Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование
информационных систем

Системное программирование

Тарасова Полина Максимовна

Восстановление виртуального адресного пространства процесса в Windows 10

Выпускная квалификационная работа

Научный руководитель:
ст. преп. Губанов Ю.А.

Научный консультант:
системный архитектор Тимофеев Н.М.

Рецензент:
разработчик Мухаматулин М. С.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

Software Engineering

Tarasova Polina

Reconstruction of process address space in Windows 10

Graduation Project

Scientific supervisor:
Senior lecturer Gubanov Y.A.

Scientific consultant:
Systems architect Timofeev N.M.

Reviewer:
Software Developer Muhamatulin M.S.

Saint-Petersburg
2017

Введение

В современном мире компьютерные и информационные технологии непрерывно развиваются: появляются новые гаджеты, социальные сети, платежные системы, и многое другое, а вместе с этим, прогрессируют и средства хищения пользовательской информации.

В расследовании преступлений такого рода помогает наука — компьютерная криминалистика. Компьютерная криминалистика — это ветвь криминалистики, сфокусированная на нахождении доказательств в компьютерах и других средствах хранения цифровой информации. Она включает в себя изучение цифровых носителей (данных) с целью выявления, сохранения, восстановления, анализа и представления фактов о цифровой информации.

В случае, когда устройство остается во включенном состоянии, существует возможность получить информацию, хранящуюся в оперативной памяти: запущенные процессы, образы программ (в частности, вредоносных), сетевые соединения, удалённые файлы и многое другое.

Для обработки такой информации, необходимо осуществить снятие образа (дампа) оперативной памяти, который будет содержать в себе данные и процессы, активные в момент создания дампа. Каждый процесс имеет своё адресное пространство, восстановив которое, можно провести более точный анализ.

Данная работа является продолжением работы Антона Овчинникова ([6]), в которой была реализована возможность восстановления адресного

пространства процессов для семейств ОС Windows 7 x86 и Windows 7 x86_64.

На данный момент широкое распространение получила Windows 10. Актуальность работы состоит в том, что структура оперативной памяти этой версии операционной системы сильно отличается от предыдущих версий Windows и на данный момент мало изучена, а литература по ней отсутствует.

1 Постановка задачи

Целью работы является разработка и внедрение подсистемы для восстановления виртуального адресного пространства процесса приложения из образа оперативной памяти Windows 10. Для достижения цели необходимо было решить следующие задачи:

- Изучить структуры памяти Windows 10
- Разработать метод восстановления адресного пространства процесса Windows 10
- Реализовать предложенный метод в продукте Belkasoft Evidence Center
- Провести тестирование

2 Обзор

2.1 Структура данных EPROCESS

В Windows каждый процесс описывается структурой данных EPROCESS (Executive Process Block) [1]. Эта структура содержит множество атрибутов, связанных с рассматриваемым процессом, а также ссылки на ряд других связанных с ним структур. Так, у каждого процесса есть один или несколько потоков, которые представляются структурой ETHREAD (Executive Thread Block), а в подструктуре KPROCESS (kernel process — процесс ядра) хранятся данные, необходимые для планирования потоков.

Для того, чтобы более подробно ознакомиться со структурой EPROCESS, можно воспользоваться командой dt nt! _epcosess в отладчике WinDbg.

```
lkd> dt nt!_eprocess
   +0x000 Pcb               : _KPROCESS
   +0x0a8 ProcessLock       : _EX_PUSH_LOCK
   +0x0ac RundownProtect    : _EX_RUNDOWN_REF
   +0x0b0 VdmObjects        : Ptr32 Void
   +0x0b4 UniqueProcessId   : Ptr32 Void
   +0x0b8 ActiveProcessLinks : _LIST_ENTRY
...
   +0x11c Win32Process      : Ptr32 Void
   +0x120 Job               : Ptr32 _EJOB
...
   +0x140 OwnerProcessId    : Uint4B
   +0x144 Peb               : Ptr32 _PEB
   +0x148 Session           : Ptr32 _MM_SESSION_SPACE
...
   +0x28c VadRoot           : _RTL_AVL_TREE
   +0x290 VadHint           : Ptr32 Void
   +0x294 VadCount          : Uint4B
   +0x298 VadPhysicalPages  : Uint4B
   +0x29c VadPhysicalPagesLimit : Uint4B
...
   +0x330 SequenceNumber    : Uint8B
   +0x338 CreateInterruptTime : Uint8B
   +0x340 CreateUnbiasedInterruptTime : Uint8B
```

Рис. 1 Частичный вывод на 32-битной системе Windows 10: некоторые поля структуры EPROCESS и их смещения.

Таким образом, для корректного анализа образа памяти, необходим метод поиска структур EPROCESS.

2.2 Метод списков

В операционной системе Windows структуры EPROCESS организованы в круговой двусвязный список [2]. EPROCESS содержит в себе поле ActiveProcessLinks, структура которого, LIST_ENTRY, имеет следующий вид:

```
typedef struct _LIST_ENTRY {
 struct _LIST_ENTRY *Flink;
 struct _LIST_ENTRY *Blink;
}
```

В ней хранятся указатели на следующий (FLINK) и предыдущий (BLINK) процессы в списке относительно текущего. Для того чтобы найти интересующий процесс, необходимо получить хотя бы один элемент этого списка.

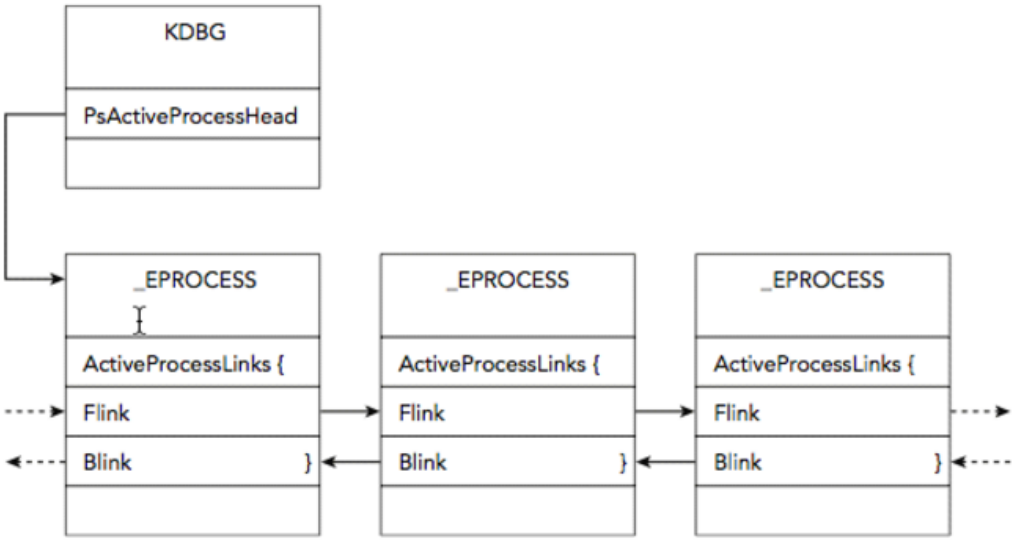


Рис. 2 Список процессов [8]

Глобальная переменная PsActiveProcessHead — это указатель на голову двусвязного списка, который есть в любой структуре EPROCESS. Для того, чтобы его получить, необходимо найти адрес какого-то блока EPROCESS из списка активных процессов. Достаточно найти два таких блока и убедиться, что они ссылаются друг на друга. Для этого отлично подходят системные процессы smss.exe и crss.exe. Обычно (Windows 2000 / XP / 2003) поле FLINK одного процесса (smss.exe) указывает на поле FLINK другого процесса (csrss.exe) этого же списка.

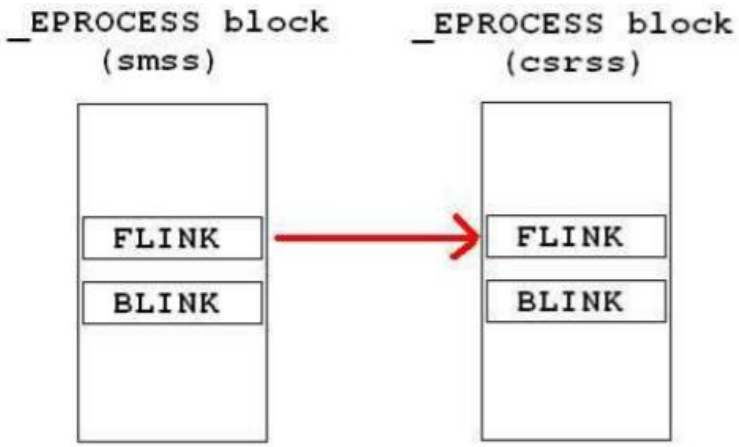


Рис. 3 smss.exe ссылается на csrss.exe (из [2])

В работе [5] говорится, что в Windows Vista, и более поздних версиях, может быть больше одного процесса crss.exe. Кроме того, если в системе зарегистрировано несколько учетных записей, или активен протокол RDP (Remote Desktop Protocol — протокол удалённого рабочего стола), или открыта терминальная сессия, точно существует несколько копий csrss.exe. В этих случаях, необходимо из всех экземпляров csrss.exe выделить требуемый экземпляр процесса, что является нетривиальной задачей, так как необходимо разобрать таблицы дескрипторов и списки памяти для всех экземпляров csrss.exe.

2.3 Сигнатурный метод

Сигнатурный метод [6] заключается в поиске смещений некоторых структур в физической памяти и в дальнейшем сопоставлении их с заранее заданными шаблонами, таким образом определяется принадлежность к определенной структуре данных. Так, KPROCESS начинается с подструктуры DISPATCHER_HEADER, поля которой, Type и Size, должны быть одинаковыми для всех процессов операционной системы. Но совпадение смещений, конечно, не дает гарантию того, что найдена именно структура EPROCESS. Необходимо проверить, что следующий (ThreadListHead.Flink) и предыдущий (ThreadListHead.Blink) потоки процесса принадлежат виртуальному адресному пространству ядра. Подробное исследование сигнатурного метода на примере ОС Windows XP приводится в [3].

Главная проблема этого метода в том, что при обновлении операционной системы значения и смещения полей в структуре EPROCESS могут измениться.

Так как в базовом механизме сигнатурного поиска нет отличий от работы [6], он будет переиспользован в данной работе.

2.4 KPCR-метод

KPCR (Kernel Processor Control Region) — это внутренняя структура данных, где ядро хранит специфические для процессора данные, которые могут быть просмотрены с помощью отладчика. В более ранних версиях для поиска этой структуры существовали неизменяющиеся сигнатуры. Так, виртуальный адрес структуры имел значение 0xFFDFF00. Далее, по смещению 0x1c от начала структуры хранилась ссылка на саму эту структуру, а по смещению 0x20 — на структуру KPCRB. Структуру EPROCESS, связанную с текущим запущенным процессом, можно обнаружить с помощью KPCRB (Kernel Processor Control Block — блок управления процессором), эта структура — расширение KPCR, в ней содержится ссылка на текущий ETHREAD, которая, в свою очередь, содержит ссылку на текущий EPROCESS ([4]).

Но в версиях, начиная с Windows 8, виртуальный адрес структуры KPCR стал динамическим (с использованием технологии Address Space Layout Randomization (ASLR)), что существенно замедляет поиск данной структуры, более того поле KdVersionBlock структуры KPCR, содержащее указатель на PsActiveProcessHead, в Windows 10 имеет значение “null”, и его никак нельзя использовать для поиска списка процессов.

2.5 Существующие решения: Volatility Framework, PTFinder

Volatility Framework

Volatility Framework — фреймворк, реализованная на языке Python, с открытым исходным кодом, предоставляющий набор инструментов для извлечения цифровых артефактов из оперативной памяти. Эта платформа корректно разбирает дампы памяти, созданные на различных операционных системах (Windows, OS X, Linux), при этом предоставляя полную информацию о памяти исследуемой системы.

Для корректного анализа образа памяти в качестве одного из аргумента необходимо передать название плагина, с помощью которого будет проведен разбор. Плагин — отдельный модуль, который выбирается в зависимости от того, какую информацию требуется получить. Благодаря такой организации работы, Volatility Framework имеет расширяемую архитектуру.

Данный фреймворк поддерживает сигнатурный поиск структур и поиск структур на основе отладочной информации (KDBG) и управляющих структур (KPCR). Поиск на основе KDBG (метод списков) поддерживается только для платформы Windows.

Также, в аргументы требуется передавать еще версию и битность исследуемой операционной системы. Такое усложнение является существенным минусом данной утилиты, так как требует от пользователя дополнительных усилий. Если пользователь по каким-либо причинам не имеет этой информации, можно воспользоваться следующей командой:

```
$ python vol.py -f memory.raw kdbgscan
```

вывод которой содержит необходимую информацию: версию системы, ее битность и другие дополнительные сведения.

Volatility Framework предоставляет поддержку образов памяти следующих версий Windows: XP, 2003 Server, Vista, 2008 Server, 7, 8, 10.

PTFinder

PTFinder (Process and Thread Finder) — утилита с закрытым исходным кодом, написанная на скрипте Perl и поддерживающая анализ только ОС Windows (2000/2003/XP/XP SP2/Vista). С ее помощью можно найти список процессов и потоков в образе памяти. При поиске используется сигнатурный метод на основе DISPATCHER_HEADER и некоторых других дополнительных проверок, о которых рассказывается в [3]. Последняя версия PTFinder-а была выпущена в 2007-ом году, и, по всей видимости, инструмент больше не поддерживается, а значит, не поддерживает Windows 10.

Для анализа памяти Windows 10, исследователям облегчил бы работу инструмент, поддерживающий автоматическое определение версии и битности операционной системы.

В силу этих причин было принято решение взять за основу дипломную работу Антона Овчинникова ([6]), в которой была осуществлена поддержка семейств ОС Windows 7 x86 и Windows 7 x86_64.

2.4 Обзор механизма виртуальной памяти в архитектуре x86 PAE

В x86-совместимых процессорах используются 32-разрядные физические адреса, что позволяет каждому процессу адресовать до 2^{32} = 4 Гбайт оперативной памяти. Однако, существует механизм PAE (Physical Address Extension), который позволяет адресовать до 2^{36} = 64 Гбайт оперативной памяти.

Виртуальное адресное пространство делится на части, называемые страницами. Страницы бывают двух размеров — малые (small) и большие, размером 4 Кбайта и 2 Мбайта каждая соответственно. Фактические размеры зависят от архитектуры процессора и представлены ниже.

Архитектура	Размер малых страниц (Кбайт)	Размер больших страниц (Мбайт)	Количество малых страниц в большой странице
x86	4	4 (2 с поддержкой PAE)	1024 (512 с поддержкой PAE)
x86	4	2	512

Рис. 4 Размеры страниц виртуального адресного пространства

В режиме PAE преобразование 32-битного виртуального адреса происходит с помощью трех таблиц: таблица указателей на каталог страниц PDPT (Page Directory Pointer Table), таблица каталога страниц PDT (Page Directory Table) и таблица страниц PT (Page Table).

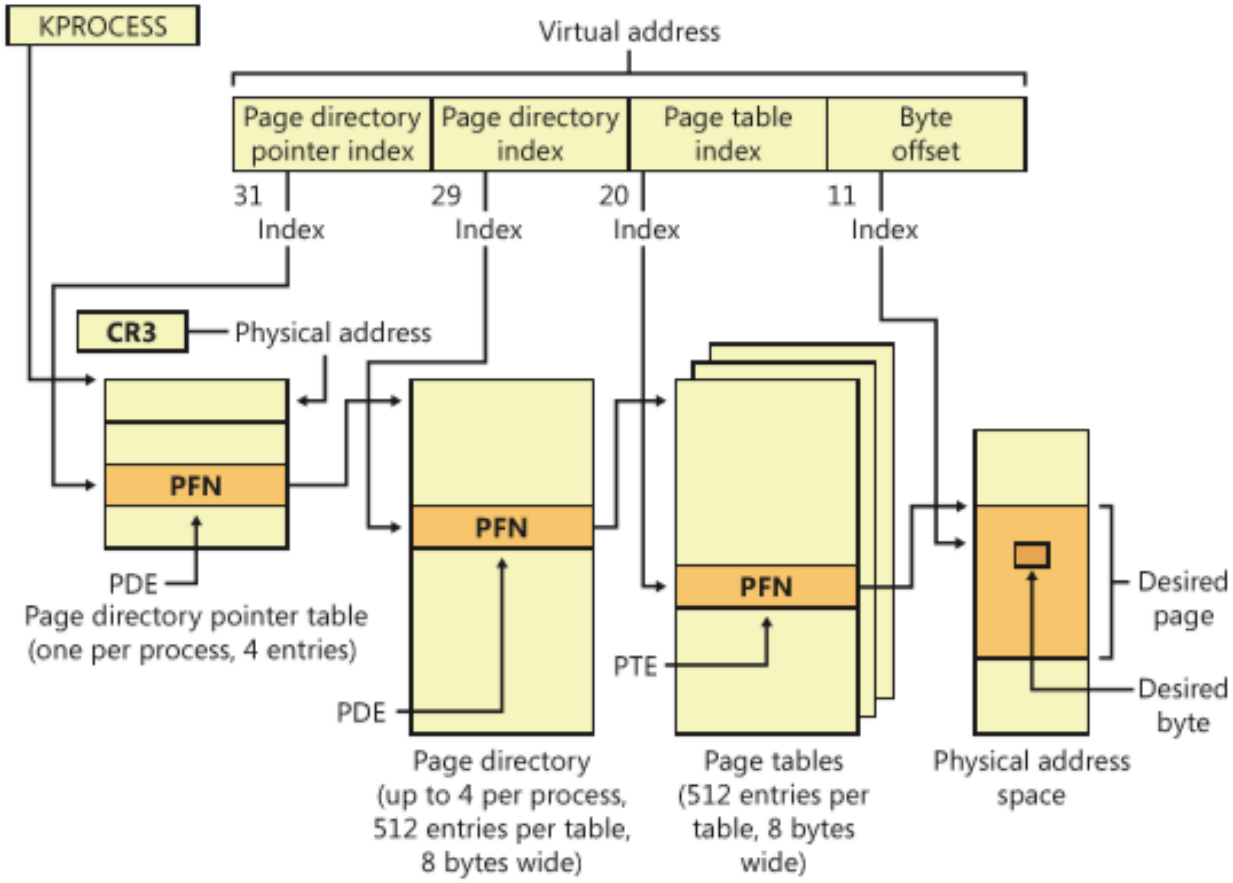


Рис. 5 Страничная адресация в режиме PAE [7]

Трансляция адресов:

Виртуальный адрес разбивается на 4 компоненты: индекс в таблице указателей на каталог страниц (PDPI), индекс в таблице каталогов страниц (PDI), индекс в таблице страниц (PTI) и смещение в странице.

1) Первая таблица, используемая для виртуальной адресации в режиме PAE — таблица указателей на каталог страниц (PDPT, Page Directory Pointer Table). Во время исполнения процесса в управляющий регистр CR3 загружен физический адрес таблицы указателей на каталог страниц. Индекс, полученный из первой компоненты виртуального адреса,

применяется в качестве индекса указателя на каталог страниц (PDPI, Page Directory Pointer Index), с его помощью находится запись указателя на каталог страниц (PDPE, Page Directory Pointer Entry).

2) Из PDPE выделяется база (физический адрес элемента следующего уровня) соответствующего каталога страниц (PD).
3) С помощью индекса, полученного из второй компоненты виртуального адреса, (Page Directory Index, PDI) из каталога страниц берется запись (PDE, Page Directory Entry).
4) Из PDE выделяется база таблицы страниц (PT). В случае, если бит 7 рассматриваемой записи выставлен в единицу, из PDE выделяется база большой страницы и далее пункт 7).
5) С помощью индекса, полученного из третьей компоненты виртуального адреса, (Page Table Index, PTI) используется запись из таблицы страниц (PTE, Page Table Entry).
6) Из PTE выделяется база соответствующей страницы.
7) С помощью четвёртой компоненты виртуального адреса, смещения, осуществляется адресация внутри найденной страницы.

3. Предлагаемый метод восстановления виртуального адресного пространства процесса

Для получения виртуального адресного пространства процесса необходимо выполнить обход табличных структур в глубину и выполнить запись в файл страниц памяти, на которые ссылаются действительные записи. Действительная запись – это запись, младший бит которой (бит V) выставлен в единицу, в противном случае запись называется недействительной и требует более тщательного изучения.

3.1 Обработка недействительных записей

Если бит V выставлен в ноль, это не означает, что данные, на которые указывает запись, не требуют дальнейшего рассмотрения. Возможно, данные никогда не загружались в память, и поэтому система считает, что они недоступны. Несмотря на это, каждая такая PDE или PTE относится к одной из категорий недействительных записей: Pagefile, Demand Zero, Transition, Prototype или Zero

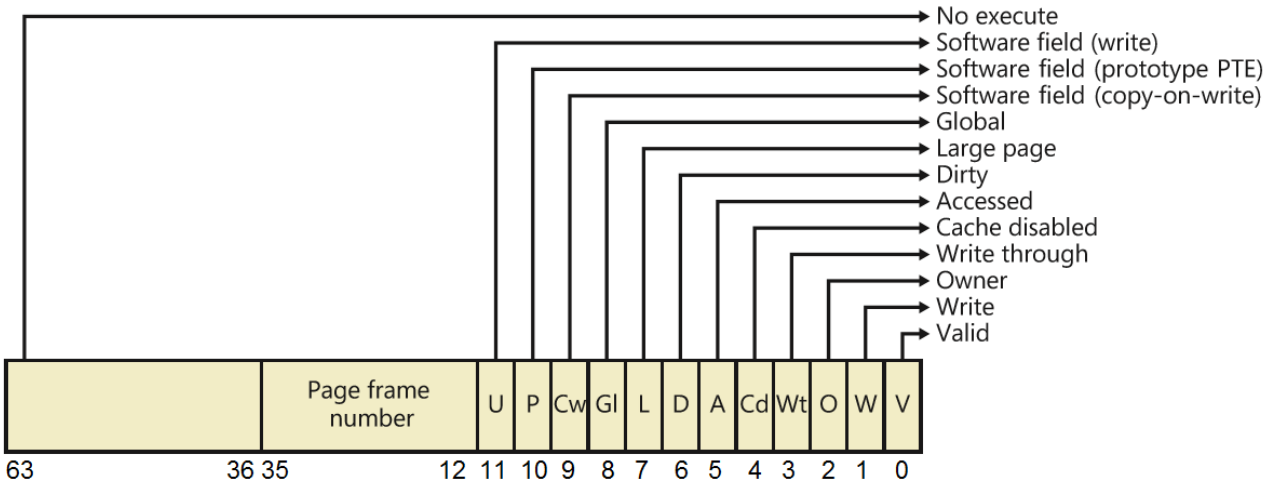


Рис. 6 PTE архитектуре X86_PAE из [7]

Pagefile

Требуемая страница находится в файле подкачки. В [7] рассказывается, что младшие четыре бита записи таблицы страниц показывают, в какой именно из 16 возможных файлов подкачки находится требуемая страница, а 24 бита (с 12 по 35) (в случае 32-битной системы без поддержки режима расширения физических адресов — 20 бит (с 12 по 31)) снабжают номером страницы в файле. Смещение в файле подкачки всегда отлично от нуля и никогда не состоит только из единичных битов (иначе говоря, первая и последняя страницы из файла подкачки не используются для самой подкачки), это позволяет существовать другим типам недействительных записей (см. далее).

Demand Zero PTE

Данный тип аналогичен PageFile, но поле смещения в файле подкачки состоит только из нулей. При запросах к странице с PT-записью данного типа операционная система должна возвращать страницу, состоящую только из нулей. В случае обращения к записи данного типа операционная система возвращает страницу, полностью состоящую из нулей.

Transition

В работе [8] говорится, что, если одиннадцатый бит в записи равен единице и десятый бит (определяющий запись как прототипную) равен нулю, страница считается находящейся в состоянии “перехода”. Это означает, что страница была изменена, но пока не записана обратно на диск. Также, не стоит забывать, что большие страницы также могут быть в состоянии “перехода”. Несмотря на то, что страница находится в

состоянии “перехода”, она все еще находилась в активной памяти и поэтому может быть извлечена.

Zero PTE

Запись таблицы страниц полностью состоит из нулей. Такие записи никак не обрабатываются, в силу своей бессодержательности.

Prototype

Данные типы записей требуют более сложной обработки: необходимо построение VAD-дерева (Virtual Address Descriptors tree). Однако, в Windows 10 изменились структуры, отвечающие за поля этого дерева, поэтому дальнейшее развитие работы может состоять в отыскании способов построения VAD-дерева с помощью других структур.

3.2 Получение адресного пространства процесса

Рассмотрим алгоритм получения виртуального адресного пространства процесса для 32-битной системы Windows 10 с использованием механизма расширения физических адресов (Physical Address Extention).

Необходимо определить процесс, адресное пространство которого требуется получить. Определение происходит с помощью выбора нужной структуры EPROCESS, полученной с помощью сигнатурного метода.

Далее нужно получить смещение от начала файла образа (базу, физический адреса) таблицы PDPT. Осуществить это можно, узнав значение, содержащееся в регистре CR3 — это и будет указателем на базу первой таблицы, в случае данной архитектуры, на PDPT (значение

регистра CR3 можно посмотреть с помощью структуры EPROCESS в поле DirectoryTableBase).

Перебрать все PDPT-записи. Если PDPT-запись — действительная, то из нее можно извлечь физический адрес соответствующей таблицы PD. Про недействительные записи и их обработку было сказано ранее.

После получения физического адреса таблицы страниц в образе оперативной памяти, аналогичным образом перебираются все записи PDE, а затем и PT-записи.

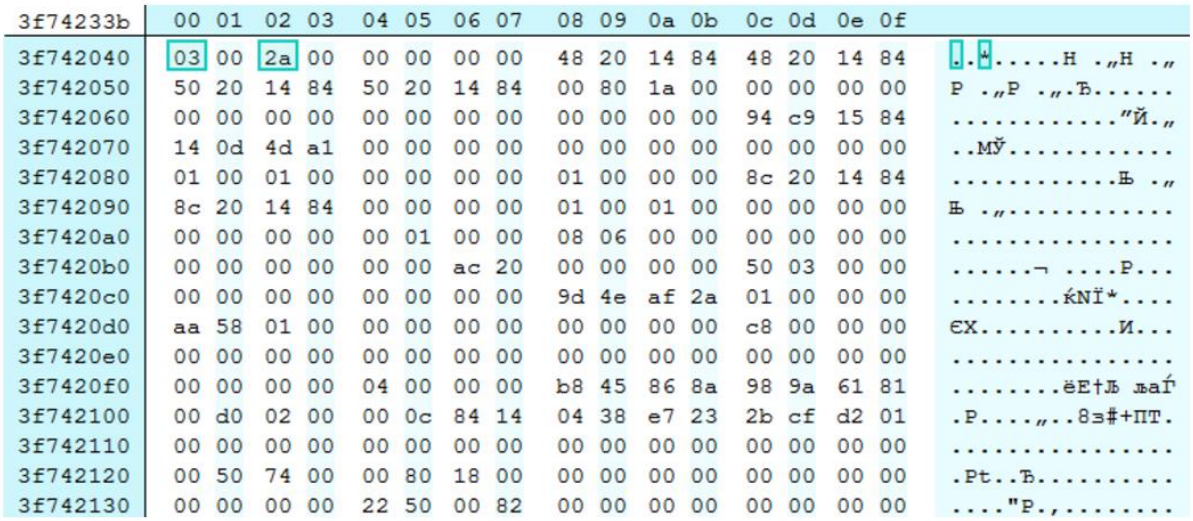
Если PT-запись — действительная, то из нее можно получить физический адрес соответствующей страницы, после чего записать страницу в выходной поток.

4 Реализация и интеграция

4.1 Поиск структур EPROCESS в ОС Windows 10

Для анализа оперативной памяти необходимо осуществить снятие образа оперативной памяти. Для этой цели использовалась утилита Belkasoft Live RAM Capturer [9], которая сохраняет снимок памяти в файл, который можно анализировать.

В упомянутой выше дипломной работе ([6]) для поиска структур EPROCESS использовался сигнатурный метод. Для начала было необходимо изучить поля Type и Size структуры DISPATCHER_HEADER. Было неочевидным, что они будут совпадать у всех процессов рассматриваемой операционной системы. Для этого с помощью WinDbg были рассмотрены несколько процессов и с помощью команды dt nt!_eprocess <procaddress> найдены смещения этих полей для нескольких процессов. Затем, посмотреть дамп памяти с использованием утилиты HexEditor, найдя там соответствующие адреса, и сравнив полученные строки. Таким образом удалось выяснить, что значения этих полей одинаковы для всех процессов Windows 10 и составляют 0x03 и 0x2a соответственно. Так как данная сигнатура подходит для идентификации структур EPROCESS, было решено использовать ее в предложенном методе.



3f74233b	00 01 02 03	04 05 06 07	08 09 0a 0b	0c 0d 0e 0f	
3f742040	03 00 2a 00	00 00 00 00	48 20 14 84	48 20 14 84	..*.....H .„H .„
3f742050	50 20 14 84	50 20 14 84	00 80 1a 00	00 00 00 00	P .„P .„.Ъ.....
3f742060	00 00 00 00	00 00 00 00	00 00 00 00	94 c9 15 84	............"Й.„
3f742070	14 0d 4d a1	00 00 00 00	00 00 00 00	00 00 00 00	..МЎ............
3f742080	01 00 01 00	00 00 00 00	01 00 00 00	8c 20 14 84	............Њ .„
3f742090	8c 20 14 84	00 00 00 00	01 00 01 00	00 00 00 00	Њ .„............
3f7420a0	00 00 00 00	00 01 00 00	08 06 00 00	00 00 00 00	....
3f7420b0	00 00 00 00	00 00 ac 20	00 00 00 00	50 03 00 00	......¬ ....P...
3f7420c0	00 00 00 00	00 00 00 00	9d 4e af 2a	01 00 00 00	........ќNЇ*....
3f7420d0	aa 58 01 00	00 00 00 00	00 00 00 00	c8 00 00 00	єX..........И...
3f7420e0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	....
3f7420f0	00 00 00 00	04 00 00 00	b8 45 86 8a	98 9a 61 81	........ёЕ†Љ љaЃ
3f742100	00 d0 02 00	00 0c 84 14	04 38 e7 23	2b cf d2 01	.Р....„..8з#+ПТ.
3f742110	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	....
3f742120	00 50 74 00	00 80 18 00	00 00 00 00	00 00 00 00	.Pt..Ъ..........
3f742130	00 00 00 00	22 50 00 82	00 00 00 00	00 00 00 00	...."P.,........

Рис. 8 Частичное представление структуры EPROCESS в образе памяти с выделенными байтами, соответствующими полям Type и Size структуры DISPATCHER_HEADER соответственно

Далее, используя флаг -b, с помощью которого выводится более развернутая информация о структуре, и команду lkd> dt -b nt!_eprocess, удалось определить корректные смещения и для других полей рассматриваемой структуры.

Также, для идентификации структуры EPROCESS, как говорилось ранее, необходимо определить, что следующий и предыдущий потоки процесса принадлежат виртуальном адресному пространству ядра, то есть, в случае 32-битной системы, они должны лежать в диапазоне: 0x80000000 — 0xFFFFFFFF. Однако, было неочевидным, что этот диапазон адресов остался неизменен в рассматриваемой операционной системе. Поэтому пришлось воспользоваться командой lkd> dp nt!mmhighestuseraddress L1. В выводе которой — значение переменной MmHighestUserAddress, в которой содержится максимальный адрес пользовательского пространства (User Space). В Windows 7, 8 и 10 результат оказался одинаковым — 0x7FFEFFFF. Это означает, что пользовательское пространство лежит в адресном

диапазоне 0x00000000 — 0x7FFEFFFF. Соответственно, виртуальное адресное пространство ядра 0x80000000 — 0xFFFFFFFF (где 0xFFFFFFFF — наибольший возможный адрес в 32-битной ОС).

Кроме того, в ходе работы было определено, что в Windows 10 выравнивание параметра Directory Table Base по значению 0x1000 не обязательно.

4.3. Интеграция

Была произведена первая итерация внедрения подсистемы в программный в основные механизмы карвинга[1] данных продукта Belkasoft Evidence Center.

Был исправлен ряд ошибок в функциональности, связанной с обработкой PTE и PDE и с константами, используемыми в случае 32-битной системы с расширением физический адресов. Также, был реализован алгоритм обхода таблиц страниц для получения виртуального адресного пространства процесса и поддержано итерирование, согласно данному алгоритму.

[1] Карвинг — технология поиска данных, позволяющая восстанавливать данные лишь на основе их содержимого, без информации, расположенной в файловой системе

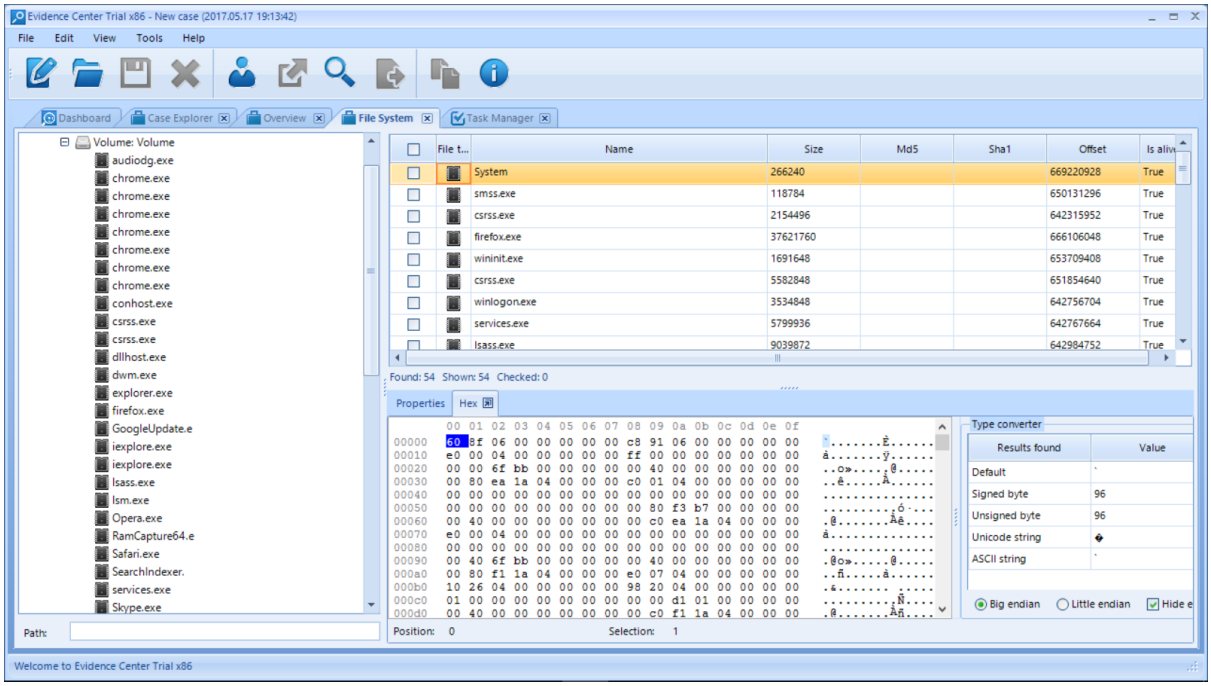


Рис. 9 Снимок экрана восстановленного адресного пространства процессов Windows 10 x86_PAE в продукте Belkasoft Evidence Center

5 Тестирование

Было проведено тестирование с использованием пяти образов памяти Windows 10 с расширением физических адресов, в ходе которого было корректно восстановлено адресное пространство процессов.

Корректность файла с восстановленным адресным пространством процесса проверялась с помощью побайтового сравнения с результатом, полученным с помощью Volatility Framework.

6 Результаты

В рамках данной работы были получены следующие результаты:

- Успешно изучены структуры памяти Windows 10
- Разработан метод восстановления адресного пространства процесса Windows 10
- Реализован предложенный метод в продукте Belkasoft Evidence Center
- Проведено тестирование внедренной системы

Дальнейшее развитие работы предполагает:

- Поддержка восстановления виртуального адресного пространства Windows 10 x86_64
- Углубленное изучение структур, связанных с VAD-дескрипторами
- Обработка недействительных страниц с использованием VAD-дескрипторов
- Оптимизация поиска процессов

Список литературы

[1] A. Schuster, “Searching for Processes and Threads in Microsoft Windows Memory Dumps”, Digital Forensic Research Workshop (DFRWS), 2006.

[2] M. Burdach, “An Introduction to Windows Memory Forensic”, 2005.

[3] B. Dolan-Gavitt, A. Srivastava, P. Traynor, “Robust Signatures for Kernel Data Structures”, 2009.

[4] R. Zhang, L. Wang, S. Zhang, “Windows Memory Analysis Based on KPCR”, Fifth International Conference on Information Assurance and Security, 2009.

[5] Michael Hale Ligh, Steven Adair, Blake Hartstein, Matthew Richard, “Malware Analyst’s Cookbook and DVD: Tools and Techniques for Fighting Malicious Code”, 2011.

[6] Овчинников А. А., “Восстановление адресного пространства процесса из расширенного образа памяти на платформе Windows”, СПбГУ 2012

[7] M. Russinovich, D. Solomon, Windows Internals (“Внутреннее устройство Microsoft Windows”), шестое издание), 2012.

[8] Presentation “Processes, Handles and Tokens”, https://www.tophertimzen.com/resources/cs407/slides/week02_02-Processes.html#slide1

[9] Belkasoft RAM Capturer: Volatile Memory Acquisition Tool, https://belkasoft.com/ram-capturer