Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра системного программирования

Восстановление адресного пространства процесса из расширенного образа памяти на платформе Windows

Курсовая работа студента 445 группы
Овчинникова Антона Андреевича

Научный руководитель
ст. преп. Губанов Ю.А.

Санкт-Петербург
2012

Содержание

Введение

Компьютерный криминалистический анализ (computer forensics) представляет собой комплекс мероприятий, направленных на изучение вычислительных устройств и носителей информации в целях поиска и фиксирования доказательств (например, в ходе расследования гражданских или уголовных дел).

До недавнего времени главным объектом изучения при компьютерном криминалистическом были жесткие диски ([6]). Они являются долгосрочными хранилищами информации, а значит, данные не уничтожаются в течение длительного времени, и сам диск может быть представлен в качестве вещественного доказательства в ходе судебного процесса. С учетом особенностей функционирования наиболее распространенных на данный момент операционных (Windows, Mac OS, GNU/Linux) и файловых (NTFS, ext, HFS) систем, имеется возможность восстанавливать, в том числе, и удаленные с носителя данные, что позволяет проследить хронологию действий, совершенных с носителем. Однако, на данный момент с изъятием и изучением исключительно долгосрочных носителей данных связано несколько проблем, описанных далее.

Проблемы, связанные с анализом жесткого диска

Основные проблемы, с которыми можно столкнуться, опираясь только на анализ жестких дисков:

1. Распространение облачных хранилищ данных.

   Вся информация или ее часть может храниться на удаленных серверах в интернете, и изъятие этих данных может быть существенно осложнено как технически, так и с законодательной точки зрения. Подозреваемый может использовать свой компьютер (ноутбук, мобильное устройство) как терминал для доступа к данным, не сохраняя их на само устройство.

2. Распространение твердотельных накопителей (SSD, Solid State Drive)

   Обладая многими достоинствами по сравнению с обычными жесткими дисками (HDD, Hard Disk Drive), твердотельные диски обладают особенностями функционирования, значительно осложняющими восстановление удаленных файлов. Основная проблема состоит в том, что SSD может самостоятельно очищать блоки памяти, если ОС помечает их как удаленные. Это необходимо для подготовки к записи, потому что ячейки флеш-памяти должны быть очищены перед тем, как на них будут записаны новые данные (см. команду TRIM[1]).

3. Шифрование и запаковка файлов

[1] http://en.wikipedia.org/wiki/TRIM

Использование RAM-дисков, шифрованных контейнеров и разделов существенно препятствует анализу. Похожая ситуация наблюдается и с запакованными и обфусцированными файлами (например, запакованные вредоносные программы). Бурное развитие индустрии вредоносного программного обеспечения приводит к появлению новых, более совершенных методов упаковки (полиморфные упаковщики, протекторы[2]), целью которых является усложнение анализа и методов борьбы с ними.

4. Невозможность получения полного представления о системе

Анализируя жесткий диск компьютера, иногда трудно восстановить полную картину деятельности его обладателя. История браузеров, файлы журналов могут намеренно удаляться для уничтожения такой информации, как посещенные веб-сайты, использованные сетевые соединения и т. п.

В данной работе предлагается использовать анализ оперативной памяти (live memory analysis).

Применение расширенного анализа памяти

Образ (дамп) памяти захватывается на работающей системе (например, с помощью программы FTK Imager [4]), после чего анализируется. В памяти может быть найдено существенное количество информации, которую трудно или невозможно получить, располагая только жестким диском: расшифрованные файлы или пароли, распакованные вредоносные программы, открытые в момент захвата образа сетевые подключения. Например, последнее может помочь в случае, если некто подозревается в управлении ботнетом (сетью из зараженных компьютеров). Более того, в памяти может быть найдена описанная выше информация, которая в момент захвата уже не использовалась. К примеру, можно установить, какие процессы были недавно завершены или какие сетевые подключения были закрыты.

Особый интерес при исследовании процесса представляет его адресное пространство (Рис. 1), восстановление которого позволяет приступить к более тщательному анализу:

[2] Протектор – программа, преобразующая другую программу в вид, который осложняет исследование и отладку путем добавления антиотладочных приемов и шифрования файла.

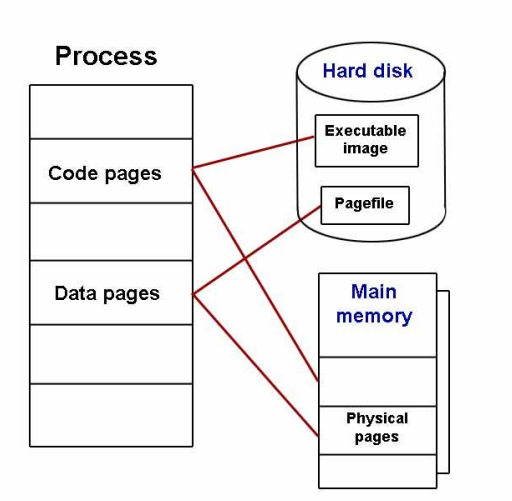

Рис. 1: Отображение адресного пространства процесса

Например, можно привлечь вирусного аналитика для исследования полученного адресного пространства, если есть подозрение, что в контексте процесса исполняется вредоносный код. Однако, обладая исключительно образом памяти, полностью восстановить адресное пространство процесса иногда просто невозможно. На то есть две основные причины:

1. **Файлы подкачки (pagefiles).** Windows использует файлы подкачки для «расширения» физической памяти, перемещая в них данные, не помещающиеся в оперативной памяти.

2. **Отображаемые в память файлы (memory-mapped files).** Это могут быть не только файлы, специально отображенные запущенным процессом, но и части файла образа этого процесса. Например, отображенным на память может быть PE-заголовок программы.

По этим причинам предлагается использовать т.н. «расширенный» образ памяти, то есть образ, дополненный файлом подкачки системы (скопированный, по возможности, в одно время с образом памяти), исполняемым файлом исследуемого процесса и файлами используемых процессом динамических библиотек. Таким образом, появится возможность полностью восстановить пользовательское адресное пространство процесса, так как в наличии будут все составляющие адресного пространства.

Обзор

В этом разделе рассматриваются основные утилиты, которые на используются для анализа образа памяти на сегодняшний день.

Утилиты: PTFinder, Volatility Framework, возможности

До середины 2000-х анализ памяти в основном состоял в использовании утилит strings и grep ([4]). Летом 2005-го года во время конференции Digital Forensic Research Workshop (DFRWS) был устроен конкурс «Memory Analysis Challenge», в ходе которого участникам были предложены два образа памяти с компьютеров под управлением Windows XP. Результатом конкурса стало несколько работ и утилит, посвященных основным подходам к поиску и анализу разнообразных структур в образе памяти.

Первыми значительными работами в этой области также являются [2], [5], [8].

PTFinder

Автор работы [2], Andreas Schuster, во время своих исследований разработал утилиту под названием PTFinder (Process and Thread Finder, последняя версия вышла в ноябре 2007-го года). Основными возможностями программы являются поиск в образе памяти информации о запущенных и недавно завершенных процессах и потоках. PTFinder представляет собой реализацию так называемого «сигнатурного» поиска процессов и потоков в памяти, о котором будет рассказано далее. Утилита написана на языке Perl.

Volatility Framework

На конференции Black Hat в марте 2007 года Aaron Walters и Nick Petroni представили набор утилит, написанных на Python, под названием volatools, предназначенных для разностороннего анализа памяти. После конференции volatools были дополнены и перевыпущены под названием Volatility Framework [11]. Сейчас Volatility поддерживается компанией Volatile Systems и является одним из самых функциональных пакетов для исследования памяти. В его возможности входит извлечение информации о запущенных процессах, открытых сетевых соединениях, открытых файлах, открытых записей реестра, извлечение образов процессов, динамических библиотек и пр. Поддерживается несколько видов сигнатурного поиска (о котором будет сказано далее). На данный момент предоставляется поддержка образов памяти Windows (от Windows XP до Windows 7). Поддержка Linux имеется в виде нестабильной ветки, функций в ней пока предоставляется недостаточно.

Методы поиска процессов и структур в памяти

Для успешного анализа образа памяти необходимо уметь находить структуры EPROCESS (Рис. 2):

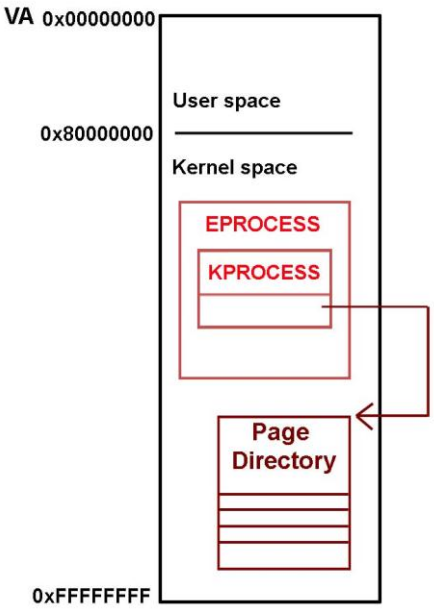

Рис. 2 Структуры EPROCESS и KPROCESS в памяти

В семействе ОС Windows NT данная структура описывает процесс для ядра ОС и находится в адресном пространстве ядра. EPROCESS содержит структуру KPROCESS как подструктуру (KPROCESS представляет собой данные, необходимые для планировщика ОС); также EPROCESS содержит ссылки на другие важные для процесса структуры (такие как список потоков, VAD-дерево, Process Environment Block и т.п.). Также EPROCESS содержит ссылку на директорию страниц процесса, с помощью которой происходит трансляция виртуальных адресов в физические и которая необходима для восстановления адресного пространства процесса.

Чтобы посмотреть содержание этой структуры на конкретной системе, можно воспользоваться WinDbg и командой 'dt _EPROCESS' в режиме отладки ядра.

Существует несколько основных методов, используемых для поиска в дампе памяти структур EPROCESS.

Метод списков (smss.exe, lsass.exe)

В пространстве ядра Windows поддерживает двусвязный список активных (то есть, запущенных в данный момент) процессов. В EPROCESS есть специальные указатели

ProcessListEntry.Flink и ProcessListEntry.Blink, которые указывают на следующий и предыдущий процессы в этом списке по отношению к текущему (Рис. 3):

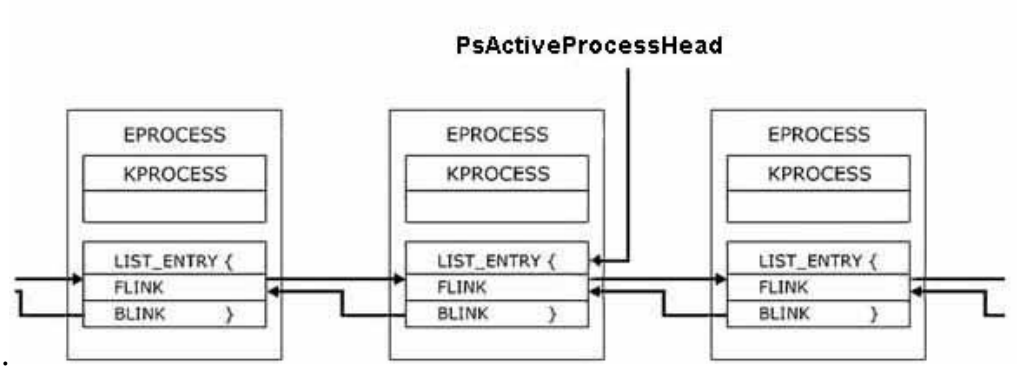

Рис. 3. Голова списка активных процессов

Глобальная переменная пространства ядра PsActiveProcessHead указывает на начало (голову) этого списка. Чтобы найти какой-нибудь элемент списка активных процессов, достаточно обнаружить два «соседних» процесса, которые ссылаются друг на друга. Так как процессы удаляются из списка только в случае завершения и указатели в структуре должны обнулиться, найденные процессы являются активными.

Впервые метод был описан в [8], он основан на поиске связи между процессами smss.exe и csrss.exe. Сначала в образе памяти ищутся две строки ('smss.exe' и 'csrss.exe'), после чего определяется смещение относительно этих строк, по которому должны располагаться поля ProcessListEntry (учитывается тот факт, что имя процесса также можно найти в структуре EPROCESS). Эти два процесса в подавляющем большинстве случаев запускаются последовательно при старте системы, а значит, с высокой вероятностью будут последовательно добавлены в список активных процессов и станут указывать друг на друга. После проверки того, что найдены именно структуры EPROCESS, можно пройти по списку и найти все запущенные в данный момент процессы.

Недостатком данного метода является его ненадежность, если речь идет о поиске руткитов[3], которые для предотвращения обнаружения исключают себя из списка активных процессов. В то же время, руткит имеет возможность продолжать исполнение, так как системный планировщик для исполнения потоков не использует информацию о связности указанного списка ([9]).

Сигнатурный метод

[3] Руткит – программа, основной задачей которой является скрытие следов присутствия в системе себя или другой программы.

Сигнатурный метод состоит в выделении некоторых полей структур, содержание которых подчиняется определенным эвристикам. Например, поле DirectoryTableBase в составе KPROCESS должно содержать адрес в пространстве ядра (> 0x80000000), выровненный по адресам, кратным 0x20; адреса ThreadListHead.Flink и ThreadListHead.Blink должны также указывать на виртуальный адрес в пространстве ядра. Или, например, было замечено, что поля Type и Size в подструктуре DISPATCHER_HEADER должны быть одинаковыми для всех процессов (конкретные значения зависят от версии Winows, для Windows 7 — они составляют 0x03 и 0x26 соответственно). Найдя в памяти места, где эти правила выполняются, и проведя более тщательные проверки (например, проверки принадлежности некоторых адресов пространству ядра), можно убедиться, что найдена именно структура EPROCESS.

Преимуществом данного подхода является то, что при удачном поиске находятся не только активные в данный момент процессы, но и те, которые были завершены в недавнем времени, так как система не производит очистку данных структур при окончании работы процесса.

Главная проблема этого метода заключается в трудности поиска удачных сигнатур. Некоторые поля структуры EPROCESS можно менять, не нанося вреда работоспособности системы, поэтому такие поля не должны входить в сигнатуру, так как руткит, исполняемый в режиме ядра, может специально изменять их значения для предотвращения сигнатурного поиска. Поэтому встает вопрос поиска «устойчивых» сигнатур, которые невозможно изменить без нанесения вреда системе. Найдя такие сигнатуры, можно будет точным образом находить нужные структуры. Хорошее исследование таких сигнатур проведено для Windows XP в [3].

Еще одна проблема состоит в системозависимости: вследствие установки новой версии системы (или даже нового пакета обновлений ОС) значения смещений полей в структуре могут измениться, как и сами значения полей.

KPCR-метод

KPCR (Kernel Processor Control Region, блок управления процессором в ядре) — это специальная структура, описывающая каждый процессор в системе ([10]). Для поиска этой структуры есть довольно устройчивые сигнатуры, которые не менялись с выходом новых версий, начиная с версии Windows XP: по смещению 0x1c от начала структура хранит ссылку на себя, а по смещению 0x20 — ссылку на структуру KPRCB, находящуюся в памяти прямо после KPCR, размер которой фиксирован — 0x120 байт (Рис. 4):

```
lkd> dt nt!_KPcr FFDFF000
nt!_KPCR
   +0x000 NtTib            : _NT_TIB
   +0x01c SelfPcr          : 0xffdff000 _KPCR
   +0x020 Prcb             : 0xffdff120 _KPRCB
   +0x024 Irql             : 0x2 ''
   +0x028 IRR              : 0
   +0x02c IrrActive        : 0
   +0x030 IDR              : 0xffff24e0
   +0x034 KdVersionBlock   : 0x8054d238
   +0x038 IDT              : 0x8003f400 _KIDTENTRY
   +0x03c GDT              : 0x8003f000 _KGDTENTRY
   +0x040 TSS              : 0x80042000 _KTSS
```

Рис. 4 Структура KPCR (из [10])

После обнаружения этой структуры, легко находятся ссылки на списки текущего и ожидающих потоков, то есть определяются адреса структур ETHREAD, описывающих потоки. По ним находятся структуры EPROCESS.

Также из KPCR можно получить ссылки на разные глобальные переменные ядра, вроде адреса системной директории страниц, GDT (global descriptor table) и т.п.

По аналогии с методом списков, с помощью данного метода невозможно найти процессы, которые были исключены из списка активных процессов.

Все три описанных метода в том или ином виде реализованы в таких утилитах для анализа памяти, как Volatility. В рамках данной работы будет использоваться сигнатурный метод как наиболее эффективный и простой в реализации, а также по причине того, что случаи руткитов, намеренно изменяющих сигнатуры, рассматриваться не будут.

Постановка задачи

На данный момент ни одна из имеющихся утилит для работы с образами памяти не поддерживает работу с файлами подкачки и другими файлами, которые можно получить, располагая жестким диском кроме образа памяти. несмотря на то, что их применение может сделать анализ более полным и эффективным.

Поэтому предлагается осуществлять анализ расширенного образа памяти, дополненного файлом подкачки, а также файлами образов процессов и динамических библиотек. Это поможет извлечь не только полные адресные пространства процессов, запущенных в момент снятия образа памяти, но и максимум информации о недавно завершенных процессах, так как области памяти, отвечающие за хранения соответствующих структур могут долгое время не перезаписываться в файле подкачки.

Поставленной задачей является извлечение из расширенного образа памяти адресных пространств конкретных процессов. Для успешного решения этой задачи необходимо:

- найти в образе памяти структуры, отвечающие за описание процесса в адресном пространстве ядра (структуры EPROCESS)
- извлечь из этих структур адреса директорий страниц (Page Directory) и VAD-дерева, необходимых для дальнейших действий
- манипулировать информацией в директории страниц: уметь перебирать все страницы, различать разные виды записей PDE и PTE
- в зависимости от вида записи PDE и PTE использовать разные источники получения страниц (образ памяти, файл подкачки, файл образа процесса и т.п.).

Необходимые сведения о подсистеме памяти Windows

Ниже приведен краткий обзор подсистемы виртуальной памяти Windows. Эти сведения будут необходимы для полного восстановления адресного пространства процесса.

Обзор механизма виртуальной памяти в архитектуре x86

Каждому процессу в ОС Windows выделяется собственное виртуальное адресное пространство размером 4 Гб (случаи x64 и PAE рассмотрены не будут, в некоторой степени они являются слегка усложненными вариантами данного). Виртуальное пространство разделено на страницы, которые могут быть малыми (small pages), размером по четыре килобайта, и большими (large pages), по два мегабайта каждая.

Преобразование, или трансляция, виртуального адреса в физический — важный механизм подсистемы памяти. Для данного преобразования используются директории и таблицы страниц (page directories, page tables).

Трансляция адресов происходит следующим образом:

1. Виртуальный адрес, который необходимо преобразовать в физический, разбивается на три части: индекс в директории страниц, индекс в таблице страниц, смещение в странице (Рис. 5) В момент исполнения процесса, в регистр CR3 процессора загружен физический адрес директории страниц процесса. По индексу, полученному из первой части виртуального адреса (10 бит), из директории страниц берется запись PDE (Page Directory Entry).
2. Из PDE выделяется физический адрес соответствующей таблицы страниц.
3. По индексу, полученному из второй части виртуального адреса (10 бит), выбирается PTE (Page Table Entry)
4. Из PTE выделяется физический адрес соответствующей страницы.
5. По третьей части виртуального адреса (смещению, 12 бит) происходит адресация внутри полученной страницы.

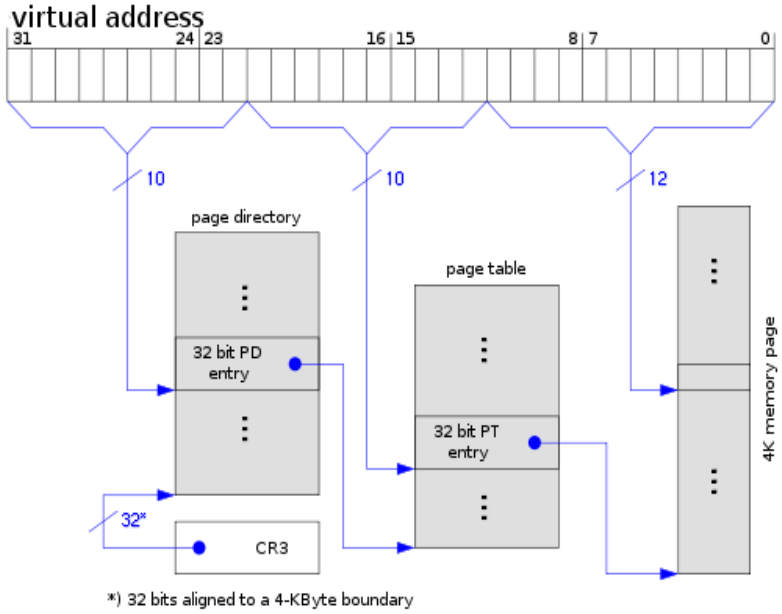

Рис. 5 Трансляция виртуального адреса

Виды PTE (Page Table Entry) и PDE (Page Directory Entry)

Как PTE, так и PDE представляют собой 4-байтные записи в соответствующих таблицах. Большую часть (20 бит) занимают адреса страниц в случае PTE и таблиц страниц в случае PDE, это значение называется PFN (page frame number). Оставшиеся биты — это флаги, в зависимости от которых страницы могут иметь различные состояния (см. Рис. 6).

Рассмотрим разнообразные виды PTE. Если V (бит номер 0) равен 1 в PTE, то соответствующие этому PTE страницы являются действительными (valid), то есть они присутствуют в оперативной памяти. Если PTE является недействительным (V=0), то необходима классификация таких PTE.

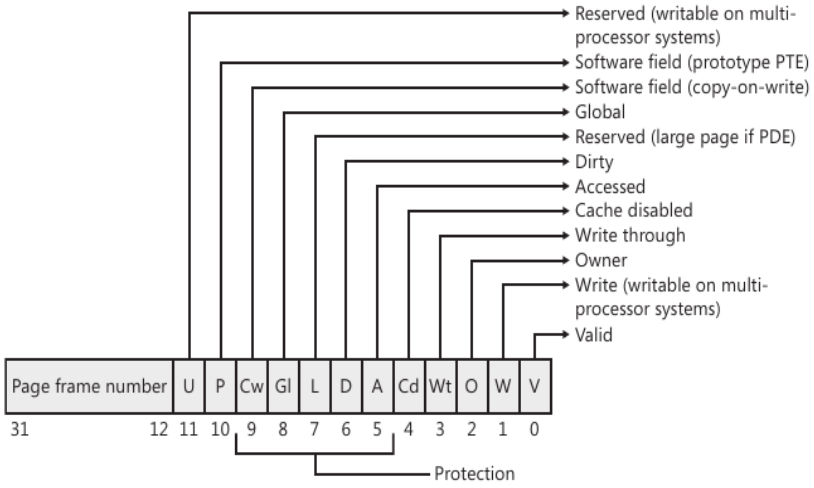

Рис. 6 Формат PTE (из [9])

Типы недействительных PTE (V=0):

Страницы из файла подкачки

У таких PTE V=U=P=0 (биты 0, 10, 11 равны нулю). В таких случаях PFN является смещением искомой страницы в файле подкачки, а биты 1-4 составляют номер файла подкачки (Windows позволяет создавать до 16 файлов подкачки).

Demand Zero PTE

Биты 10-31 и 0-4 равны нулю. Страницы такого типа являются своеобразными «заглушками»: при обращении по этому PTE ОС будет выдавать страницу, полностью состоящую из нулей.

Transition PTE

U=1, P=V=0. Такие страницы присутствуют в оперативной памяти, но находятся в «переходном» состоянии, то есть были изменены, но еще не записаны на диск

Zero

Весь PTE равен нулю. Про страницу почти что ничего не известно: можно рассмотреть соответствующую запись в VAD-дереве (Virtual Address Descriptor tree), чтобы определить статус страницы (к примеру, выделена ли она), но даже в этом случае более содержательной информации получить не удается.

Prototype

P=1, V=0. В этом случае данный PTE является ссылкой на прототипный PTE. Прототипные PTE используются для реализации механизма разделяемой памяти и спроецированных в память файлов. С их помощью одну страницу могут использовать несколько процессов (например, в случае разделяемой библиотеки). Чтобы не обновлять статус страницы в каждом адресном простанстве процесса, который использует эту страницу, сделано так, что PTE разных процессов указывают на один прототипный PTE. Таким образом, при перемещении или удалении страницы нужно обновлять только один прототипный PTE.

Решение

Описываемые исследования сосредоточены на операционных системах семейства Windows NT (в частности, приведенные примеры соответствуют Windows 7 SP1, запущенной на архитектуре x86, без Physical Address Extension).

Предполагается, что в распоряжении имеется образ памяти, снятый с указанной системы, а также файл подкачки (файл pagefile.sys) и файл образа процесса (файл someprogram.exe) с системного раздела жесткого диска.

Получение адресного пространства процесса

Получение полного адресного пространства процесса состоит из нескольких этапов:

1. Необходимо определить, адресное пространство какого процесса хочется получить. Определение осуществляется путем выбора нужной структуры EPROCESS, полученной с помощью одного из описанных методов (в данном случае, используется сигнатурное сканирование).
2. Получение физического адреса (смещения от начала файла образа памяти) директории страниц нужного процесса
3. Перебор всех PDE-записей директории страниц. Если PDE является недействительной, то из нее можно получить физический адрес соответствующей таблицы страниц. Про недействительные PDE- и PTE-записи будет сказано позже.
4. Получив физический адрес таблицы страниц в дампе, можно аналогичным образом перебрать все PTE-записи. Если запись действительная, то из нее можно извлечь физический адрес соответствующей страницы, после чего остается скопировать найденную страницу в выходной файл.

Таким образом, осуществляется обход в ширину директории и таблиц страниц.

Главной трудностью, с которой можно столкнуться при таком подходе, являются недействительные страницы. Если их игнорировать и восстанавливать только действительные, то есть присутствующие в памяти в момент ее захвата страницы, то в некоторых случаях можно упустить более 95% адресного пространства (несбрасываемыми на диск являются таблицы страниц и некоторые структуры ядра, максимум — несколько десятков мегабайт). Недействительные страницы и соответствующие им PTE/PDE бывают нескольких типов, и каждый тип требует своего подхода в обработке. Рассмотрим обработку таких страниц на примере PTE, у PDE классификация такая же.

Обработка недействительных страниц

Страницы из файла подкачки

PTE, ссылающиеся на находящиеся в файле подкачки страницы, содержат номер файла подкачки (по этому номеру можно получить путь к самому файлу, данное соответствие можно найти в реестре) и смещение страницы внутри этого файла.

Demand Zero PTE

При запросах к странице с PTE такого типа операционная система должна возвращать страницу, полностью состоящую из нулей. При обработке дампа памяти можно делать то же самое: считать, что страница данного PTE заполнена одними нулями.

Transition PTE

Состояние страниц с PTE этого типа можно считать актуальным, недействительныйми они считаются только в смысле несогласованности состояния в памяти и на диске. То есть при обработке такие страницы предлагается обрабатывать как действительные.

Zero

PTE этого типа приходится игнорировать ввиду отсутствия какой-либо содержательной информации о состоянии страницы.

Prototype

PTE, ссылающийся на прототипный, содержит адрес этого прототипного PTE. Адрес получается следующим образом: биты 1-8 прототипного PTE являются младшими битами адреса, биты 11-31 — старшими. Далее делается побитовый сдвиг влево получившегося 29-битового значения и получается 31-битовое смещение от 0x80000000, по которому находится искомый прототипный PTE.

Страница, на которую ссылается полученный прототипный PTE, может находиться в одном из шести состояний, напоминающих уже описанные PTE-состояния, за одним главным исключением: установленный бит P указывает на то, что данный прототипный PTE описывает страницу в файле, спроецированном в память (то есть запрещены ссылки одних прототипных PTE на другие).

Обработка прототипных PTE

Ниже представлена классификация прототипных PTE-записей с соответсвующим типом обработки.

- Active (V=1)

Прототипный PTE указывают на действительную страницу в памяти. Аналогично обычным действительным PTE, страница получается по ее смещению.

- Transition (V=0, U=1)

Аналогично непрототипным Transition-PTE.

- Modified No-Write (V=0, U=D=1, т.е. Dirty-бит тоже выставлен)

PTE данного типа указывают на страницы, которые находятся в списке модифицированных, но запрещенных на запись страниц. Обрабатываются аналогично Transition-случаю, смещение такой страницы в файле дампа получается по ее PFN.

- Page File

Делается то же самое, что и в случае обычных Page File PTE.

- Demand Zero

Аналогично непрототипному случаю.

- Mapped File (P=1)

Данный прототипный PTE описывает страницу, которая находится в файле, проецируемом на память. Нужно определить, к какому файлу данная страница относится, а также узнать смещение внутри этого файла, по которому страница располагается.

Организуется следующий порядок действий (Рис. 7):

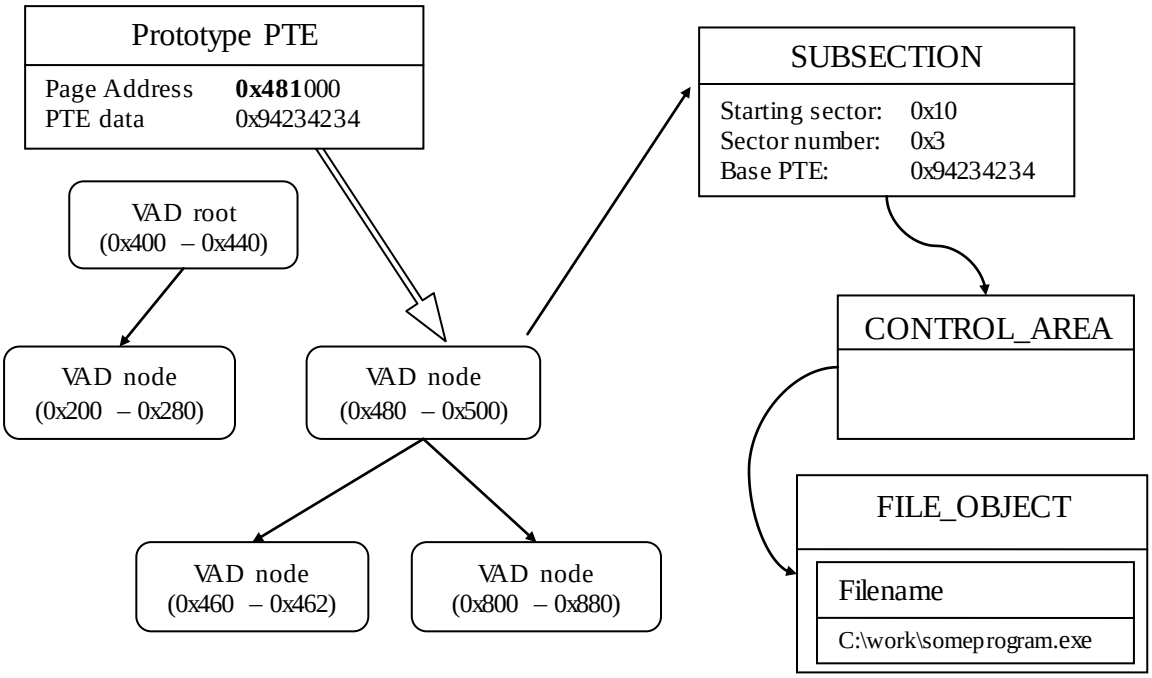

Рис. 7 Нахождение пути к файлу по прототипному PTE

1. Определяется узел VAD-дерева, к диапазону которого принадлежит адрес изучаемой страницы.

2. По узлу и содержанию прототипной PTE-записи определяется нужная структура SUBSECTION, отвечающая за проецирование частей файла на определенные виртуальные адреса.
3. В структуре SUBSECTION есть ссылка на структуру CONTROL_AREA, которая описывает проецирование в память определенного файла. Также из SUBSECTION можно узнать смещение (оно выражено в секторах — блоках данных размером по 512 байт) для страницы, на которую ссылается рассматриваемый прототипный PTE.
4. В CONTROL_AREA имеется ссылка на структуру типа FILE_OBJECT, откуда можно узнать требуемый путь к файлу.

Используя описанный алгоритм, можно получить пути к проецируемым файлам и смещения страниц внутри них. Таким образом, предоставляется возможность скопировать эти файлы с исследуемого компьютера, и данные из них будут использованы в ходе восстановления адресного пространства.

Подводя итог, общую классификацию типов PTE-записей и действий с ними можно изобразить как на Рис. 8:

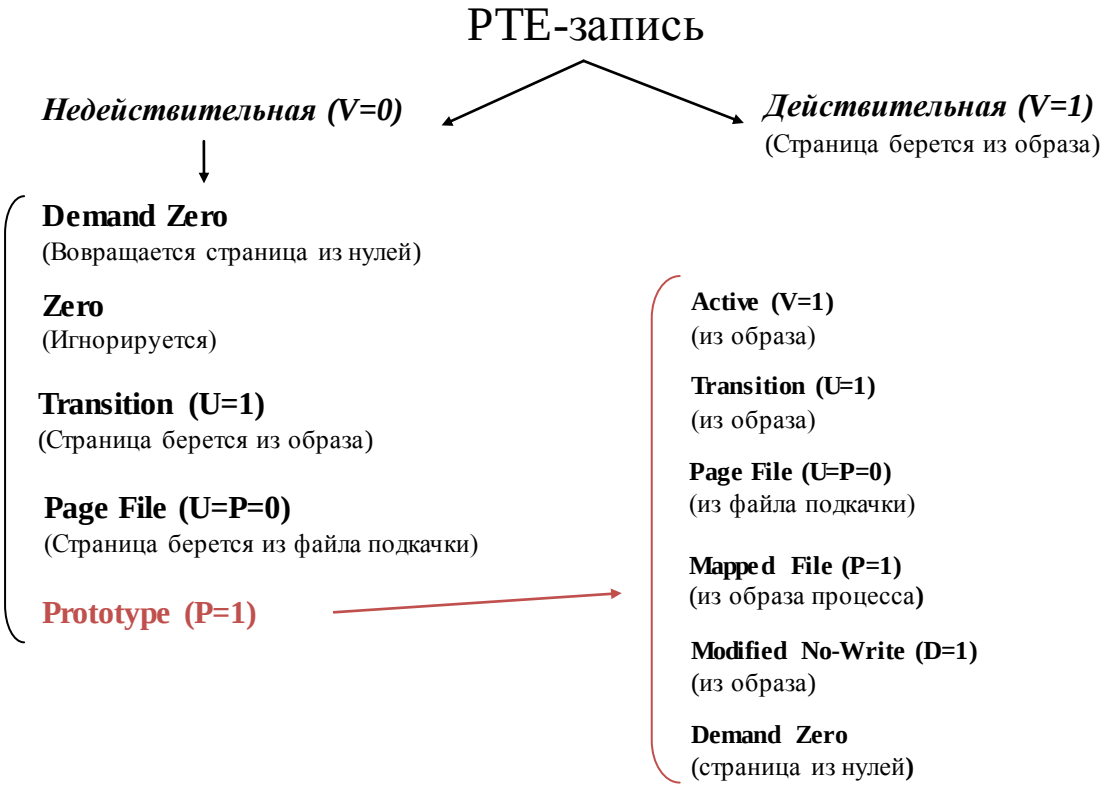

Рис. 8 Источники страниц для различных типов PTE

Программная реализация

Была написана программа на языке Ruby, которая производит:

1. Сигнатурный поиск структур EPROCESS в образе памяти. Программа производит данный тип поиска и выводит все найденные процессы с указанием Process ID, названием процесса, виртуального и физического адреса EPROCESS, а также физического адреса директории страниц.
2. Восстановление адресного пространства для конкретного процесса (указание производится по PID). В ходе этого действия происходит:
    a. Построение VAD-дерева для обработки прототипных PTE
    b. Перебор записей PTE и PDE и выполение соответствующих действий для каждого типа записей, описанных в предыдущей части.

Если обнаруживается, что прототипный PTE указывает на страницу, не присутствующую в памяти, но находящуюся в проецируемом на память файле на диске, то программа выводит путь к этому файлу на системе, с которой образ снят, и запрашивает путь к его копии на текущей системе, который следует прописать в файле конфигурации. При следующем запуске программа будет использовать данный файл как еще один источник страниц.

Получаемые в ходе восстановления страницы программа копирует в отдельный («выходной») файл. Также программа поддерживает индексный файл — вспомогательную структуру, позволяющую понять, какое смещение в получаемом файле соответствует данному виртуальному адресу. Индексный файл представляет собой массив структур, состоящих из двух частей: диапазона и смещения. Если использовать синтаксис языка C, описать данную структуру можно следующим образом:

```
struct INDEX_ENTRY {
  unsigned int STARTVA; // начальный виртуальный адрес диапазона
  unsigned int ENDVA; // конечный виртуальный адрес диапазона
  unsigned int OFFSET; // смещение начала диапазона в выходном файле
}
```

Используя индексный файл, легко можно понять, по какому смещению находятся данные, соответствующие определенному виртуальному адресу, и были ли эти данные вообще восстановлены.

Организация тестирования

Для успешного тестирования, в первую очередь, необходимо уменьшить размер физической памяти, доступной системе, чтобы страницы интенсивнее вытеснялись в файл подкачки. В рамках работы это обеспечивалось тестированием на виртуальной машине.

Для того чтобы протестировать успешность восстановления адресного пространства, нужно выбрать какой-либо процесс, адресным пространством которого можно управлять

(изучать, изменять). Было решено написать программу, удовлетворяющую таким требованиям.

Написанная программа выделяет несколько больших областей памяти (например, по 16 Мегабайт) и заполняет каждый из этих кусков данными, корректность восстановления которых легко проверить. В данном случае всю первую страницу программа заполняет байтами со значением 0, вторую — со значением 1 и т.д. После страницы с номером 256, которая будет заполнена значением 0xFF, процесс повторяется с 0x0 (Рис. 9). После выделения и заполнения памяти описанными значениями, программа останавливается (не завершается), переходя в состояние ожидания ввода пользователя, после чего можно снять дамп памяти и скопировать файл подкачки с системы, остановив сессию виртуальной машины или воспользовавшись специальными утилитами (mdd, FTK Imager для захвата памяти, Fastdump Pro для файла подкачки).

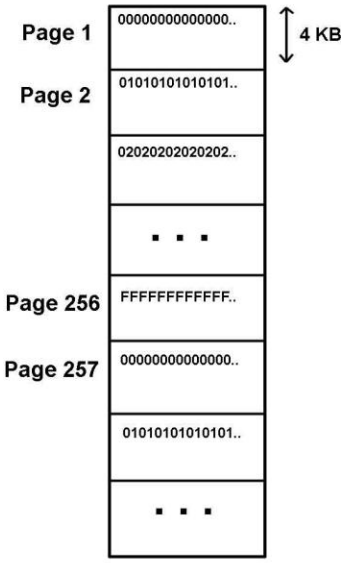

Рис. 9 Заполнение памяти тестовой программой

Стоит отметить, что в реальных условиях ситуация сложнее: использовать сторонние утилиты в системе, с которой требуется снять образ памяти, будет не совсем правильно, так как руткиты могут скрывать свое пребывание в системе, подменяя вызовы системных библиотек и напрямую манипулируя со структурами ядра. Для решения этих проблем существует отдельная область исследований, сосредоточенных на изъятии представляющих интерес данных с помощью аппаратных средств (например, можно использовать Direct Memory Access, подключившись специальным образом через порт FireWire). В рамках данной работы предполагается, что получаемые образы памяти и файла подкачки являются корректными, то есть снятыми с помощью аппаратных средств примерно в одно и то же время (для согласованности).

Результаты тестирования

Написанной программе удалось восстановить 480 мегабайт из 493, которые были выделены операционной системой процессу тестовой программы. То есть, доля восстановленного адресного пространства составляет примерно 97%. При этом на момент захвата образа в памяти находилось примерно 320 мегабайт из выделенных, что говорит о том, что около 160 мегабайт было восстановлено из файла подкачки.

В восстановленном адресном пространстве содержалась вся выделенная тестовой программой память, заполненная специальным образом (как описано в предыдущем разделе). Таким образом, корректность восстановления для этих областей памяти была доказана.

Часть сегментов адресного пространства восстановить не удалось. Если посмотреть на PTE-записи, принадлежащие этим сегментам, в таблице страниц, то можно увидеть, что большинство из этих записей являются Zero PTE. Восстановить данные области памяти в рамках описываемого подхода не удалось.

Тем не менее, проведенное тестирование можно считать успешным, если принять во внимание тот факт, что в некоторых случаях те же самые 95-97% адресного пространства могут быть "сброшены" в файл подкачки, и восстановить их, обладая только образом памяти, не представляется возможным.

Заключение

В работе рассмотрены основные преимущества использования расширенного образа памяти и его использование при анализе для решения практической задачи восстановления адресного пространства процесса. Данный подход может облегчить анализ запакованных или зашифрованных вредоносных файлов, а также может помочь при извлечении из образа информации произвольного типа (изображений, pdf-документов и т.п.).

В ходе проведенного тестирования была подтверждена возможность восстановления большей части адресного пространства процесса.

Дальнейшее развитие

Одним из направлений дальнейшего развития темы может быть извлечение из образа памяти работоспособного исполняемого файла. Полученный таким образом файл может использоваться при динамическом анализе (например, отладке), что в некоторых случаях является иключительно важным моментом для вирусных аналитиков. Обладая только адресным пространством процесса, пусть и проиндексированным, можно проводить только статический анализ.

Еще одним важным направлением анализа памяти является анализ образа, снятого с системы, в котором исполняется виртуальная машина. С развитием средств виртуализации (как программной, так и аппаратной) ситуация, когда вся интересующая деятельность происходит в виртуализированной среде, становится все более частой. Следовательно, необходимо исследование и разработка по возможности общих подходов при анализе памяти для этой ситуации.

Наконец, важным результатом было бы получение устойчивых сигнатур для поиска структур в памяти. Например, если сигнатуры будут опираться на какие-нибудь архитектурные особенности данного семейства операционных систем, не меняющиеся при появлении новых версий и пакетов обновлений, то будет возможность создать более универсальные средства анализа памяти, чем те, которые имеются на сегодняшний день.

Список литературы

[1] A. Martin, “FireWire Memory Dump of a Windows XP Computer: A Forensic Approach”, 2007

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

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

[4] G. L. Garcia, “Forensic physical memory analysis: an overview of tools and techniques”, 2007

[5] J. Kornblum, “Using Every Part of the Buffalo in Windows Memory Analysis”, Digital Investigation Journal, January 2007.

[6] J. Medeiros, “NTFS Forensics: a programmers view of raw file systems data extraction”, 2008

[7] J. Okolica, G. L. Peterson, “Windows operating systems agnostic memory analysis”, Digital Forensic Research Workshop (DFRWS), 2010.

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

[9] M. Russinovich, D. Solomon, Windows Internals (“Внутреннее устройство Microsoft Windows”), четвертое издание), 2009.

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

[11] The Volatility Framework: Volatile memory artifact extraction utility framework, https://www.volatilesystems.com/default/volatility