7. Скрытая установка SEH-обработчиков Крис Касперски, 01.12.2008

○ Комментарии

◆ 311

○ Добавить в закладки

взлом

Энциклопедия антиотладочных приемов

Содержание статьи

```
04. Кража чужих обработчиков
05. Рукотворный SetUnhandledExceptionFilter
Продолжая окучивать плодородную тему структурных исключений, погово-
рим о методах скрытой установки SEH-обработчиков, используемых для зат-
руднения дизассемблирования/отладки подопытного кода, а также обсудим
возможные контрмеры антиантиотладочных способов.
```

ПОСТАНОВКА ПРОБЛЕМЫ Структурные исключения представляют собой мощное антиотладочное средство, в чем мы уже убедились на примере предыдущих выпусков. Там же мы познакомились и с техникой

исследования программ, играющихся исключениями, работу с которыми достаточно трудно

Всякий раз, когда в тексте программы встречается конструкция MOV FS:[0], xxx, хакер сра-

зу встает торчком — раз это FS:[0], значит, программа устанавливает собственный SEH-

замаскировать.

souriz()

обработчик и, судя по всему, сейчас будет бросать исключения. Теоретически возможно засунуть MOV FS:[0], ххх в самомодифицирующийся код, убрав его из дизассемблерных листингов, однако против аппаратной точки останова по записи на MOV FS: [0], ххх ничто не спасет. В момент установки нового SEH-обработчика отладчик тут же «всплывет», демаскируя защитный механизм. A SetUnhandledExceptionFilter вообще представляет собой API-функцию, экспортируемую KERNEL32.DLL, которую легко обнаружить любым API-шпи-

оном, даже без анализа всего дизассемблерного кода! Задача: установить собственный обработчик структурных исключений, но так, чтобы это как можно меньше бросалось в глаза и не палилось тривиальной установкой точек останова. Решением мы сейчас, собственно, и займемся, предложив широкий ассортимент антиотладочных трюков, один интереснее другого. ПЕРЕЗАПИСЬ СУЩЕСТВУЮЩЕГО ОБРАБОТЧИКА Вместо того чтобы устанавливать новый обработчик структурных исключений, некоторые (и достаточно многие) защиты предпочитают модифицировать указатель на существующий. Даже если приложение и не устанавливает никаких SEH-обработчиков, система все равно впихивает ему SEH-обработчик по умолчанию, смотрящий куда-то в дебри KERNEL32.DLL.

и более поздних версиях). Обработчик по умолчанию не делает ничего полезного, и потому без него можно обойтись, «позаимствовав» указатель — на время или навсегда. Конкретный пример реализации приведен ниже.

Установка своего SEH-обработчика без перезаписи ячейки FS:[0]

На этом, кстати говоря, основан популярный примем поиска базового адреса загрузки KERNEL32.DLL, в котором нуждается shell-код, а также программы, написанные без использования таблицы импорта (из-за ошибки в системном загрузчике они работают только на ХР

__asm{ mov eax, fs:[0]

```
add eax, 4
        mov [eax], ecx
   return *p;
Внешне код очень похож на классический способ установки SEH-обработчика, но, прис-
мотревшись внимательнее, мы видим, что в нашем примере модифицируется отнюдь
не ячейка FS:[0], а то, на что она указывает. Точка останова по записи на FS:[0] уже не сра-
ботает, однако сегментный регистр FS режет глаз, да и бряк на FS:[0] по доступу продол-
жает работать, а потому для эффективного противодействия хакеру требуются дополнитель-
ные уровни маскировки. Ну и чего мы сидим? Вперед!
 main()
                 int *p=0;
                     asm{
```

} return *p; \$SEH-01.exe hello, nezumı Скрытая установка SEH-обработчика

ПРЯЧЕМ FS Ослепить дизассемблеры совсем не трудно. Перезаписать указатель на системный SEHобработчик можно и без явного использования сегментного регистра FS. Самое простое, что можно сделать, — скопировать его в любой другой сегментный регистр (например, GS). С точки зрения процессора, регистры FS и GS совершенно равноправны. Главное, чтобы в регистре содержался «правильный» селектор, а его название — уже дело десятое. Создавать новые селекторы мы не можем (точнее, можем, но это тема отдельного разговора), а вот загружать существующие — почему бы и нет? Усиленный фрагмент защиты приведен ниже. Прячем регистр FS от любопытных глаз $__asm{}$ mov ax, fs mov gs, ax $__asm{}$ mov eax, gs:[0]

Небольшое пояснение. Поскольку ни один известный мне компилятор не использует регистр GS для своих целей, его можно инициализировать в одной процедуре, а использовать в другой. Единственное условие — обе процедуры должны принадлежать одному потоку,

Начинающих хакеров обращение к регистру GS дробит на части, сваливая в вертикальный штопор. Короче, это как обухом по голове. «Ольга» (в отличие от «Айса») не показывает зна-

Опытных реверсеров таким макаром не проведешь, но никаких гарантий, что GS в данный момент содержит именно FS, а не, например, DS, у нас нет. А потому статический анализ ста-

Самое интересное, что блок окружения потока, засунутый в селектор (который хранится в сегментном регистре FS), отображается на плоское адресное пространство, а значит, доступен для чтения и через остальные селекторы. Например, через сегментный регистр DS. На W2K блок окружения первичного потока начинается с адреса 7FFDB000h (7FFDE000h

на XP), поэтому вместо FS:[0] допустимо использовать конструкцию DS:[7FFDB000h]. Что-

бы избежать краха, надо отталкиваться от того факта, что в настоящем блоке окружения

поскольку каждый поток обладает собственным регистровым контекстом.

чений сегментных регистров, чем серьезно осложняет ситуацию.

но точки останова — это уже динамический, а не статический анализ!

потока по смещению 30h байт от его начала расположен указатель на блок окружения процесса, лежащий на 1000h байт ниже. Благодаря чему мы можем найти указатель на SEHобработчик даже на неизвестной операционной системе! Конечно, реализация алгоритма существенно усложняется, но это даже хорошо, поскольку чем больше строк кода — тем дольше их будет анализировать хакер, особенно если эти строки бессмысленны сами по себе.

pp -= 0x1000;if (IsBadReadPtr(pp, 4)) continue; if (IsBadReadPtr((pp + 0x30), 4)) continue; if $(*((size_t^*)(pp + 0x30)) == ((size_t) pp + 0x1000))$ *(size t*) (*((size t*)pp) + 4) = (size t*) souriz; return *p; } printf("not found\n");

Во-первых, мы обошлись без ассемблерных вставок, реализовав алгоритм на чистом С (с тем же успехом можно использовать паскаль). Во-вторых, вместо характерного FS в программе появилась куча констант, смысл которых понятен только посвященным, да и то не без пристального анализа, сопровождаемого глубокой медитацией. В-третьих, факт передачи управления на функцию souriz по return *p (где p == 0) совершенно не оче-

виден. К тому же сам указатель на souriz можно зашифровать, помешав дизассемблерам

реконструировать перекрестные ссылки. Как это сделать на С (без ассемблерных вставок),

Существуют и другие способы поиска указателя на блок окружения потока. Рассмотрим только два самых популярных. Просматривая карту памяти (а просмотреть ее можно с помощью API-вызова VirtualQuery), даже удав заметит, что блоки окружения процесса и потока лежат в своих собственных секциях памяти с атрибутами Private и правами на чте-

описывалось в 1Eh выпуске сишных трюков.

002F 0000|

00401000

99496999

00006000

00400000|00001000|SEH-06

00405000|00001000|SEH-06

77F80000|00001000|ntdll

77F81000|00044000|ntdll

77FC5000|00005000|ntd11

77FCA000|00004000|ntd11

77FCE000|00003000|ntd11

77FD1000|00001000|ntdll

77FD2000|00028000|ntd11

77FFA000|00003000|ntd11

79430000|00001000|KERNEL32|

79431000|00059000|KERNEL32|

7948A000|00004000|KERNEL32|

7948E000|00052000|KERNEL32|

794E0000|00004000|KERNEL32|

не заботит, и они используют его сплошь и рядом.

7F6F0000|00007000

7FFB0000|00024000

7FFDE000|00001000

7FFDF 000| 00001 000

7FFE0000|00001000

| 00004000| SEH-06

| 00002000| SEH-06

00300000|00008000

00410000|00002000

процесса расположен по смещению 30h байт от блока окружения потока. То есть если *(($size_t^*)(block_1+30h)) == block_2, то block_1 - блок окружения потока, a block_2 - блок$ окружения процесса и MOV EAX, FS:[0] равносильно MOV EAX, block 1/MOV EAX, [EAX]. Вывод: без FS можно по-любому обойтись.

Address Size Owner Section Contains Type Access Initial 00010000|00001000| Priv RW RWPriv RW RW 00001000 00020000| 0012D000|00001000 Priv RW Guar RW stack of maiPriv RW Guai RW 0012E000|00002000 00130000|00004000 Prio RW RW RW 00230000|00003000 RW Map 00240000|00016000 R Мар 00260000|0002F000 R Map R 00290000|00041000 Мар 002E0000|00004000 Map

.text

.data

.text

ECODE

PAGE

.data

EDATA

.rsrc

.text

.data

.rsrc

.reloc

Блок окружения потока на карте памяти процесса

Указатель на блок окружения потока также находится в стеке потока, куда его кладет операционная система. В W2K/XP это третье двойное слово от вершины. И хотя в последующих

версиях его местоположение может измениться, вирусов это обстоятельство, похоже, никак

И что в итоге? Мы рассмотрели множество приемов скрытого обращения к ячейке FS:0,

.reloc

.rdata

Priv|RW

Priv|RW

Imag R

Imag R

Imag R

Imag R

Imag|R

Imag R

Imag R

Imag|R

Imag|R

Imag R

Imag | R

Imag R

Imag R

Imag|R

Imag R

Map

data block (Priv RWE

Map | R E

Priv RWE

Priv|R

Imag

Imag

Map

PE header

PE header

resources

PE header

resources

relocations

code,import

relocations

code,export

imports

code

data

code

code

data

data

RW

RW

RWE

RΕ

RWE

RWE

R

R

```
какой адрес используется — смещение 0 по селектору FS или же смещение 7FFDB000h
по селектору DS.
Непорядок! Хорошая защита должна справляться не только с дизассемблерами, но и
с отладчиками.
КРАЖА ЧУЖИХ ОБРАБОТЧИКОВ
Системный обработчик структурных исключений расположен на дне стека потока — и обра-
щаться к блоку окружения для его поисков совсем не обязательно, поскольку местополо-
жение обработчика непостоянно и зависит от версии операционной системы. С учетом этого
мы должны выработать эвристический алгоритм поиска.
Системный обработчик, назначаемый по умолчанию, есть не что иное, как функция
__except_handler3, расположенная в недрах KERNEL32.DLL и не экспортируемая наружу,
но присутствующая в отладочных символах. Которые теоретически можно в любой момент
скачать с серверов Microsoft, но практически такое решение будет слишком громоздким,
неудобным, ненадежным, да и довольно «прозрачным» для хакера.
 Address Hex dump
                                                                     0012FFD8 0012FFC8
79432828 FF FF FF FF 00 00 00 00 00 5F 45 79 00 00 00 00 98 9898....purg....

79432838 FF FF FF FF 00 00 00 00 00 5F 93 45 79 53 00 61 00 98989...._~EyS.a.

79432848 66 00 65 00 44 00 6C 00 6C 00 53 00 65 00 00 00 r.c.h.M.o.d.e...

79432868 72 00 63 00 68 00 4D 00 6F 00 64 00 65 00 00 00 r.c.h.M.o.d.e...

79432868 79 00 5C 00 4D 00 41 00 69 00 73 00 74 00 72 00 \R.e.g.i.s.t.r.
                                                                    0012FFDC 00000000
                                                                    0012FFE4 00401000 SE handler
                                                                    0012FFEC 00000000
0012FFF0 00000000
 79432888 45 00 5C 00 53 00 79 00 73 00 74 00 65 00 6D 00 E.\.S.y.s.t.e.m
79432898 5C 00 43 00 75 00 72 00 72 00 65 00 6E 00 74 00 \.C.u.r.r.e.n.t
794328A8 43 00 6F 00 6E 00 74 00 72 00 6F 00 6C 00 53 00 C.o.n.t.r.o.l.S
                                                                    0012FFF4 00000000
                                                                    0012FFF8 004010E3 SEH-06.<ModuleEntryPoint>
 79432888 65 00 74 00 5C 00 43 00 6F 00 6E 00 74 00 72 00 e.t.\.C.o.n.t.r
794328C8 6F 00 6C 00 5C 00 53 00 65 00 73 00 73 00 69 00 o.l.\.S.e.s.s.i
                                                                     0012FFFC
                                                                             00000000
 Access violation when reading [00000003] - use Shift+F7/F8/F9 to pass exception to progran
                   Указатель на системный SEH-обработчик лежит на дне стека потока
Хорошо, будем отталкиваться от того, что __except_handler3 смотрит в KERNEL32.DLL и что
перед ним всегда расположено двойное слово FFFFFFFh, а после него — указатель на сек-
цию данных KERNEL32.DLL, опять-таки содержащий в себе двойное слово FFFFFFFh. Пос-
леднее обстоятельство системно-зависимо, но справедливо как для W2K, так и для XP,
а потому его можно использовать без особых опасений.
Практический пример приведен ниже:
Прямой поиск указателя на SEH-обработчик в стеке
for (a=0;a<69;a++,pp++)
     if (IsBadReadPtr((pp+2), 4)) break;
     if (*pp == 0xFFFFFFFF)
          if (IsBadReadPtr(*(pp + 2), 4)) continue;
          if (*((unsigned int*)*(pp + 2)) == 0xFFFFFFF)
                *(pp + 1) = (unsigned int*) souriz;
                return *p;
} printf("not found\n");
Точка останова на FS:0 на этот раз идет лесом и не срабатывает, поскольку обращения
к этой ячейке памяти уже не происходит. К тому же разобраться, что именно ищет прог-
рамма в стеке, можно после серии экспериментов (ну или чтения этой статьи). Способов
поиска системного обработчика исключений намного больше одного. Это существенно
```

усложняет задачу хакера, и универсальных «отмычек» тут нет, что в плане защиты очень даже хорошо. Однако просмотр цепочки обработчиков структурных исключений (в «Ольге» через меню View → SEH Chain) немедленно разоблачает хакнутый обработчик, на который несложно установить точку останова на исполнение со всеми вытекающими отсюда пос-

→ L E M T W H C / K B R ... S = ?

🔀 SEH chain of main thread

Address | SE handler

| 0012FFB0|SEH-06.00402538

10012FFE0|SEH-06.00401000

_ 🗆 ×

ледствиями.

🙀 OllyDbg - SEH-06.exe

Log

<u>M</u>emory

<u> T</u>hreads

<u>W</u>indows

SEH chain

Patches

Call stack

W<u>a</u>tches

Breakpoints

<u>R</u>eferences

Run trace

Source files

Source:

Te<u>x</u>t file

.text:7945BC45

.text:7945BC45

ЧТО

Bce,

.text:7945BC45 8B 4C 24 04

.text:7945BC54 C2 04 00

.text:7945BC49 A1 F0 A1 48 79

нам

text:77E2D187

text:77E2D193

text://E2D194

text:77E2D197

text:77E2D199 text:77E2D19F

text:77E2D1A5

text:77E2D1A6

text:77E2D1A7

text:77E2D1AC

text:77E2D1AE text:77E2D1B4

text:77E2D1B4 text:77E2D1B6

text:77E2D1BC

text:77E2D1C2

text:77E2D1C4

text:77E2D1CA

text:77E2D1B4 loc_77E2D1B4:

text:77E2D1CA loc_77E2D1CA:

text

text

77E2D180

.text:7945BC4E 89 0D F0 A1 48 79

00401037 ... EB 12

00401039 > 8B4D F8

<u>H</u>andles

<u>⊂</u>PU

Heap

C

File View Debug Plugins Options

Executable modules Alt+E

Alt+L

Alt+M

Alt+C

Ctrl+P

Alt+K

Alt+B

<u>W</u>indow <u>H</u>elp

PUSH EBF

PUSH ECX

ADD ESP.4

MOV ESP,EBP

MOV EBP,ESP

SUB ESP, OC

РУКОТВОРНЫЙ SETUNHANDLEDEXCEPTIONFILTER

PUSH 0

POP EBP

PUSH EBP

RETH

MOV EBP, ESP

PUSH SEH-06.00406030

CALL SEH-06.004010B2

CALL DWORD PTR DS:[<&

MOV DWORD PTR SS:[EBP LEA EAX, DWORD PTR SS: MOU DWORD PTR SS:[EBP

MOU DWORD PTR SS: [EBP 51,

JMP SHORT SEH-06.0040104B

MOV ECX, DWORD PTR SS: [EBP-8]

Просмотр SEH-цепочек в «Ольге»

API-функция SetUnhandledExceptionFilter, как уже отмечалось в предыдущих выпусках,

ную переменную BasepCurrentTopLevelFilter, хранящуюся внутри KERNEL32.DLL и исполь-

Дизассемблерный листинг API-функции SetUnhandledExceptionFilter из W2K

mov ecx, [esp+lpTopLevelExceptionFilter]

mov eax, BasepCurrentTopLevelFilter

mov _BasepCurrentTopLevelFilter, ecx

BasepCurrentTopLevelFilter

импортировать

Самый большой недостаток функции SetUnhandledExceptionFilter в том, что ее вызов очень трудно замаскировать, но трудно еще не значит невозможно. К тому же реализация функции проста как движок от «запора». Фактически она всего лишь устанавливает глобаль-

.text:7945BC45 _SetUnhandledExceptionFilter@4 proc near

.text:7945BC45 lpTopLevelExceptionFilter= dword ptr 4

retn

.text:7945BC54 _SetUnhandledExceptionFilter@4 endp

нужно,

4

ЭТО

необходимости

найти

SetUnhandledExceptionFilter (или UnhandledExceptionFilter) и прописать сюда ука-

затель на свой собственный обработчик исключений. К сожалению, это не избавляет нас

SetUnhandledExceptionFilter/UnhandledExceptionFilter или получить эффективный

адрес путем ручного разбора таблицы экспорта KERNEL32.DLL. Да, конечно, ручной разбор

с использованием хеш-сум вместо имен АРІ-функций до некоторой степени скрывает наши намерения от хакера. Увы, нет ничего тайного, что не стало бы явным. Даже если выб-

зуемую только функцией UnhandledExceptionFilter.

ранный хеш-алгоритм математически необратим, запустив программу под отладчиком, всегда можно установить, какой именно АРІ-функции какой хеш соответствует. ☐ IDA View-A ___ public SetUnhandledExceptionFilter text:77E2D187 SetUnhandledExceptionFilter proc near text:77E2D187 text:77E2D187 var_220 = dword ptr -220h text:77E2D187 lpTopLevelExceptionFilter= dword ptr 4 text:77E2D187 arg 4 = dword ptr 8 text:77E2D187 text:77E2D187 FUNCTION CHUNK AT .text:77E18ECE SIZE 00000007 BYTES FUNCTION CHUNK AT .text:77E2D31E SIZE 00000048 BYTES text:77E2D187 FUNCTION CHUNK AT .text:77E574F0 SIZE 0000000E BYTES text:77E2D187 FUNCTION CHUNK AT .text:77E63669 SIZE 00000027 BYTES text:77E2D187 text:77E2D187

edi, edi

ebp, esp esp, 220h

esi, [ebp+arg_4]

[ebp+var_220]

; CODE XREF: SetUnhandledExcepti

loc_77E18ECE

sub_77E2D248

eax, eax loc_77E574F0

esi, esi loc_77E18ECE

loc_77E2D31E

ebx, ebx

ebx, dword_77ECBFD4

ebp

ebx

eax,

MOV

MOV

sub

push

push

MOV

test

lea

push

push

call

test

test

MOV

test

push

CODE XREF: SetUnhandledExcepti SetUnhandledExceptionFilter+1D text:77E2D1CA text:77E2D1CA push text:77E2D1CB text:77E2D1D0 RtlEncodePointer call push offset unk_77ECBFD0 Дизассемблерный листинг API-функции SetUnhandledExceptionFilter из Висты. Как видно, со времен W2K ее реализация сильно усложнилась последних версиях Windows появилась шифровка указателей, и BasepCurrentTopLevelFilter хранится в закодированном виде. Естественно, возможность «ручной» работы с указателями никуда не делась, и в NTDLL.DLL появились фун-

воляет нам перехватывать ее путем правки таблицы импорта (или любым другим способом). Конечно, модификация импорта — грязный трюк, привлекающий к себе внимание, поэтому лучше хакнуть непосредственно саму библиотечную функцию обработки исключений. В случае MS VC эта функция носит имя XcptFilter. Первые байты трогать нежелательно — иначе IDA-Pro ее не распознает; впрочем, байт байту рознь. IDA-Pro пропускает относительные вызовы, поскольку они непостоянны и подвержены сезонным вариациям. Нам нужно найти CALL func и заменить func адресом нашей функции my_func, выполняющей некоторые действия и при необходимости возвращающей управление оригиналь-

высокого уровня, интенсивно использует API-функцию UnhandledExceptionFilter, что поз-

Дизассемблерный фрагмент библиотечной функции __XcptFilter

ной func. Анализ кода __XcptFilter обнаруживает вызов _xcptlookup, осуществляемый

в основном блоке кода, то есть не «шунтируемый» никакими ветвлениями, что очень хорошо:

```
= dword ptr 8
.text:00401C9A arg_0
.text:00401C9A ExceptionInfo
                                = dword ptr OCh
.text:00401C9A
.text:00401C9A
                                ebp
                        push
.text:00401C9B
                        mov ebp, esp
.text:00401C9D
                        push
                                ebx
.text:00401C9E
                                [ebp+arg_0]
                        push
.text:00401CA1
                        call
                                xcptlookup; xcptlookup? my invisible seh
.text:00401CA6
                        test
                                eax, eax
```

рительного, поскольку IDA-Pro не проверяет целостность библиотечных функций и никто Скачаню се сайная на вирет в турет в трисое диняйся!

01. Постановка проблемы 02. Перезапись существующего обработчика 03. Прячем FS

main() int *p=0;

lea ecx, souriz

printf("hello, nezumi\n"); ExitProcess(0);

```
mov eax, fs:[0]
lea ecx, souriz
add eax, 4
mov [eax], ecx
```

новится неоднозначным и требует реконструкции последовательности вызываемых функций. Причем обращения к FS в явном виде может и не быть — его значение легко прочитать API-функцией GetThreadContext, на которую, конечно, нетрудно поставить точку останова,

lea ecx, souriz

mov [eax], ecx

add eax, 4

```
Поиск блока окружения потока в стеке
int a; int *p=0;
   unsigned char *pp = (unsigned char*) 0x7FFE0000;
   for(a = 0; a < 6; a++)
```

ние/запись. Размер каждого блока равен 1000h, плюс ко всему указатель на блок окружения OllyDbg - SEH-06.exe <u>File View Debug Plugins Options Window Help</u> 원 원 원 원 원 →: L E M T W H C / K B R ... S <u>≔</u>| 44| ×| C CPU - main thread, module SEH-06 M Memory map

однако все они действуют только против дизассемблеров, а отладчики просто ставят сюда точку останова по доступу, и все обращения к FS:0 немедленно палятся. Независимо от того,

сама по себе представляет проблему для отладчиков, поскольку установленный ею фильтр исключений верхнего уровня при запуске программы под отладчиком не выполняется и приходится использовать разнообразные плагины для «Ольги», чтобы заставить систему считать, что никакого отладчика здесь нет. Или же, как вариант, насильственно включать фильтр верхнего уровня в цепочку обработчиков структурных исключений.

кции RtlEncodePointer и RtlDecodePointer, имена которых говорят сами за себя. Все это существенно усложняет реализацию защиты и делает ее экономически нецелесообразной, вынуждая искать другие пути. И такие пути действительно есть! Библиотечный обработчик структурных исключений, поставляемый вместе с языками

.text:00401C9A XcptFilter proc near .text:00401C9A

Обнаружить наш обработчик исключений практически невозможно. Он отсутствует в SEHцепочке (точнее, присутствует, но прячется внутри обработчика, устанавливаемого RTL языка высокого уровня), и «Ольга» в упор его не видит. Конечно, при пошаговой трассировке хакерский обработчик будет выявлен — вот только трассировать мегабайты системного и библиотечного кода никто не будет. Дизассемблирование также не покажет ничего подоз-