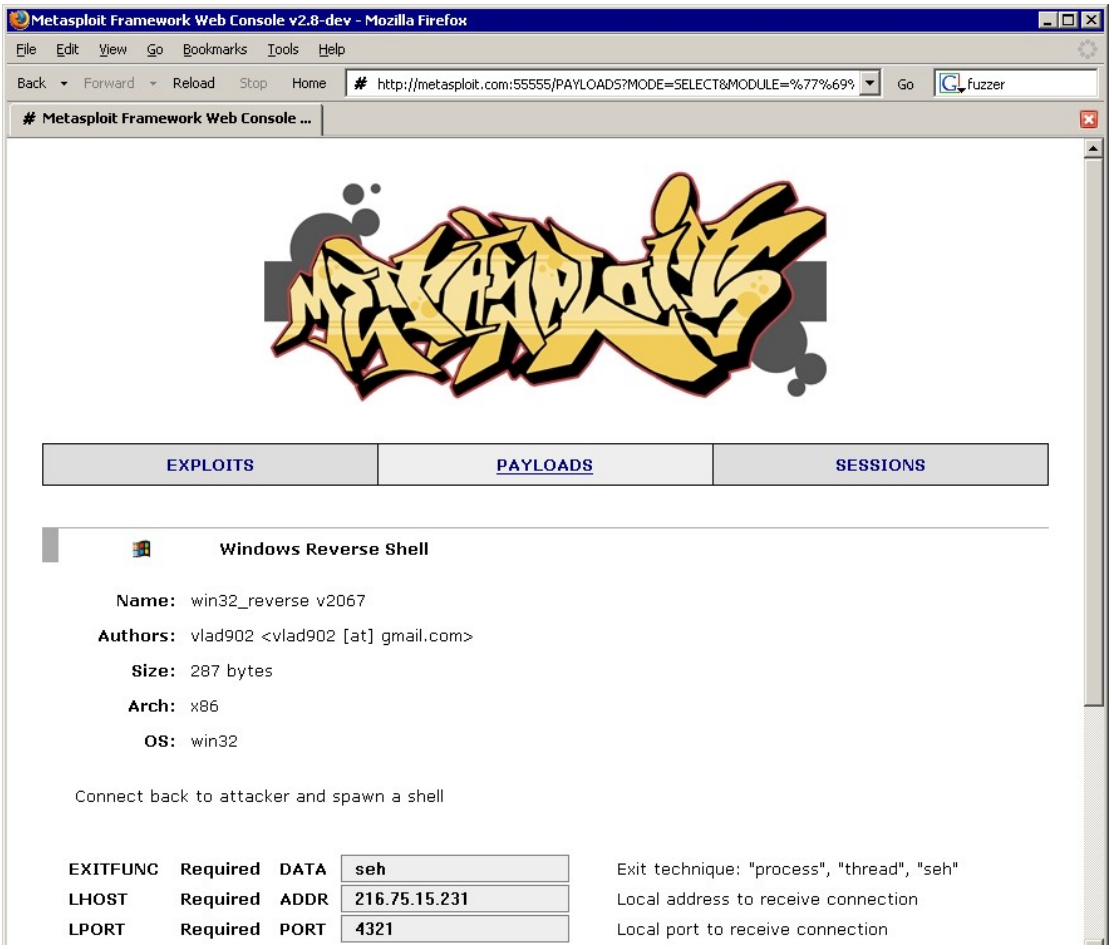

Рисунок 5 богатый выбор боевых shell-кодов на www.metasploit.com

>>> врезка DEP и борьба с ним

Процессоров, поддерживающих NX/XD биты, с каждым днем становится все больше и больше, а, начиная с XP SP2, исполнение кода на стеке и в куче разрешено только для несистемных приложений. IE, увы, не попадает в категорию "амнистированных" и потому приходится осуществлять дополнительные телодвижения. Какие именно — зависит от специфики уязвимого приложения и опять-таки далеко выходит за рамки сценария атаки на неинициализированные указатели, а потому в данном контексте заслуживает лишь беглого упоминания.

заключение

Атаки на неинициализированные указатели только начинаются! К счастью для пользователей и большому хакерскому разочарованию, их очень легко предотвратить. Достаточно заставить аллокатор автоматически очищать выделяемые блоки памяти. Поскольку, большинство приложений взаимодействуют с системным аллокатором не напрямую, а через библиотечные переходники типа malloc и new, то это означает, что для защиты потребуется перекомпиляция всего ранее написанного кода. Так что определенный запас по времени у хакеров все-таки есть — весь вопрос в том, успеют ли они им воспользоваться.