Баранов Артем artembaranovex@gmail.com

КРИМИНАЛИСТИЧЕСКИЙ АНАЛИЗ ПАМЯТИ

Исследуем процессы в Windows 7

Привет, мой дорогой читатель! Сегодня я познакомлю тебя с продвинутым способом детекта скрытых модулей процессов. Ты уже знаешь, что большинство руткитов скрывают свои модули широко известным способом — через удаление их из PEB. Но вот способ, с помощью которого такие скрытые модули определяются, и которым пользуются антируткиты, недокументирован и мало где описан. Сейчас я внесу свой светлый луч в это темное царство. Но для понимания этой статьи от тебя понадобятся знания в области внутреннего устройства ядра.

СУТЬ ПРОБЛЕМЫ

Всем известно, что Windows очень гибко управляет распределением физической памяти. Она выдается процессу только тогда, когда он к ней реально обратится (исключение составляет лишь #PF). В момент такого обращения диспетчер памяти (или VMM) должен различить ситуацию нарушения доступа (обращения к участку памяти, который не был зарезервирован, или под него не выделена физическая память) от ситуации, когда память передана, но физически еще не выделена. VMM полагается на структуры дескрипторов виртуальных адресов (Virtual Address Descriptor), которые организованы в дерево на основе номеров страниц.

Общая схема работы VAD есть у Руссиновича в главе про диспетчер виртуальной памяти. Детальную же информацию по структурам нам может дать только windbg. Суть VAD'ов в том, что они помогают обнаруживать библиотеки, загруженные в память процессов, которые скрывают руткиты (например, давно известным способом удаления из PEB). Когда я впервые заинтересовался структурой VAD для обнаружения скрытых dll, мне на глаза попалась замечательная статья журнала Digital Investigation — «The VAD tree: A process-eye view of physical memory» (dfrws.org/2007/proceedings/p62-dolan-gavitt.pdf). Материал дает подробную информацию об устройстве VAD, но, к сожалению, вопрос лишен практической стороны. Обобщая эту статью и добавляя

По команде !process можно получить информацию о Vad

практическую часть, я наглядно объясню, как с помощью анализа VAD антируткиты показывают список загруженных в процессы DLL.

ЧТО ТАКОЕ VAD, И С ЧЕМ ЕГО ЕДЯТ

Условно, VAD — это структура, которая описывает регион адресного пространства. Например, при вызове функции VirtualAlloc с параметром MEM_RESERVE резервируется регион требуемого размера и создается VAD, описывающий этот регион. При передаче физической памяти параметра MEM_COMMIT система пометит в этой структуре количество переданных страниц. Другой пример: система загружает DLL в адресное пространство процесса, соответственно, это приводит к резервированию региона памяти, и тогда создается VAD, который описывает данный регион. Принцип один, разница лишь в том, какая структура VAD будет использоваться для того или иного случая резервирования.

Внутри себя ядро использует функцию MiCheckVirtualAddress, которая первым аргументом принимает виртуальный адрес, а возвращает указатель на соответствующий этому адресу PTE. При этом в третий аргумент записывается указатель на соответствующий адрес VAD.

Ее структура выглядит так:

```
MiCheckVirtualAddress (
    IN PVOID VirtualAddress,
    OUT PVOID Unknown,
    OUT PMMVAD *VadOut
    )
```

VAD БЫВАЕТ РАЗНЫМ

Видов VAD бывает несколько: _MMVAD_SHORT, _MMVAD и _MMVAD_LONG. Причем каждый последующий фактически расширяет предыдущий. Шапка у всех одна и выглядит так (см. также _MMADDRESS_NODE):

```
typedef struct _MMVAD_SHORT
{
   union
   {
      LONG32 Balance : 2;
      struct _MMVAD* Parent; // родительский VAD
   } u1;
   struct _MMVAD* LeftChild; // левый дочерний VAD
   struct _MMVAD* RightChild; // правый дочерний VAD
   ULONG32 StartingVpn; // стартовый номер страниц
   ULONG32 EndingVpn; // последняя страница, номер
   union
   {
      ULONG32 LongFlags;
      struct _MMVAD_FLAGS VadFlags; //полезные флаги
   } u;
   …
} MMVAD_SHORT, *PMMVAD_SHORT.
```

```
typedef struct _MMVAD_FLAGS
{
   ULONG32 CommitCharge : 19; // 0 BitPosition
   ULONG32 NoChange : 1;      // 19 BitPosition
   ULONG32 VadType : 3;       // 20 BitPosition
   ULONG32 MemCommit : 1;     // 23 BitPosition
   ULONG32 Protection : 5;    // 24 BitPosition
   ULONG32 Spare : 2;         // 29 BitPosition
   ULONG32 PrivateMemory : 1; // 31 BitPosition
}MMVAD_FLAGS, *PMMVAD_FLAGS;
```

CommitCharge — количество выделенных (COMMIT) страниц в этом узле.
VadType — тип Vad.
Protection — атрибут защиты страниц.
MMVAD_SHORT не подойдет для наших целей, так как с его помощью описываются обычные приватные страницы, выделенные, например, с помощью VitualAlloc. А вот _MMVAD — как раз то, что нужно.

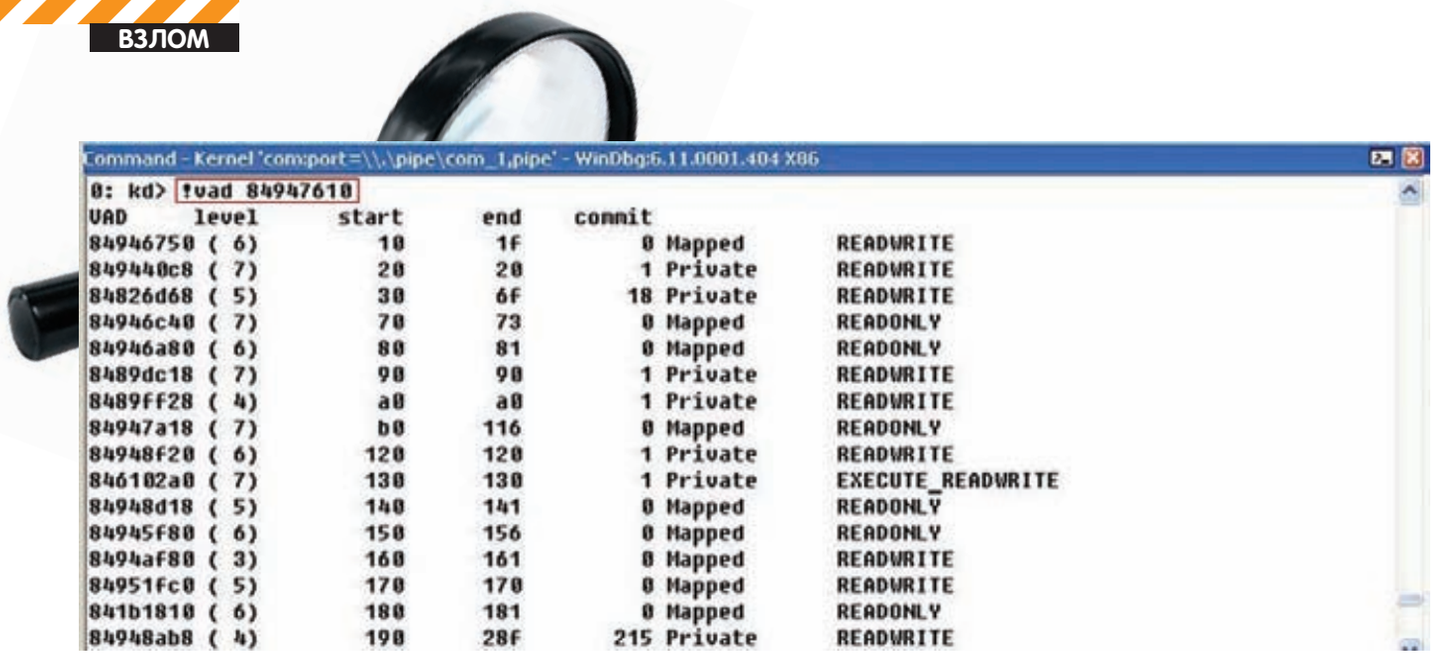
```
typedef struct _MMVAD
{
   union
   {
      LONG32 Balance : 2;
      struct _MMVAD* Parent;
   } u1;
   struct _MMVAD* LeftChild;
   struct _MMVAD* RightChild;
   ULONG32      StartingVpn;
   ULONG32      EndingVpn;
   union
   {
      ULONG32 LongFlags;
      struct _MMVAD_FLAGS VadFlags;
   }u;
   …
   //далее данные, которые связывают VAD с секцией раздела
   union
   {
      struct _SUBSECTION* Subsection;
      struct _MSUBSECTION* MappedSubsection;
   };
   struct _MMPTE* FirstPrototypePte;
   struct _MMPTE* LastContiguousPte;
   struct _LIST_ENTRY ViewLinks;
   struct _EPROCESS* VadsProcess;
} MMVAD, *PMMVAD;
```

А так выглядит подраздел.

```
typedef struct _SUBSECTION
{
   // указатель на область управления
   PCONTROL_AREA ControlArea;

   // далее следуют поля, необходимые VMM для сопостав-
ления образа на диске и в памяти для подкачки

   union
   {
      ULONG LongFlags;
      MMSUBSECTION_FLAGS SubsectionFlags;
   } u;
   …
} SUBSECTION, *PSUBSECTION;
```



Команда !vad дампит содержимое дерева

Версия NT	Смещение поля VadRoot
2195 (2k)	0x194
2600 (XP)	0x11C
3790 (2k3)	0x250
6000 (Vista)	0x238
7100 (Seven RC)	0x274
7600 (Seven RTM)	0x278

Через область управления разделом мы можем выйти на FILE_OBJECT, который специально создан для проекции файла.

```
typedef struct _CONTROL_AREA
{
   PSEGMENT Segment;
   …
   struct _EX_FAST_REF FilePointer;
   …
}CONTROL_AREA, *PCONTROL_AREA;
```

При получении указателя на FileObject мы зануляем первые три бита FilePointer маской 0xFFFFFFF8.
Указатель на корневой узел хранится в EPROCESS в поле VadRoot, смещение которого меняется от версии к версии, поэтому неплохо было бы захардкодить его под разные версии.
Как видно из структуры _VAD, первые два бита адреса Parent, используются для служебных целей, поэтому для получения валидного адреса их также нужно занулять. Поле применяется для получения указателя на VAD верхнего уровня.

```
VadRoot = *(PULONG )( (PUCHAR )Eprocess +
   EPROCESS_VadRoot_Offs ) & 0xFFFFFFFC;
```

Формат VadRoot в EPROCESS различен в разных версиях NT. В Windows 7 он представляет собой структуру MM_AVL_TABLE (очевидно, разработчики добавили информацию о балансировке дерева).

```
typedef struct _MM_AVL_TABLE {
   struct _MMADDRESS_NODE BalancedRoot;
   struct {
      ULONG32 DepthOfTree : 5;
      ULONG32 Unused : 3;
      ULONG32 NumberGenericTableElements : 24;
   };
   VOID* NodeHint;
   VOID* NodeFreeHint;
}MM_AVL_TABLE, *PMM_AVL_TABLE;
```

Некоторая сложность заключается в том, что BalancedRoot.u1.Parent не является истинной вершиной дерева, для этого нужно анализировать RightChild, но его можно использовать для начала обхода дерева, потому что он указывает сам на себя.
При прохождении по дереву VAD нужно отделять MMVAD_SHORT от MMVAD и MMVAD_LONG. Это можно делать по тэгу блока пула, которому принадлежит структура. Собственно тэг хранится по смещению 4 от самого блока (и по обратному смещению -4 от начала структуры). Диспетчер памяти присваивает тэг «VadS» для _MMVAD_SHORT, «Vad» для _MMVAD, «Vadl» для _MMVAD_LONG. Кроме того, разумеется, нужно проводить валидацию промежуточных параметров типа ControlArea, Subsection, FilePointer.

ПРАКТИКУЕМСЯ

Проведем исследование вадов. Выберем из списка процессов (!process 0 0) некий процесс:

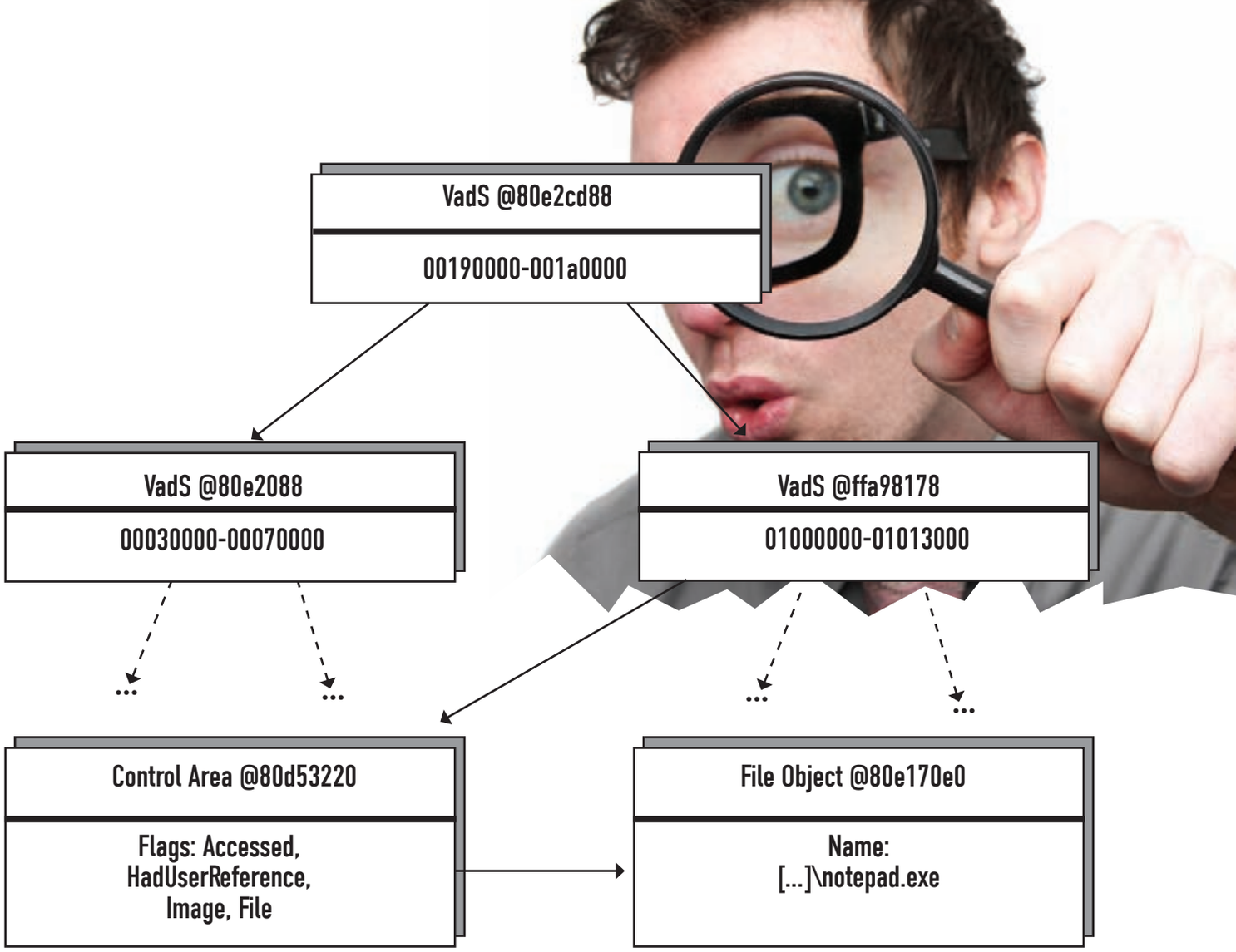
```
kd> !process 84944418  0
PROCESS 84944418  SessionId: 1  Cid: 0a40    Peb:
7ffdf000  ParentCid: 08e4
    DirBase: 3ec0c420  ObjectTable: 993f0830
HandleCount: 256.
    Image: TOTALCMD.EXE
```

Адрес корня дерева:

```
kd> dt _MM_AVL_TABLE 84944418+278 -r2
nt!_MM_AVL_TABLE
   +0x000 BalancedRoot: _MMADDRESS_NODE
      +0x000 u1 : <unnamed-tag>
         +0x000 Balance  : 0y00
         +0x000 Parent: 0x84944690 указывает в себя
(см. ниже)
      +0x004 LeftChild : (null)
      +0x008 RightChild  : 0x84947610 истинный
адрес VadRoot
…
kd> dd 84944418+278 l1
84944690  84944690
```

Возьмем первый VAD:

```
kd> dc 0x84947610-4 l1
```



Часть дерева VAD для notepad.exe

```
8494760c  20646156    Vad
```

Это MMVAD.

```
kd> dt _MMVAD 0x84947610
nt!_MMVAD
   +0x000 u1               : <unnamed-tag>
   +0x004 LeftChild        : 0x84949290 _MMVAD
   +0x008 RightChild       : 0x84941b48 _MMVAD
   +0x00c StartingVpn      : 0x703b0
   +0x010 EndingVpn        : 0x703e1
...
   +0x024 Subsection       : 0x84820110 _SUBSECTION
   +0x024 MappedSubsection : 0x84820110 _MSUBSECTION
   +0x028 FirstPrototypePte : 0x98f1ece0 _MMPTE
   +0x02c LastContiguousPte : 0xfffffffc _MMPTE
   +0x030 ViewLinks        : _LIST_ENTRY [ 0x848e3618 -
0x84820108 ]
   +0x038 VadsProcess      : 0x84944419 _EPROCESS
```

VAD описывает регион 0x703b0000-0x703e1000 включительно, а также имеет указатель в подраздел 0x84820110 (секция раздела, в случае с EXE-файлами — подраздел на каждую секцию).

```
kd> dt _subsection ControlArea 0x84820110
nt!_SUBSECTION
   +0x000 ControlArea : 0x848200c0 _CONTROL_AREA

kd> dt _control_area 0x848200c0 -r1
nt!_CONTROL_AREA
   +0x000 Segment          : 0x98f1ecb0 _SEGMENT
...
   +0x024 FilePointer      : _EX_FAST_REF
      +0x000 Object           : 0x88cbc79a
      +0x000 RefCnt           : 0y010
      +0x000 Value            : 0x88cbc79a
```

Несложным расчетом определяем, что объект: 0x88cbc79a & 0xFFFFFFF8 = 0x88CBC798.

```
kd> dt _file_object filename 88CBC798
nt!_FILE_OBJECT

   +0x030 FileName : _UNICODE_STRING "\Windows\
System32\winmm.dll"
```

В итоге мы получили информацию о том, что VAD описывает спроецированную winmm.dll и тем самым вышли на конкретный файл!

Обрати внимание, что в Windows 2000/XP не нужно выравнивать указатель при получении адреса на VadRoot, потому что там он имеет вид PVOID VadRoot.

Дерево VAD доступно только в «живых» или запущенных процессах. Ядро зануляет указатель на вершину дерева с завершением процесса. Это ограничивает полезность данного метода для изучения некоторых процессов.

На этом пока все. Изучай структуры, экспериментируй, и ты выявишь абсолютно любой руткит, даже базируясь на моем примере. Если есть вопросы — пиши на почту, и я всегда помогу.