SEH на службе контрреволюции

крис касперски ака мыщъх

перезапись SEH-обработчика – это модный и относительно молодой механизм борьбы с защитой от переполнения буферов в Windows 2003 Server, также находящий себе и другие применения. это отличный способ перехвата управления и подавления сообщений о критических ошибках, демаскирующих факт атаки

введение

Структурной обработкой исключений (***S**tructured **E**xception **H**andling*, **SEH,** в шутку расшифровываемый как ***S**exual **E**xception **H**andling*) называется механизм, позволяющей приложениям получать управление при возникновении исключительных ситуаций (например, нарушениях доступа к памяти, делении на ноль, выполнении запрещенной инструкции), и обрабатывать их самостоятельно, не вмешивая в это дело операционную систему. Необработанные исключения приводят к аварийному завершению приложения, обычно сопровождающемуся всем известным окном "программа выполнила… и будет закрыта".

Указатели на SEH-обработчики в подавляющем большинстве случаев располагаются в стеке, в так называемых **SEH-фреймах**, и переполняющееся буфера могут затирать их. Перезапись SEH-фреймов обычно преследует две цели: перехват управления путем подмены SEH-обработчика и подавление аварийного завершения программы при возникновении исключения. Защита от переполнения буфера, встроенная в Windows 2003 Server, как и многие другие защиты данного типа, функционирует именно на основе SEH. Перехватывая SEH-обработчик и подменяя его своим мы тем самым перекрываем защите "кислород" и она не срабатывает.

Захватывающие перспективы, не правда ли? Во всяком случае, они стоят того, чтобы в них разобраться!

кратко о структурных исключениях

Будучи вполне легальным механизмом взаимодействия с операционной системой, структурная обработка исключений неплохо документирована (во всяком случае нас будет интересовать именно документированная часть).

Внимательнейшим образом проштудируйте раздел "Frequently Asked Questions: Exception Handling" из MSDN. Там же вы найдете замечательную статью Мэтта Питерека "A Crash Course on the Depths of Win32 Structured Exception Handling". Из русскоязычных авторов лучше всего о структурных исключениях рассказывает Volodya – читайте "Об Упаковщиках В Последний Раз", что лежит на wasm'e – (http://www.wasm.ru/article.php?article=packlast01 и http://www.wasm.ru/article.php?article=packers2). Много интересного содержит и заголовочный файл EXCPT.H, входящий в состав SDK. Учитывая, что читатель может быть незнаком со структурными исключениями вообще, кратко введем его в курс дела.

Адрес текущего SEH-фрейма содержится в двойном слове по смещению ноль от селектора FS, для извлечения которого можно воспользоваться следующей ассемблерной абракадаброй: `mov eax,FS:[00000000h]/mov my_var,eax`. Он указывает на структуру типа `EXCEPTION_REGISTRATION`, прототип которой описывается так:

```
_EXCEPTION_REGISTRATION struc
       prev            dd      ?       ; адрес предыдущего SEH-фрейма
       handler         dd      ?       ; адрес SEH-обработчика
_EXCEPTION_REGISTRATION ends
```

Листинг 1 описание структуры EXCEPTION_REGISTRATION

При возбуждении исключения, управление передается текущему SEH-обработчику. Проанализировав ситуацию, SEH-обработчик, кстати говоря представляющий собой обычную cdecl-функцию, должен возвратить либо `ExceptionContinueExecution`, сообщая операционной системе, что исключение успешно обработано и исполнение программы может быть продолжено, либо `ExceptionContinueSearch`, если он не знает, что с этим исключением делать и тогда операционная система переходит к следующему обработчику в цепочке (собственно говоря, возвращать управление необязательно, и SEH-обработчик может

удерживать его хоть до Мамаева пришествия, как обработчики, установленные shell-кодом обычно и поступают).

Последним идет обработчик, назначенный операционной системой по умолчанию. Видя, что дело труба и никто с исключением не справляется, он лезет в реестр, извлекает оттуда ключ `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug` и, в зависимости от его состояния, либо прихлопывает сбойнувшее приложение, либо передает управление отладчику (или, как вариант, Доктору Ватсону).

При создании нового процесса, операционная система автоматически добавляет к нему первичный SEH-фрейм с обработчиком по умолчанию, лежащий практически на самом дне стековой памяти, выделенной процессу. "Дотянуться" до него последовательным переполнении практически нереально, т. к. для этого потребуется пересечь весь стек целиком! *Таких* катастрофических переполненений старожилы не встречали уже лет сто!

Стартовый код приложения, прицепляемый компоновщиком к программе, добавляет свой собственный обработчик (хотя и не обязан это делать), который также размещается в стеке, располагаясь намного выше первичного обработчика, но все же недостаточно близко к переполняющимся буферам, которым потребуется пересечь стековые фреймы всех материнских функций, пока они не доберутся до локальной памяти стартовой функции приложения.

Разработчик может назначать и свои обработчики, автоматически создающиеся при упоминании "волшебных" слов **`try`** и **`except`** (такие обработчики мы будем называть "пользовательскими"). Несмотря на все усилия Microsoft'а, основная масса программистов совершенно равнодушна к структурной обработке исключений (некоторые из них даже такого слова не слышали!), поэтому, вероятность встретить в уязвимой программе "пользовательский" SEH-фрейм достаточно невелика, но все же они встречаются! В противном случае, для подмены SEH-обработчика (а первичный SEH-обработчик в нашем распоряжении есть всегда), придется прибегнуть к индексному переполнению или псевдофункции poke, которую мы обсуждали в предыдущих статьях.

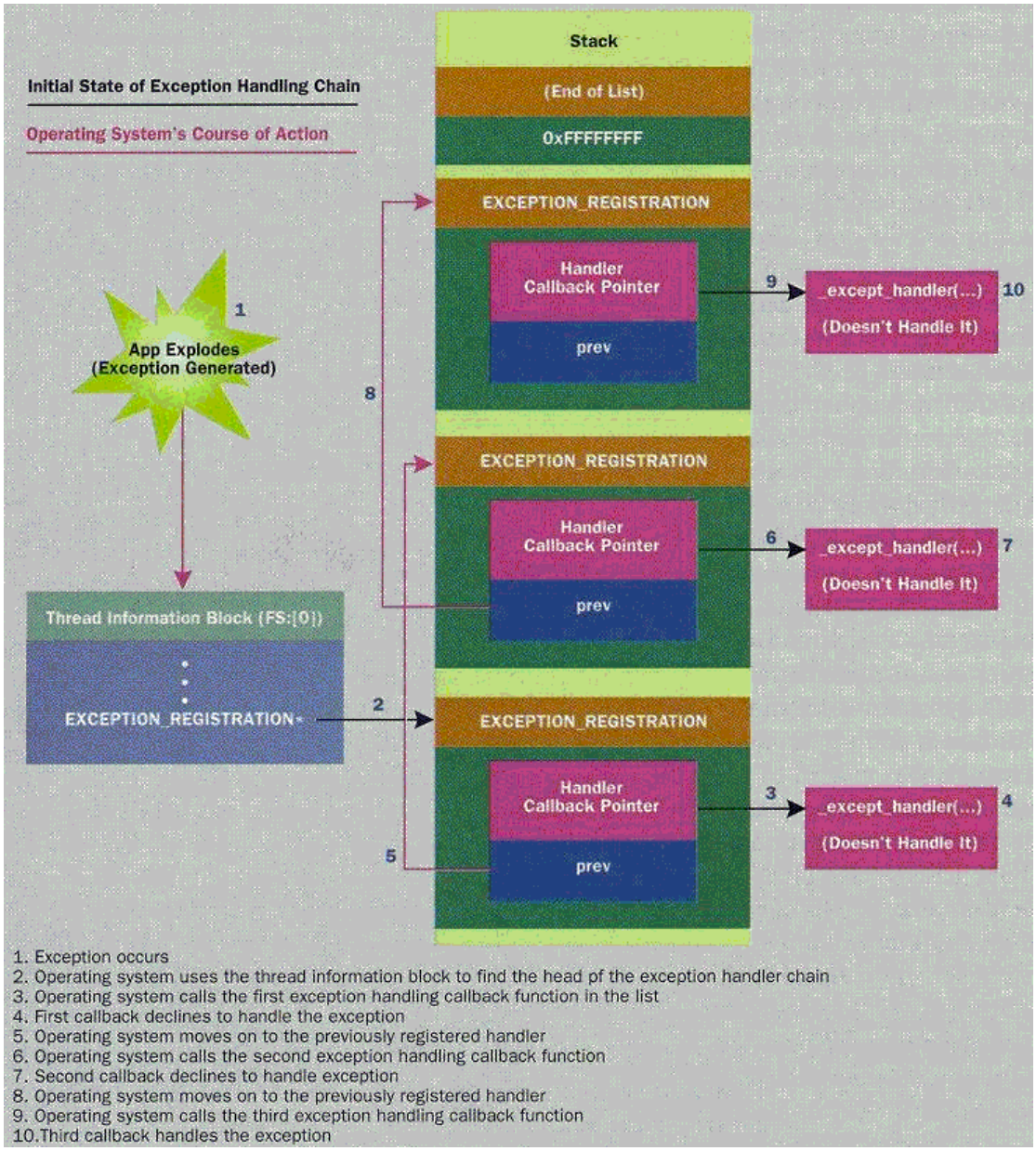

Рисунок 1 глобальное развертывание цепочки структурных исключений (рисунок позаимствован из MSDN).
1 – возникла исключительная ситуация;
2 – операционная система анализирует TIB (Thread Information Bock – Информационный Блок Потока) для поиска первого SEH-фрейма в цепочке;
3 – операционная система передает управление первому SEH-обработчику;
4 – обработчик прикидывается шлагом и уходит в отказ;
5 – операционная система переходит к следующему фрейму в цепочке;
6 – операционная система передает управление SEH-обработчику;
7 – и этот обработчик не знает, что делать с исключением;
8 – операционная система переходит к следующему фрейму;
9 – операционная система передает управление SEH-обработчику;
10 – этот обработчик обрабатывает исключение (не обработать его он не может, т.к. это первичный обработчик, просто прихлопывающий приложение от безысходности)

Для исследования структурных обработчиков исключений, напишем нехитрую программку, трассирую SEH-фреймы и выводящую их содержимое на экран. Законченная реализация может выглядеть, например, так:

```
main(int argc, char **argv)
```

```
{
        int *a, xESP;
        __try{
                __asm{
                        mov eax,fs:[0];
                        mov a,eax
                        mov xESP, esp
                } printf(      "ESP                              : %08Xh\n",xESP);

                while((int)a != -1)
                {
                        printf( "EXCEPTION_REGISTRATION.prev    :%08Xh\n"\
                                "EXCEPTION_REGISTRATION.handler :%08Xh\n\n", a, *(a+1));
                        a = (int*) *a;
                }
        }
        __except (1 /*EXCEPTION_EXECUTE_HANDLER */) {
                printf("exception\x7\n");
        }
        return 0;
}
```

Листинг 2 простой визуализатор SEH-фреймов

Откомпилировав программу и запустив ее на выполнение, мы получим следующий результат (естественно, адреса SEH-фреймов и обработчиков в вашем случае скорее всего будут другими):

```
ESP                             : 0012FF54h   ; текущий указатель вершины стека
EXCEPTION_REGISTRATION.prev     : 0012FF70h   ; "пользовательский" SEH-фрейм
EXCEPTION_REGISTRATION.handler : 004011C0h   ; "пользовательский" SEH-обработчик

EXCEPTION_REGISTRATION.prev     : 0012FFB0h   ; SEH-фрейм стартового кода
EXCEPTION_REGISTRATION.handler : 004011C0h   ; SEH-обработчик стартового кода

EXCEPTION_REGISTRATION.prev     : 0012FFE0h   ; первичный SEH-фрейм
EXCEPTION_REGISTRATION.handler : 77EA1856h   ; SEH-обработчик по умолчанию
```

Листинг 3 раскладка SEH-фреймов в памяти

Смотрите, "пользовательский" SEH-фрейм, сформированный ключевым словом `try`, лежит в непосредственной близости от вершины стека текущей функции и его отделяют всего 1Ch байт (естественно, конкретное значение зависит от размера памяти, выделенной под локальные переменные, ну и еще кое от чего).

Следующим в цепочке идет фрейм, сформированный стартовым кодом. Он расположен намного ниже – от вершины стека его отделяют аж 5Ch байт и это-то в демонстрационной программе, содержащей минимум переменных!!!

Первичный фрейм, назначаемый операционной системой, отстоит от вершины стека на целых 8Ch байт, а в реальных полновесных приложениях и того больше (идентифицировать первичный фрейм можно по "ненормальному" адресу SEH-обработчика, лежащего в старших адресах первой половины адресного пространства). Его линейный адрес, равный 12FFE0h, идентичен для первого потока всех процессов, запущенных в данной версии операционной системы, что создает благоприятные условия для его подмены. Однако, для гарантированного перехвата управления, shell-код должен перехватывать *текущий*, а не первичный обработчик, поскольку до первичного обработчика исключение может и не дожить. Проверьте: если при переполнении буфера бессмысленной строкой наподобие "XXXXX…", возникает стандартное диалоговое окно критической ошибки, подменять первичный обработчик можно, в противном случае, его перезапись ничего не даст и shell-код сдохнет прежде, чем успеет получить управление.

Первичный фрейм всех последующих потоков располагается на `dwStackSize` байт выше предыдущего фрейма, где `dwStackSize` – размер памяти, выделенной потоку (по умолчанию: 4 Мбайт на первый поток и по 1 Мбайту на все последующие). Доработаем нашу тестовую программу, включив в нее следующую строку:

```
CreateThread(0, 0, (void*) main, 0,0, &xESP); gets(&xESP);
```

Листинг 4 исследования раскладки SEH-фреймов во многопоточной среде

Результат ее прогона будет выглядеть приблизительно так:

```
ESP                              : 0012FF48h   ; текущая вершина стека 1го потока
EXCEPTION_REGISTRATION.prev      : 0012FF70h   ; "пользовательский" SEH-фрейм 1го потока
EXCEPTION_REGISTRATION.handler : 00401244h

EXCEPTION_REGISTRATION.prev      : 0012FFB0h   ; SEH-фрейм стартового кода всех потоков
EXCEPTION_REGISTRATION.handler : 00401244h

EXCEPTION_REGISTRATION.prev      : 0012FFE0h   ; первичный SEH-фрейм 1го потока
EXCEPTION_REGISTRATION.handler : 77EA1856h

ESP                              : 0051FF7Ch   ; текущая вершина стека 2го потока
EXCEPTION_REGISTRATION.prev      : 0051FFA4h   ; "пользовательский" SEH-фрейм 2го потока
EXCEPTION_REGISTRATION.handler : 00401244h

EXCEPTION_REGISTRATION.prev      : 0051FFDCh   ; первичный SEH-фрейм 2го потока
EXCEPTION_REGISTRATION.handler : 77EA1856h

ESP                              : 0061FF7Ch   ; текущая вершина стека 3го потока
EXCEPTION_REGISTRATION.prev      : 0061FFA4h   ; "пользовательский" SEH-фрейм 3го потока
EXCEPTION_REGISTRATION.handler : 00401244h

EXCEPTION_REGISTRATION.prev      : 0061FFDCh   ; первичный SEH-фрейм 3го потока
EXCEPTION_REGISTRATION.handler : 77EA1856h

ESP                              : 0071FF7Ch   ; текущая вершина стека 4го потока
EXCEPTION_REGISTRATION.prev      : 0071FFA4h   ; "пользовательский" SEH-фрейм 4го потока
EXCEPTION_REGISTRATION.handler : 00401244h

EXCEPTION_REGISTRATION.prev      : 0071FFDCh   ; первичный SEH-фрейм 4го потока
EXCEPTION_REGISTRATION.handler : 77EA1856h
```

Листинг 5 раскладка SEH-фреймов в памяти

Заметно, что первичный SEH-фрейм всех потоков находится на идентичном расстоянии от текущей вершины стека, что существенно облегчает задачу его подмены. Первичные фреймы первого и второго потоков разнесены на 4 Мбайта (51FFDCh – 12FFE0h == 0x3EFFFC ~4 Мбайт), а остальные – на 1 Мбайт (61FFDCh – 51FFDCh == 71FFDCh – 61FFDCh == 10.00.00 == 1 Мбайт), ну в общем разобраться можно.

Поскольку, большинство серверных приложений конструируются по многопоточной схеме, уметь ориентироваться в потоках жизненно необходимо, иначе вместо перехвата управления, атакующий получит полный DoS. Кстати, об управлении…

перехват управления

Перехват управления можно осуществить двояко (существует по меньшей мере два пути перехвата управления, рассмотрим их):

Путь первый: проанализируйте уязвимую программу и определите какой из обработчиков будет текущим на момент переполнения и где именно расположен его SEH-фрейм (учитывая, что адрес последнего может быть непостоянным и зависящим от множества трудно прогнозируемых обстоятельств, например, от рода и характера запросов, предшествующих переполнению). Теперь придумайте как переполнить буфер так, чтобы затереть `handler`, подменив содержащийся в нем указатель на адрес shell-кода. Значение поля `prev` не играет никакой роли (shell-код ведь не собирается на халяву возвращать таким трудом захваченное управление!).

Путь второй: зарегистрируйте свой собственный SEH-фрейм. Как же мы сможем что-то зарегистрировать в системе, если еще не перехватили управления? – воскликнете вы. Это, что, шутка?! А вот и нет! Указатель на текущего обработчика всегда содержится в одном и том же месте – в первом двойном слове TIB'a, лежащего по адресу `fs:[00000000h]` и псевдо-функцией poke его вполне реально перезаписать. Пусть вас не смущает наличие сегментного регистра FS – вся память, принадлежащая процессу, отображается на единое адресное пространство и до TIB'a можно дотянуться и через другие сегментные регистры, например, через тот же DS, используемый процессором по умолчанию. Естественно, при адресации через DS, TIB будет располагаться совсем по другому смещению и, чтобы его узнать, придется прибегнуть к услугам отладчика. Вы можете использовать soft-ice, Microsoft Kernel Debugger или любой другой отладчик по своему вкусу.

Сначала необходимо определить значение селектора, загруженного в регистр FS. В soft-ice за это отвечает команда "**CPU**" (если soft-ice настроен правильно, то все основные регистры автоматически отображаются в верхней части окна). Затем, просматривая таблицу глобальных дескрипторов, содержимое которой выводит команда "**GDI**", находим соответствующий ему базовый адрес. Для первого потока процесса на всех NT-подобных системах он равен **FFDFF00h,** а все последующие потоки уменьшают его на 1000h, т. е. мы получаем ряд указателей вида: 7FFDE000h, 7FFDD000h, 7FFDC000h…

В любом случае, протестировать вашу машину не помешает (вдруг какая-то из NT поведет себя иначе?). Протокол работы с отладчиком приводится ниже.

```
:cpu

Processor 00 Registers
----------------------
CS:EIP=0008:8046455B  SS:ESP=0010:8047381C
EAX=00000000  EBX=FFDFF000  ECX=FFDFF890  EDX=00000023
ESI=8046F870  EDI=8046F5E0  EBP=FFDFF800  EFL=00000246
DS=0023  ES=0023  FS=0030  GS=0000

:gdt
Sel.  Type      Base      Limit     DPL  Attributes
GDTbase=80036000  Limit=03FF
0008  Code32    00000000  FFFFFFFF  0    P   RE
0010  Data32    00000000  FFFFFFFF  0    P   RW
001B  Code32    00000000  FFFFFFFF  3    P   RE
0023  Data32    00000000  FFFFFFFF  3    P   RW
0028  TSS32     80295000  000020AB  0    P   B
0030  Data32    FFDFF000  00001FFF  0    P   RW
003B  Data32    00000000  00000FFF  3    P   RW
```

Листинг 6 определение адреса указателя на текущий SEH-фрейм

Обратите внимание, FFDFF000h – это не адрес текущего SEH-фрейма. Это – *указатель* на фрейм. Сам же фрейм должен быть сформирован непосредственно в shell-коде, а в FFDFx000h занесен указатель на него (см. рис. 1).

Затем остается лишь совершить что-нибудь недозволенное или же пустить все на самотек, дождавшись пока исковерканная переполнением программа не вызовет исключения естественным путем и тогда наш SEH-обработчик немедленно получит управление. Остальное, как говориться, дело техники…

подавление аварийного завершения приложения

Независимо от того, каким путем shell-код захватил управление, он может зарегистрировать свой собственный обработчик структурных исключений. Это делается приблизительно так:

```
PUSH handler                  ; заносим адрес нашего SEH-обработчика
PUSH FS:[00000000h]           ; заносим адрес на предыдущий SEH-фрейм
MOV  FS,[00000000h], ESP      ; регистрируем новый SEH-фрейм
```

Листинг 7 регистрация собственных обработчиков структурных исключений

Теперь, если shell-код нечаянно дотронется до запрещенной ячейки или совершит другую ошибку подобного типа, атакуемое приложение уже не будет захлопнуто операционной системой и управление вновь возвратиться shell-коду, давая ему понять, что туда ходить не надо и следует немедленно сменить тактику поведения, используя резервные алгоритмы жизнеобеспечения.

Исключения в процессе работы shell-кода могут происходить многократно, главное – следить за тем, чтобы не переполнился стек. Предельно допустимая степень вложенности хоть и велика, но все же не безгранична.

заключение

В структурную обработку исключений был изначально заложен огромный потенциал, только-только начинающий раскрывать себя. Описанные здесь способы перехвата управления – первые ласточки. За структурными исключениями – будущее! Нас ждут десятки хитроумных

трюков, которые еще предстоит найти. И какие бы изощренные защитные механизмы ни придумались, у нас есть что им противопоставить!